Advanced Game Theory Applications:
Equilibrium Enumeration for $2 \times 2 \times 2$ Games
and
Empirical Game-Theoretic Analysis of a Pricing Game

Sahar Jahani

A thesis submitted for the degree of
Doctor of Philosophy

Department of Mathematics
The London School of Economics
and Political Science

London, December 2024

Declaration

I certify that the thesis I have presented for examination for the PhD degree of the London School of Economics and Political Science is solely my own work, with the exceptions outlined below.

I declare that this thesis consists of 52,297 words.

Statement of co-authored work

I confirm that all chapters are joint work with my academic supervisor, Professor Bernhard von Stengel.

A preliminary version of Chapter 1 has been published in Jahani and von Stengel [36] (a refereed conference proceedings article).

Abstract

This thesis comprises two main projects. In the first project, we analyse the Nash equilibria of three-player games in their simplest form, known as $2 \times 2 \times 2$ games. We explore the best-response surfaces for each player in detail and develop an algorithm that computes the complete set of Nash equilibrium components, improving upon previous algorithms that only guarantee finding a single equilibrium in these games. In addition to the algorithm, we implemented a software tool that visually represents these games in 3D using best-response surfaces and calculates and displays all Nash equilibria. Another significant contribution of this project is our theoretical proof establishing an upper bound of nine on the number of Nash equilibria in non-generic $2 \times 2 \times 2$ games. Moreover, we extend the definition of degeneracy from the two-player setting to these games and provide directions for extending the upper bound result to the broader class of non-degenerate games.

The second project presents an Empirical Game-Theoretic Analysis (EGTA) of a classical multi-round duopoly pricing game with an infinite strategy space. We employ the Policy-Space Response Oracles (PSRO) framework to iteratively construct a finite approximation of this infinite-strategy game, referred to as the "meta-game." Within this framework, single-agent reinforcement learning (RL) is utilised to compute best-response strategies against Nash equilibria of the evolving meta-game.

Starting with custom implementations of basic RL algorithms, we encountered limitations that motivated the adoption of advanced learning methods better suited to capture the pricing game's complexity. We discuss the strengths and weaknesses of each method applied, alongside techniques developed to enhance their performance. Through comprehensive experiments varying RL algorithms and model specifications, we analyse the resulting meta-games, the estimated equilibria, and the emergent behaviours of RL-trained pricing strategies. We further explore conditions leading to the emergence of collusive behaviour among the trained strategies.

To support future research, we release our framework publicly, facilitating similar analyses by adapting it to other duopoly pricing games. Our framework integrates seamlessly with the advanced RL algorithms provided by the Stable-Baselines3 library and automates the tracking and analysis of learning dynamics, meta-game approximations, and emerging equilibria.

Acknowledgements

I would like to begin by thanking my supervisor, Professor Bernhard von Stengel, for his invaluable guidance and support throughout my research, from whom I have learned so much.

I am also grateful to LSE and its studentship scheme for providing a supportive and inspiring environment that allowed me to focus on my research while developing valuable skills. The opportunity to teach, lead, and collaborate with dedicated professors and talented students has been an enriching experience.

I dedicate a special thank you to my mother, who always encouraged me to pursue higher education. Though she is no longer with us, I hope she would be proud of what I have achieved.

I am deeply thankful to my father and my sister, Sarah, for their unwavering support and encouragement, which have been my point of strength throughout this journey and at every milestone in my life.

A warm thank you goes to my husband, Dominic, for his support, patience, and understanding through all the ups and downs. He has been a constant source of motivation, and I am truly grateful for everything he has done to help me along the way.

Finally, I would like to thank my friends, PhD mates, students, and members of the game theory reading group, who have made this experience truly unforgettable. I'd like to thank Amedeo, Raymond, Justin, Mahsa, Jun, Rennie, Aron, Rebecca, Kashish, Charlotte, Nadia, Pouya, Meshkat, Ed, Francisco, Johannes, Domenico, Galit, Katerina and Rahul, with whom I have made many wonderful memories.

I would also like to thank the contributors of OpenAI's ChatGPT tool, which was very helpful for grammar checks, typesetting issues, and debugging code.

Contents

Part I.

Equilibrium Enumeration in $2 \times 2 \times 2$ Games

Introduction to Part I

Game theory provides mathematical models for multi-agent interactions. The primary solution concept is Nash equilibrium, and its refinements (e.g., perfect equilibrium [61]) or generalisations such as correlated equilibrium [2] (which arises from regret-based learning algorithms). For two-player zero-sum games, finding a Nash equilibrium can be achieved by solving a linear programming (LP) problem [48]. However, this is not the case for non-zero-sum games or games with more than two players. Already for general-sum two-player games, finding just one Nash equilibrium is PPAD-hard [10, 12].

However, this "intractability" of the Nash equilibrium concept applies to *large* games. Many games that are used as economic models are small, with fewer than a dozen payoff parameters, and often given in extensive form as game trees. It would be desirable to have a *complete* analysis of *all* Nash equilibria of such a game, in order to study the implications of the model. Such a complete analysis is known for two-player games. Their Nash equilibria can be represented as unions of "maximal Nash subsets" [74]. These are maximally "exchangeable" Nash equilibrium sets, that is, products of two polytopes of mixed strategies that are mutual best responses. Their non-disjoint unions form the topologically connected components of Nash equilibria, and are computed by the *lrsNash* algorithm of Avis, Rosenberg, Savani, and von Stengel [3], which works well for games with up to about twenty strategies per player.

For games with more than two players, the set of all Nash equilibria cannot be described in such a way, because it is determined by equations and inequalities between nonlinear polynomials. The Gambit software package [57] provides access to polynomial solvers in order to compute the Nash equilibria of generic games. "Generic" means that the payoffs do not represent edge cases. The edge cases can be encoded as the zeros of a suitable polynomial in the game parameters and form a set of measure zero. Generic games have only finitely many equilibrium points. Non-generic games can have infinite sets of equilibria.

However, rather remarkably, to our knowledge, there is no algorithm that computes (in some description) the entire set of Nash equilibria for even the simplest game with more than two players if the game is non-generic, which naturally occurs for games in extensive form, such as "Selten's horse" [61]; see Section 1.4.1 below.

This work describes an algorithm that computes the entire set of Nash equilibria for arbitrary $2 \times 2 \times 2$-games, that is, three-player games where every player has two strategies. These are the simplest games with more than two players that do not have a special structure

(such as being a polymatrix game arising from pairwise interactions, see [31]). While this seems like a straightforward task, it is already challenging in its complexity.

One contribution of our work is to reduce this complexity by carefully preserving the symmetry among the players, and a judicious use of intermediate parameters (see (1.10) in Section 1.3.2) derived from the payoffs. We define the notion of *non-degeneracy* in $2 \times 2 \times 2$ games, which implies the game's genericity and allows us to exclude edge cases through conditions imposed on the payoff parameters. We determine a quadratic equation (see (1.20)) that has a regular structure using determinants (not known to us before), which also implies that a generic $2 \times 2 \times 2$ game has at most two completely mixed equilibria (shown much more simply than in [11] or [44]). The standard approach to manipulating such complicated algebraic expressions is to use a computer algebra system [13].

As a "binary" game with only two pure strategies per player, the equilibria of a $2 \times 2 \times 2$ game can be visualised as points and surfaces inside a cube. However, making such visualisations accessible requires 3D graphics.

We believe that effective visualisations of the geometry of a game's equilibrium solutions are essential for understanding their structure and properties, both for practical applications and theoretical research in game theory.

To this end, we have implemented equilibrium computation algorithms (as discussed in Section 1.3), along with graphical representations for $2 \times 2 \times 2$ games, using Python. Our tool is publicly available at [34]. Given a $2 \times 2 \times 2$ game as input, the application generates an interactive 3D visualisation of all best-response correspondences and equilibria, along with a text output highlighting the key features of the game and its equilibrium structure.

All figures presented in Chapters 1 and 2 were generated using our tool. Throughout these chapters, we reference a test number for each figure, which can be used to regenerate the corresponding graphics and game descriptions by running our code with the predefined test case.

Furthermore, numerous studies have explored the structure of equilibria in generic games (e.g., [44], [29]). Upper bounds on the number of different types of equilibria have been studied for bimatrix games [32], as well as for various forms of multiplayer games [28]. Our representation of $2 \times 2 \times 2$ games provides a deeper understanding of the equilibria in such games.

Another significant contribution of this work is the proof of an upper bound of nine for the total number of equilibria in generic $2 \times 2 \times 2$ games. One part of this proof, presented in Section 2.2.1, relies on prior results regarding the concept of equilibrium *index* in generic games to show the impossibility of a game having the case of 4 pure, 6 partially mixed, and 1 completely mixed equilibria. We also discuss that the rest of our proof of the nine-equilibria bound applies to the broader class of non-degenerate games. To extend this upper bound to non-degenerate games, further properties of the equilibrium index for this class need to be established. Since the equilibrium index has a rather complex definition for games with more than two players, we intend to specialise this definition for $2 \times 2 \times 2$ games, where it should be simpler, and to further complete our proof for non-degenerate games.

In Chapter 1, we begin by formulating $2\times2\times2$ games using a symmetric parametrisation for the players. We then study the best-response correspondences of the players and the various forms these can take. We describe the edge cases in such games and extend the previous definition of degeneracy from two-player games to $2 \times 2 \times 2$ games in order to identify these edge cases.

Next, we proceed to enumerate the equilibria of general $2 \times 2 \times 2$ games using two algorithms. The first algorithm identifies partially mixed equilibria (located on the faces or edges of the cube of mixed strategies), which arise from the equilibria of reduced two-player games where the third player plays a pure strategy that remains optimal. This method generalises straightforwardly to games with a larger number of players, each with two strategies, and may be particularly useful for preliminary analysis of such games.

The second part focuses on computing completely mixed equilibria, which is more challenging and does not generalise as easily. While a substantial portion of the algorithm is outlined, we do not describe it in full due to the large number of case distinctions required to systematically handle non-generic scenarios (which can still arise in game trees even when payoffs are generic). Both algorithms discussed have been implemented in our Python tool.

Our formulation of the $2 \times 2 \times 2$ game in Chapter 1 corrects and extends the earlier version presented in the unpublished undergraduate thesis by Zhu [76] at LSE.

In Chapter 2, we focus on non-degenerate and generic games, which can only have a finite number of equilibria. We prove an upper bound on the number of Nash equilibria in generic games, and outline the steps needed to generalise this proof to the broader class of non-degenerate games.

In Theorem 2.9, we establish that generic $2 \times 2 \times 2$ games can have at most nine equilibria, and that this maximum can only occur in one of the following two configurations:

- 4 pure, 3 partially mixed, and 2 completely mixed equilibria;
- 4 pure, 4 partially mixed, and 1 completely mixed equilibrium.

This result fully establishes the cases above for both classes of generic and non-degenerate games. However, our argument for excluding the configuration with 4 pure, 6 partially mixed, and 1 completely mixed equilibrium relies on prior work involving the concept of the *equilibrium index*, which has so far been established only for generic games.

In generic games, each Nash equilibrium is isolated and has an index of either $+1$ or -1, with the total sum of indices equal to 1. Pure equilibria always have index $+1$, while the index of a partially mixed equilibrium equals that of the corresponding 2×2 subgame (in which the third player plays a pure strategy).

To extend these index-based arguments to non-degenerate games, we need to adapt and simplify the definition of equilibrium index specifically for $2 \times 2 \times 2$ games. Currently, the definition for games with more than two players relies on advanced topological results (see, e.g., Ritzberger [55]), which we believe can be significantly simplified in the special case of the $2 \times 2 \times 2$ games considered in this work. Such an adaptation and verification of the

corresponding index properties would allow us to extend the upper bound of nine equilibria to all non-degenerate games.

Furthermore, our result confirms the upper bound conjectured by Vujić [69] for m-player games with two strategies per player. For the case $m = 3$, studied here, our work provides a proof of the conjectured maximum of nine equilibria.

Finally, we note that these results do not extend to degenerate games, which can have sets of infinite number of equilibria forming components of various dimensions. Such equilibrium components can be one-dimensional (lines or curves), two-dimensional surfaces, or even three-dimensional regions (e.g., the entire cube when all payoffs are zero). A complete algorithmic framework for analysing degenerate games is developed in Chapter 1.

1. Equilibrium Enumeration for $2 \times 2 \times 2$ Games

The set of all Nash equilibria of a non-cooperative game with more than two players is defined by equations and inequalities involving nonlinear polynomials, which makes it challenging to compute. This chapter presents an algorithm to compute this set for the simplest game with more than two players and arbitrary (possibly non-generic) payoffs, a task that has not been done before.

We provide new, elegant formulas for completely mixed equilibria, as well as an algorithm to compute partially mixed equilibria by analysing the games between each pair of players. Furthermore, we compute and visualise the best-response correspondences and their intersections in 3D, which define the Nash equilibrium set.

These computations and visualisations have been implemented in Python and will be part of a public, web-based software tool for automated equilibrium analysis.

For small games, which are often studied in economic models, a complete Nash equilibrium analysis is desirable and should be feasible. This project demonstrates the difficulties of this task and offers pathways for extensions to larger games.

1.1. Background

This section introduces the notations that will be used in subsequent sections and the proofs of the next chapter.

1.1.1. General Form of a $2 \times 2 \times 2$ Game

The following table describes a normalised form of a three-player game in which each player has two strategies:

I \ III, II	Left: $1-q$	Right: q
Up: $1-p$	0 0 0	0 a 0
Down: p	0 0 A	0 c B

Front: $1-r$

I \ III, II	Left: $1-q$	Right: q
Up: $1-p$	α 0 0	γ b 0
Down: p	β 0 C	δ d D

Back: r

(1.1)

This game is played by players I, II, III, choosing (simultaneously) their second strategy with probability p, q, r, respectively. Player I chooses a row, either Up or Down (abbreviated U and D), player II chooses a column, either Left or Right (abbreviated L and R), and player III chooses a panel, either Front or Back (abbreviated F and B). The strategy names are also chosen to remember the six faces of the three-dimensional unit cube of mixed-strategy profiles (p, q, r), shown in Figure 1.1. The coordinates of each point on this cube represent the probability of each player choosing the second strategy.

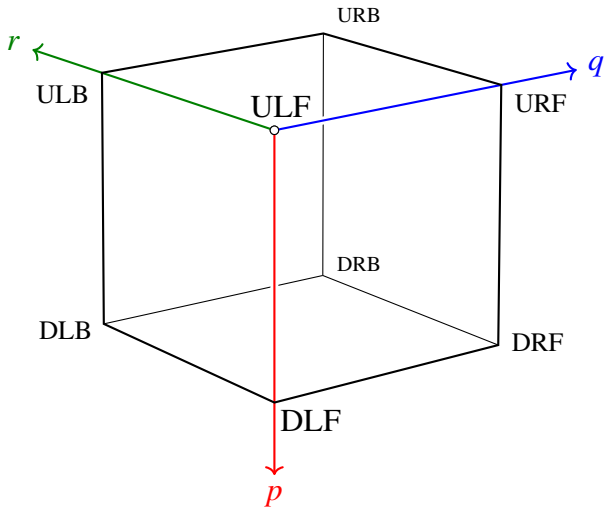


Figure 1.1.: Cube of mixed-strategy probabilities (p, q, r) depicted with pure-strategy names abbreviated from (1.1); for instance, DRB stands for the strategy profile (Down, Right, Back)

Each of the eight cells in (1.1) has a payoff triple (T, t, τ) to the three players, with the payoffs to player I, II, III in upper case, lower case, and Greek letters, respectively. The payoffs in (1.1) are staggered and shown in colours to distinguish them more easily between the players.

The payoffs have been normalised so that each player's first pure strategy has payoff of zero throughout.

This normalisation is obtained by subtracting a suitable constant from the player's payoffs for each combination of opponent strategies (e.g., each column for player I). This does not affect best responses [67, p. 239]. With this normalisation, the first strategy of each player always gives expected payoff zero, which makes the subsequent best-response analysis more straightforward.

For each player's second strategy, the expected payoffs are as follows:

$$\begin{aligned}
\text{player I}: \quad S(q,r) &= (1-q)(1-r)\,A + q(1-r)\,B + (1-q)r\,C + qr\,D, \\
\text{player II}: \quad s(r,p) &= (1-r)(1-p)\,a + r(1-p)\,b + (1-r)p\,c + rp\,d, \\
\text{player III}: \quad \sigma(p,q) &= (1-p)(1-q)\,\alpha + p(1-q)\,\beta + (1-p)q\,\gamma + pq\,\delta,
\end{aligned} \tag{1.2}$$

where the payoff parameters A, B, C, D are the payoffs to player 1 for the four combinations of pure strategies played by the other players. Similarly, a, b, c, d represent the payoffs to player 2, and $\alpha, \beta, \gamma, \delta$ are the payoffs to player 3, as shown in the payoff matrices (1.1).

With this definition, the three players can be treated symmetrically. The cyclic shifts among p, q, r in (1.2), along with the corresponding choices of where to place b, c, β, and γ in (1.1), lead to more symmetric solutions.

The mixed-strategy profile (p, q, r) is a *mixed equilibrium* if each player's mixed strategy is a *best response* against the other players' strategies. That best response is a pure (deterministic) strategy, unless the two pure strategies have *equal* expected payoffs [47, p. 287]. Hence, p is a best response of player I to (q, r) if the following conditions hold:

$$P(q,r) = \begin{cases} p = 0 & \text{if} \quad S(q,r) < 0 \\ p \in [0,1] & \text{if} \quad S(q,r) = 0 \\ p = 1 & \text{if} \quad S(q,r) > 0\,. \end{cases} \tag{1.3}$$

We designate the $q \times r$ plane as the *base plane* for player I. Similarly, q is a best response of player II to (r, p) and r is a best response of player III to (p, q) if and only if

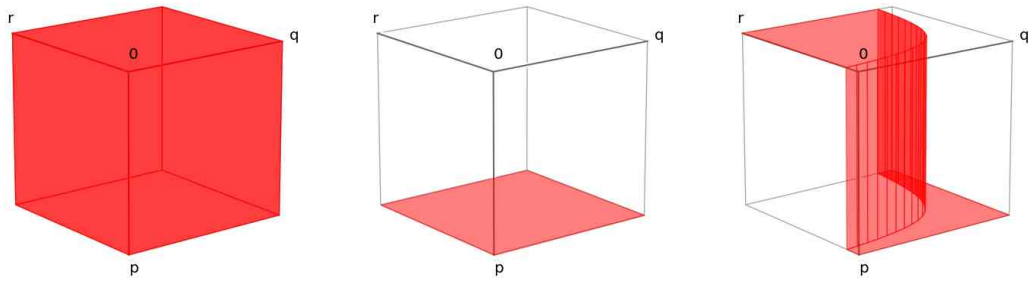
$$Q(r,p) = \begin{cases} q = 0 & \text{if} \quad s(r,p) < 0 \\ q \in [0,1] & \text{if} \quad s(r,p) = 0 \\ q = 1 & \text{if} \quad s(r,p) > 0 \end{cases} \qquad R(p,q) = \begin{cases} r = 0 & \text{if} \quad \sigma(p,q) < 0 \\ r \in [0,1] & \text{if} \quad \sigma(p,q) = 0 \\ r = 1 & \text{if} \quad \sigma(p,q) > 0\,. \end{cases} \tag{1.4}$$

Similarly, $r \times p$ and $p \times q$ planes are called the base planes for players II and III, respectively. For each player I, II, or III, the triples (p, q, r) that fulfil the respective conditions for p, q, or r in (1.3) and (1.4) define the *best-response correspondence (BR)* of that player, a subset of the cube $[0, 1]^3$.

$$\begin{aligned}
BR_1 &= \{\ (p,q,r) \in [0,1]^3 \ \mid\ p \in P(q,r)\ \}, \\
BR_2 &= \{\ (p,q,r) \in [0,1]^3 \ \mid\ q \in Q(r,p)\ \}, \\
BR_3 &= \{\ (p,q,r) \in [0,1]^3 \ \mid\ r \in R(p,q)\ \}.
\end{aligned} \tag{1.5}$$

The set of *Nash equilibria* is defined as the intersection of the best-response correspondences (BRs) of all players. Therefore, before analysing the structure of Nash equilibria, we first examine the different possible forms that a best-response correspondence can take.

For instance, player I's best-response correspondence can exhibit several distinct forms, as shown in Figure 1.2. In particular, the vertical surface in panel (c) can itself take on various shapes, which we study in detail later.



(a) (b) (c)

Figure 1.2.: Different forms of best-response correspondence.

- (a) If $A = B = C = D = 0$, then $S(q, r) = 0$ for all $q, r \in [0, 1]$, and player I's best-response correspondence is the entire cube of $[0, 1]^3$. This illustrates a case where the player is completely indifferent between their two strategies. Such a situation is an edge case that can lead to undesirable sets of equilibria. We say it makes the game *degenerate*, and we exclude these games later, as explained in Section 1.2.
- (b) If $A, B, C, D < 0$, then $S(q, r) < 0$ for all $(q, r) \in [0, 1]^2$ and strategy Up strictly dominates Down, so that player I will always prefers to play Up, and the game reduces to a two-player game between players II and III. The same happens when $A, B, C, D > 0$, in which case Down strictly dominates Up. In these two cases, the best-response correspondence of player I is the upwards "Up face" or downwards "Down face" of the cube (as in Figure 1.2(b)), respectively.
- (c) In all other cases, the best response of player I to (q, r) is sometimes Up and sometimes Down. The best-response correspondence of player I is then a surface that consists of subsets of the Up or Down face according to (1.3), which are connected by vertical parts, as in Figure 1.2(c) where player I is indifferent between Up and Down. The shape of the indifference surface will be discussed comprehensively in Section 1.1.4.

Figure 1.3 shows the best-response correspondences of all three players in a $2 \times 2 \times 2$ game. The first player's best-response correspondence is plotted in red, the second player's in blue, and the third player's in green. In the top-left plot, all best-response correspondences are displayed together. Additionally, the Nash equilibria are marked in black. We elaborate further on our 3D representation in Section 1.4.

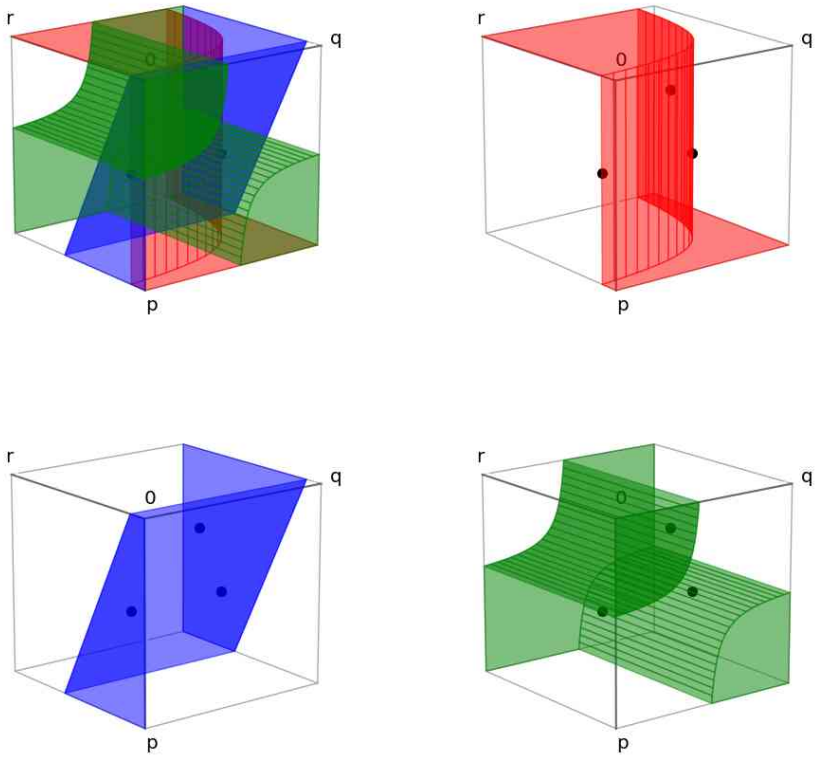


Figure 1.3.: Example of best-response surfaces of a game with two completely mixed equilibria and one partially mixed equilibrium, marked by black dots. (This example corresponds to Test 10 in our Python code, which can be visualised as a 3D animation and explored interactively.)

1.1.2. Components of Best Response Correspondence

Building up on the formulation in (1.3) and (1.4), the best response correspondence of each player can be represented as the union of two sets of surfaces:

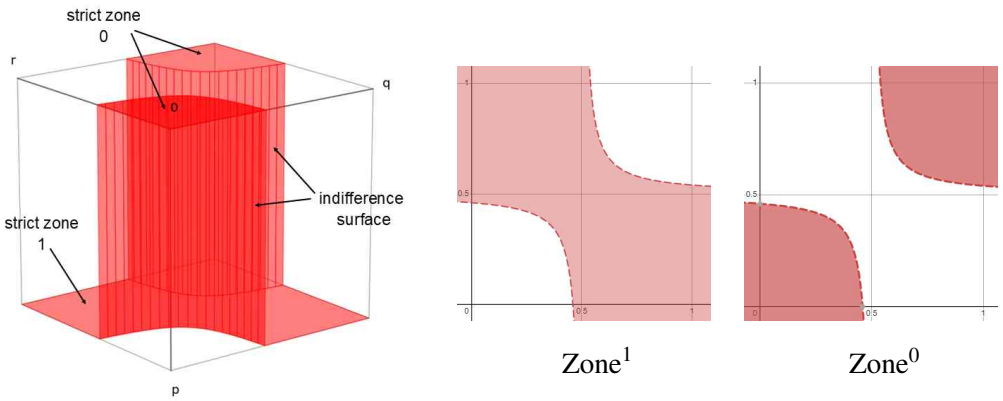
- Indifference Surfaces (IS): the set of points in the cube where the expected payoff is equal to zero. For instance, for player I the indifference surfaces are identified as follows:

$$\mathrm{IS}_1 = \{(p, q, r) \in [0, 1]^3 \mid S(q, r) = 0\}. \tag{1.6}$$

- Strict Surfaces (SS): the set of points in the cube where the expected payoff is positive or negative. For player I, the strict surfaces are defined as follows:

$$\mathrm{SS}_1 = \{(1, q, r) \in [0, 1]^3 \mid S(q, r) > 0\} \cup \{(0, q, r) \in [0, 1]^3 \mid S(q, r) < 0\}\,. \tag{1.7}$$

Hence, the best-response correspondence for each player i is given by $\mathrm{BR}_i = \mathrm{IS}_i \cup \mathrm{SS}_i$.



Player I's best response surface

Figure 1.4.: Components of best-response correspondence of player I.

Another concept aiding in the visualisation of best response surfaces, and later in the proofs, is that of strict zones. The strict surfaces projected on the base plane consist of two regions referred to as *Strict Zones*. Zone^0 includes points where the expected payoff is negative; therefore, the best response is 0, while Zone^1 comprises points where the expected payoff is positive. Strict zones are two-dimensional surfaces defined on the base plane. For instance, for the first player, the zones are defined as follows:

$$\text{Zone}^0 = \{(q,r) \in [0,1]^2 \mid S(q,r) < 0\}, \quad \text{Zone}^1 = \{(q,r) \in [0,1]^2 \mid S(q,r) > 0\}.$$

The strict zones are divided by indifference surfaces and may take different forms depending on the shape of the indifference surface. By definition, it is evident that strict zones do not intersect. Furthermore, each zone forms part of the best-response correspondence in three dimensions as a strict surface with a fixed third coordinate,0 for Zone^0 and 1 for Zone^1. Consequently, the two distinct strict surfaces are located on different faces of the cube. These surfaces are depicted in Figure 1.4 for an example of the first player's best-response correspondence.

1.1.3. Formulation of Hyperbolas

One of the common forms that an indifference surface can take when it is projected on the base plane is a hyperbola.

Definition 1.1. In a two-dimensional setting, a *hyperbola* is the set of all points for which the absolute difference of their Euclidean distances from two fixed points is constant. A hyperbola consists of two continuous, disconnected curves called "branches". These branches approach two lines, known as asymptotes, but intersect them only at infinity.

The general equation for a hyperbola with asymptotes parallel to the coordinate axes in the $x \times y$ plane is given by:

$$(x-a)(y-b) = c \tag{1.8}$$

in which the lines $x = a$ and $y = b$ are the asymptotes.

In this thesis, the hyperbolas we study all have asymptotes parallel to the rectangular axes and follow the formulation in (1.8).

If $c = 0$ in (1.8), the hyperbola is called *degenerate*, as the branches become the asymptotes. Otherwise, it is called *non-degenerate*.

Hyperbolas have interesting properties that assist us later in our proofs:

- The branches of a hyperbola approach the asymptotes and get arbitrarily close to them but never intersect them, except at infinity.
- The branches are located on different sides of both asymptotes. Consequently, the domain and range of the branches are divided by the asymptotes.
- Hyperbolas exhibit monotonic behaviour, except in between the two branches, where there is a jump in the function. This means the branches are either both increasing or both decreasing.

1.1.4. Classification of Indifference Surfaces

To analyse Nash equilibria, we begin by examining the different forms that indifference surfaces can take, as their intersections correspond to equilibria within the interior of the $[0, 1]^3$ cube. As defined in (1.6), the indifference surface IS_i consists of the strategy profiles for which the expected payoff function of player i is equal to zero. We can rewrite the expected payoff equations (1.2) as:

$$\begin{aligned} S(q, r) &= A + Kq + Lr + Mqr \\ s(r, p) &= a + kr + lp + mrp \\ \sigma(p, q) &= \alpha + \kappa p + \lambda q + \mu pq \end{aligned} \tag{1.9}$$

with

$$\begin{aligned} K &= B - A\,, & L &= C - A\,, & M &= A - B - C + D\,, \\ k &= b - a\,, & l &= c - a\,, & m &= a - b - c + d\,, \\ \kappa &= \beta - \alpha\,, & \lambda &= \gamma - \alpha\,, & \mu &= \alpha - \beta - \gamma + \delta\,. \end{aligned} \tag{1.10}$$

We now analyse the indifference surface of player I; the analysis for the other players is analogous.

Figure 1.2 already illustrates examples of best-response correspondences:

(a) If $(A, B, C, D) = (0, 0, 0, 0)$, then the best-response correspondence (BR) is the whole cube.

(b) If A, B, C, D are all positive or all negative, then the BR is only one face of the cube.

We now focus on the form of IS_1 in part (c).

Using (1.10) and (1.9), the indifference condition $S(q, r) = 0$ defines the surface

$$IS_1 = \{(p, q, r) \in [0, 1]^3 \mid S(q, r) = A + Kq + Lr + Mqr = 0\}, \tag{1.11}$$

where IS_1 can take different forms in the $q \times r$ base plane depending on the values of the parameters M, K, L, and A.

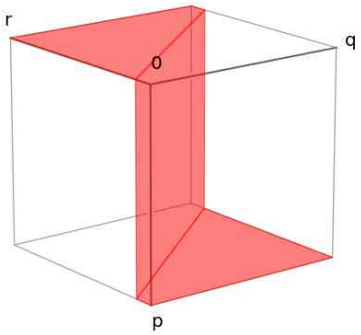
(i) Linear case: If $M = A - B - C + D = 0$, then

$$A + Kq + Lr = 0. \tag{1.12}$$

If $K = L = 0$, then $A = B = C = D$, and either case (a) or (b) from above applies; i.e., (1.12) has either infinitely many or no solutions. So let us assume $(K, L) \neq (0, 0)$.

If $K = 0$, the line is defined by a constant value of r, namely $r = -A/L$. If $L = 0$, the line is defined by a constant value of q, namely $q = -A/K$. Otherwise, (1.12) defines a standard linear relationship between q and r.

In all cases, the indifference surface is a vertical plane in the p-axis direction, extending a line in the $q \times r$ base plane. According to (1.3), on this indifference plane, Player I is indifferent between the first and second strategies. For points on each side of the plane, the best response is $p = 0$ or $p = 1$ depending on the sign of $S(q, r)$.



(ii) Hyperbolic case: Suppose $M \neq 0$. Then (1.11) is equivalent to:

$$qr + \frac{K}{M}q + \frac{L}{M}r + \frac{A}{M} = 0. \tag{1.13}$$

Adding $\frac{KL}{M^2} - \frac{A}{M}$ to both sides and using (1.10) gives:

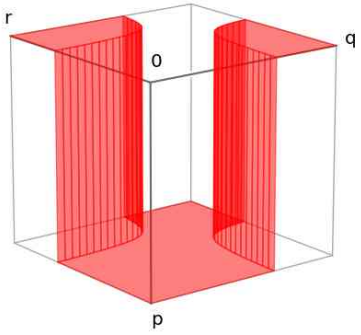
$$\left(q + \frac{L}{M}\right)\left(r + \frac{K}{M}\right) = \frac{KL - AM}{M^2} = \frac{BC - AD}{M^2}, \tag{1.14}$$

which describes a hyperbola in the $q \times r$ plane with asymptotes $q = -\frac{L}{M}$ and $r = -\frac{K}{M}$ (see Definition 1.1).

If $BC - AD \neq 0$, this defines a non-degenerate hyperbola with two branches. Depending on the values of A, B, C, D, the $[0, 1] \times [0, 1]$ square may contain both branches, one branch (as in the green and red best-response surfaces in Figure 1.3), or neither (e.g., when the game has a dominated strategy, case (b) above).

For points (q, r) not lying on the hyperbola, Player I's pure best response is determined by the sign of $S(q, r)$. These form the strict surfaces. Note that when the equality in (1.11) is replaced with '<" or >", if $M < 0$, the inequalities in (1.13) will be reversed.

In this form, the points in the interior of the two hyperbola branches have the same best response and, consequently, lie on the same strict best-response surface. In contrast, the points located between the two branches have a different best response and are situated on the opposite face of the cube.

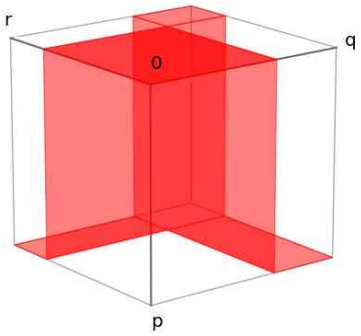


(iii) Degenerate hyperbolic case: If $M \neq 0$ and $BC - AD = 0$ in (1.14) above, then the indifference condition reduces to

$$q = -\frac{L}{M} \quad \cup \quad r = -\frac{K}{M} \tag{1.15}$$

which defines two perpendicular lines on the base plane. This case corresponds to a degenerate hyperbola on the base plane, extended in the p direction. It can be interpreted as the limiting case of the previous scenario, where the hyperbola branches converge to their asymptotes.

The blue best response in Figure 1.8 is an example of this case for player II (extended in the q direction).



1.1.5. Solving Indifference Equations System

In Section 1.1.4, we explained the different forms that indifference surfaces can take. Equation (1.9) shows the system of equations for all indifference surfaces of all players. This system consists of three equations with three parameters, representing the mixed strategies

of each player. To compute the intersections of these surfaces, we proceed to solve this system as follows:

The expressions in (1.9) are linear in each of p, q, r, and we consider when they are equal to zero, which defines the indifference surfaces:

$$\begin{aligned} A \;+\; Kq \;+\; (L + Mq)r \;&=\; 0 \\ a \;+\; lp \;+\; (k + mp)r \;&=\; 0 \\ \alpha \;+\; \kappa p \;+\; (\lambda + \mu p)q \;&=\; 0. \end{aligned} \tag{1.16}$$

We first eliminate r by multiplying the first equation in (1.16) by $(k + mp)$ and the second by $-(L + Mq)$, and then adding the resulting expressions. This gives:

$$(A + Kq)(k + mp) - (L + Mq)(a + lp) = 0, \tag{1.17}$$

or, using determinants:

$$\begin{vmatrix} A & L \\ a & k \end{vmatrix} + \begin{vmatrix} A & L \\ l & m \end{vmatrix} p + \begin{vmatrix} K & M \\ a & k \end{vmatrix} q + \begin{vmatrix} K & M \\ l & m \end{vmatrix} pq = 0. \tag{1.18}$$

In the same way, we eliminate q by multiplying the last equation in (1.16) by $\begin{vmatrix} K & M \\ a & k \end{vmatrix} + \begin{vmatrix} K & M \\ l & m \end{vmatrix} p$ and equation (1.18) by $-(\lambda + \mu p)$, then summing the resulting expressions. This gives:

$$\left(\begin{vmatrix} K & M \\ a & k \end{vmatrix} + \begin{vmatrix} K & M \\ l & m \end{vmatrix} p \right)(\alpha + \kappa p) - \left(\begin{vmatrix} A & L \\ a & k \end{vmatrix} + \begin{vmatrix} A & L \\ l & m \end{vmatrix} p \right)(\lambda + \mu p) = 0, \tag{1.19}$$

or (verified by expanding each 3×3 determinant in the last column),

$$\begin{vmatrix} A & L & \kappa \\ K & M & \mu \\ l & m & 0 \end{vmatrix} p^2 + \left(\begin{vmatrix} A & L & \alpha \\ K & M & \lambda \\ l & m & 0 \end{vmatrix} + \begin{vmatrix} A & L & \kappa \\ K & M & \mu \\ a & k & 0 \end{vmatrix} \right) p + \begin{vmatrix} A & L & \alpha \\ K & M & \lambda \\ a & k & 0 \end{vmatrix} = 0, \tag{1.20}$$

which gives us a *quadratic equation* for p.

The system (1.9) can be solved in exactly the same manner to derive a quadratic equation for q. In this case, we move the first equation to the last position and cyclically permute the constants, replacing A, a, α with a, α, A respectively, and similarly for the other coefficients.

Then, similar to (1.20), for q we have the following quadratic equation:

$$\begin{vmatrix} a & l & K \\ k & m & M \\ \lambda & \mu & 0 \end{vmatrix} q^2 + \left(\begin{vmatrix} a & l & A \\ k & m & L \\ \lambda & \mu & 0 \end{vmatrix} + \begin{vmatrix} a & l & K \\ k & m & M \\ \alpha & \kappa & 0 \end{vmatrix} \right) q + \begin{vmatrix} a & l & A \\ k & m & L \\ \alpha & \kappa & 0 \end{vmatrix} = 0. \tag{1.21}$$

Similarly, the quadratic equation for r is:

$$\begin{vmatrix} \alpha & \lambda & k \\ \kappa & \mu & m \\ L & M & 0 \end{vmatrix} r^2 + \left(\begin{vmatrix} \alpha & \lambda & a \\ \kappa & \mu & l \\ L & M & 0 \end{vmatrix} + \begin{vmatrix} \alpha & \lambda & k \\ \kappa & \mu & m \\ A & K & 0 \end{vmatrix} \right) r + \begin{vmatrix} \alpha & \lambda & a \\ \kappa & \mu & l \\ A & K & 0 \end{vmatrix} = 0. \tag{1.22}$$

These quadratic equations reveal subsets of Nash equilibria. However, they must be interpreted with care, as simply computing their roots is not sufficient to fully identify the equilibria. We explain how to deduce the sets of equilibria from these equations in Section 1.3.2. Before addressing the computation of equilibria, we first introduce some foundational concepts that will assist us.

1.1.6. Intersection of Two Indifference Surfaces (IOS)

In the previous section, during the algebraic derivation of the quadratic equations, we implicitly first computed the intersection of two indifference surfaces. This occurred when we eliminated the first parameter and reached equation (1.18). That equation represents the intersection of the indifference surfaces of players I and II. We then computed the intersection of this curve with the third indifference surface, which led to the quadratic equations.

In this section, we take a closer look at the curve formed by the intersection of two indifference surfaces, which we refer to as *IOS curves*. Studying the properties of these curves helps us better understand the possible equilibrium components.

We define IOS_i as the intersection of the indifference surfaces of the two players other than player i:

$$\mathrm{IOS}_1 = \mathrm{IS}_2 \cap \mathrm{IS}_3, \quad \mathrm{IOS}_2 = \mathrm{IS}_3 \cap \mathrm{IS}_1, \quad \mathrm{IOS}_3 = \mathrm{IS}_1 \cap \mathrm{IS}_2.$$

We have already computed IOS_3 in equation (1.18), which represents the intersection of the indifference surfaces of the first and second players. For this intersection (IOS_3), we have the following.

$$
\begin{aligned}
\mathrm{IS}_1 : \quad & A + Kq + Lr + Mqr = 0, \\
\mathrm{IS}_2 : \quad & a + kr + lp + mrp = 0, \\
\implies \mathrm{IOS}_3 &= \mathrm{IS}_1 \cap \mathrm{IS}_2 \\
&= \Bigg\{ (p,q,r) \in [0,1]^3 \ : \\
&\begin{vmatrix} A & L \\ a & k \end{vmatrix} + \begin{vmatrix} A & L \\ l & m \end{vmatrix} p + \begin{vmatrix} K & M \\ a & k \end{vmatrix} q + \begin{vmatrix} K & M \\ l & m \end{vmatrix} pq = 0, \\
& r = \frac{A + Kq}{L + Mq} = \frac{a + lp}{k + mp} \Bigg\}.
\end{aligned}
\tag{1.23}
$$

Although (1.23) resembles the third player's indifference surface equation, it actually defines a curve in 3D space, as the value of r is determined by the values of p and q. In contrast, in IS_3, r is free to vary in the interval $[0, 1]$. Similarly, equations for the intersection curves of any two players' indifference surfaces can be derived.

These curves are displayed in the plots, such as Figure 2.1 and Figure 1.7, in purple. For example, in the top-right cube, the first player's indifference surface (IS_1) is shown in red, and the intersection curve of the other two players (IOS_1) is depicted in purple.

IOS curves exhibit several interesting properties that prove useful in subsequent proofs:

(a) Like the players' indifference surfaces, in special edge cases, the IOS curves may contain planes or surfaces rather than being purely one-dimensional curves. These situations arise directly from edge cases in the indifference surfaces themselves, whose intersection defines the IOS curves. However, since we will exclude such cases in our analysis, we refer to these sets as *IOS curves* for simplicity.

(b) These curves are the intersection of two perpendicular indifference surfaces (e.g., $\mathrm{IOS}_1 = \mathrm{IS}_2 \cap \mathrm{IS}_3$). As a result, projecting the IOS curve onto the base plane of either indifference surface preserves its form. For instance, projecting IOS_1 onto the $r \times p$ plane yields the same form as IS_2, and projecting it onto the $p \times q$ plane yields the same form as IS_3. However, this does not imply full coverage of the domain of the indifference surfaces: certain regions of one surface may lie outside the domain of the other and, consequently, outside the domain of their intersection.

(c) Interestingly, IOS_i exhibits the same form as an indifference surface—namely, a line, a degenerate hyperbola, or a non-degenerate hyperbola—when projected onto the base plane of player i. This follows from the functional relationship between p and q in (1.23). Considering part (b), we conclude that an IOS curve has a hyperbolic form when projected onto any of the coordinate planes $p \times q$, $q \times r$, or $r \times p$.

(d) IOS curves are monotonic when projected onto any two-dimensional coordinate plane. This is a direct consequence of part (c), as each branch of a curve with hyperbolic form is monotonic.

(e) When the IOS curve takes the form of a non-degenerate hyperbola in its base-plane projection, fixing any one of the coordinates p, q, or r uniquely determines the corresponding values of the remaining coordinates for points on the curve.

1.1.7. Types of Nash Equilibria

The set of Nash equilibria of a game can be categorised based on the strategies employed by the players:

- **Pure Equilibria (PRE)**: In these equilibria, all players choose pure strategies, meaning each player's mixed strategy parameter is either 0 or 1. These equilibria correspond to the vertices of the cube of mixed strategies.
- **Partially Mixed Equilibria (PME)**: In these equilibria, at least one player employs a pure strategy, but not all players do. These equilibria are situated on the edges and faces of the cube.
- **Completely Mixed Equilibria (CME)**: In these equilibria, none of the players employs a pure strategy. They are located in the interior of the cube.

To compute all Nash equilibria of $2 \times 2 \times 2$ games, we employ two distinct algorithms: one for identifying pure and partially mixed equilibria, and another for computing completely mixed equilibria. The rationale for using separate approaches lies in the nature of the best response correspondences. On the boundaries of the cube, a best-response correspondence manifests as strict surfaces, whereas in the interior, they appear as indifference surfaces. Consequently, for each player, one algorithm examines the intersection of the other players' best-response correspondences with the player's strict surfaces, while the other algorithm considers their intersection with the player's indifference surface. The union of the equilibria identified from these two methods yields the full set of Nash equilibria.

In both algorithms, certain edge cases can arise that result in unstable equilibria under small perturbations of the payoffs or lead to infinite sets of equilibria. Before presenting the algorithms, we introduce the concept of degenerate games, which captures these edge cases. We then explain how each algorithm accounts for them.

1.2. Generic and Non-Degenerate Games

The term *generic game* is often used somewhat loosely in the literature on non-cooperative games. Broadly speaking, a generic game refers to one whose payoffs are drawn from a general or "typical" set of real-life scenarios, meaning that no two distinct strategy profiles yield exactly the same payoff for any player. In one-shot strategic-form games, genericity implies that players strictly prefer one strategy profile over another, and ties in payoffs of pure strategies occur only in special, finely tuned cases.

However, when we convert extensive-form games to their strategic form, such ties are quite common even when the payoffs are generic. As a result, non-generic games arise more

frequently in these settings, as the tree structure often leads to multiple strategy profiles having the same payoff.

From a mathematical perspective, generic games correspond to a set of payoff configurations with full measure (i.e., measure one), while non-generic games lie in a set of measure zero. These measure-zero cases are considered edge cases and are typically excluded from general theoretical analysis. This is because they can exhibit problematic or unstable behaviour. For example, best-response surfaces may align exactly, resulting in infinitely many equilibria that are not robust under small perturbations in the payoff parameters. In contrast, generic games avoid such cases and are more stable, making them more representative of real-world scenarios where slight changes in preferences do not cause drastic shifts in strategic outcomes.

For two-player games, von Stengel [67, p. 164] introduced the notion of *degeneracy* as a way to more directly identify games that represent edge cases.

Definition 1.2. A two-player non-cooperative game is called *degenerate* if there exists a player with a mixed strategy whose support has size $k \geq 1$, such that the other player has more than k pure best responses to this strategy. Equivalently, if no such strategy exists for either player, the game is called *non-degenerate*.

In [66], the author proved the equivalence of Definition 1.2 with the earlier notion of degeneracy introduced in [42], where the Lemke–Howson algorithm was developed to enumerate equilibria of non-degenerate games.

An important consequence of Definition 1.2 is that non-degenerate games cannot have components of infinitely many equilibria; that is, the number of equilibria is finite, and each equilibrium is *isolated*.

Both non-degenerate and generic games aim to exclude pathological edge cases, but the two concepts are not equivalent. Every generic game is non-degenerate, but the converse does not hold: some games with repeated or symmetric payoffs can be non-degenerate yet still non-generic.

Since the set of generic games is contained within the set of non-degenerate games, results proven for non-degenerate games also apply to generic games.

Since the subgames between each pair of players in our setting are 2×2 games, the following lemma specifies the condition for non-degeneracy in such games.

Lemma 1.3. *A normalised* 2×2 *game with the payoff table (1.24) is non-degenerate if and only if the payoff parameters of both players,* (x, y, w, z)*, are nonzero.*

I \ II	Left: $1-q$	Right: q
Up: $1-p$	0 0	w 0
Down: p	0 x	z y

(1.24)

Proof. Because each player has only two strategies, degeneracy occurs if and only if a pure strategy has two pure best responses. This happens if at least one of x, y, w, z is zero. Therefore, excluding this case ensures that the game is non-degenerate.

□

When the game involves more than two players, extending this definition of degeneracy becomes challenging, as each player now has multiple opponents to consider. However, studying degeneracy in subgames (i.e., games played between any two players while other players' strategies are fixed) provides valuable intuition for understanding degeneracy in three-player games.

Building on the concept of non-degeneracy in two-player games, we extend the definition of non-degeneracy to our three-player game setting as follows.

Definition 1.4. A $2\times2\times2$ game G, represented as in Table 1.1, is said to be *non-degenerate* if the following conditions hold:

i. Every 2×2 subgame, obtained by fixing one player's strategy to a pure strategy, is non-degenerate. In terms of payoffs, this means that all payoff parameters of all players are nonzero:

$$A,\ B,\ C,\ D, \quad a,\ b,\ c,\ d, \quad \alpha,\ \beta,\ \gamma,\ \delta \neq 0.$$

ii. If all the quadratic equations (1.20), (1.21), and (1.22) have at least one real solution in the interval $[0, 1]$, then
   - no such solution equals 0 or 1,
   - none of the equations admits infinitely many solutions (i.e., is of the form $0 = 0$),
   - any equation that is properly quadratic (i.e., not linear) has exactly two distinct real solutions (i.e., no double root).

In the first condition of Definition 1.4, the games played on the faces of the cube $[0, 1]^3$ are examined. On each face of the cube, one player's strategy is fixed as a pure strategy, and the resulting 2-player subgame between the remaining players is analysed for degeneracy. If any of these 2-player subgames is degenerate, it may result in components of partially mixed Nash equilibria located on the edges of the cube in the 3-player game.

Although when the 2-player game is degenerate, it is possible to modify the fixed player's payoffs to not allow the equilibrium component to lie on the remaining best response correspondence, the overall game is still considered degenerate. This is because, under such conditions, one player is indifferent between their pure strategies when the other two players both play pure strategies. The enumeration of such partially mixed equilibrium components is discussed in more detail in Section 1.3.1.

The second condition addresses types of degeneracy that either result in a continuum of completely mixed equilibria or in edge cases arising from the intersections of indifference surfaces. In Section 1.1.5, we derived a system of quadratic equations from the intersection of all indifference surfaces. Any completely mixed equilibrium must satisfy these equations, as it lies on all indifference surfaces. Typically, these quadratic equations admit at most two isolated (singleton) roots, unless they reduce to the trivial equation $0 = 0$, in which case they correspond to infinitely many solutions. If the game has no completely mixed equilibrium, this is reflected in the system of equations either by the absence of real roots in the interval $(0, 1)$ or by a contradiction of the form "non-zero constant = 0". Moreover, in previous work proving the oddness of the number of equilibria [24], double roots of the intersection equations were shown to be edge cases, and we therefore exclude them here for compatibility.

Condition (ii) ensures that when completely mixed equilibria exist, the system does not contain any trivial equations of the form $0 = 0$, thereby ruling out the possibility of infinitely many solutions.

If a game admits a component of infinitely many equilibria, this is typically due to the presence of at least one coordinate that is unrestricted, i.e., it can vary freely within an interval. In such cases, the corresponding quadratic equation becomes trivial, allowing for infinitely many solutions along that direction. When none of the coordinates are unrestricted, all quadratic equations must be satisfied throughout the component. In this case, degeneracy is indicated by all quadratic equations admitting infinitely many solutions. Condition (ii) excludes these degenerate configurations in the interior of the cube. A more detailed discussion on the enumeration of these components is provided in Section 1.3.2.

Since our definition of degeneracy extends the concept of degeneracy defined for 2-player games, we prove that this definition also excludes the games with components of infinite equilibria.

Lemma 1.5. *If* $2 \times 2 \times 2$ *game* G*, represented as in table (1.1), is non-degenerate according to Definition 1.4, then* G *has a finite number of equilibria; that is,* G *does not have any components of infinite equilibria.*

Proof. Towards a contradiction, suppose that G is non-degenerate and has a component of infinite equilibria. Denote this set of equilibrium points by V. There are two possible cases for V: either the equilibria in V are partially mixed, or they are completely mixed (possibly except for endpoints that lie on the boundary). If V is partially mixed, then the component

lies on the boundary of the mixed strategy cube. If V is completely mixed, then it lies in the interior of the cube.

- If V is partially mixed, then at least one coordinate is fixed for all equilibria in V to a pure strategy (i.e., either 0 or 1). This corresponds to a face of the cube and a corresponding 2×2 subgame G' which we analyse. Without loss of generality, assume G' is the game on the face $r = 1$ (the back face of the cube). The table on the right in (1.1) shows the 2×2 subgame between players I and II corresponding to G' (the payoffs of the third player are ignored in this subgame analysis).

  In the 3-player game G, by the definition of Nash equilibrium, any point in V lies on all players' best response correspondences. Therefore, in the 2-player subgame G', any point in V must also lie on the best response correspondences of both players I and II. Hence, V constitutes an infinite equilibrium component of the subgame G', which implies that G' is degenerate. However, by Lemma 1.3, we know that if the payoff parameters of the subgame, C, D, b, d, are all non-zero, then the game is non-degenerate. Therefore, at least one of C, D, b, d must be zero, contradicting the assumption that G is non-degenerate (since condition (i) in Definition 1.4 requires all these parameters to be non-zero).

  Considering the other subgames of G, the union of conditions from all of them makes it clear that the payoff parameters of all players must be non-zero. This condition is necessary to avoid degeneracy in subgames that would otherwise give rise to components of infinite partially mixed equilibria.

- If V is completely mixed, then the mixed strategy parameters at every point in V must satisfy equations (1.20), (1.21), and (1.22). Therefore, all three quadratic equations must admit solutions.

  Since V is not a single point, at least one mixed strategy parameter must vary among the points in V. The associated quadratic equation for this parameter must be satisfied for all values it takes within V. This is only possible if the corresponding quadratic equation has infinite solutions; that is, it reduces to the form $0 = 0$. However, this contradicts the non-degeneracy assumption of the game because the trivial quadratic equation violates condition (ii) in Definition 1.4.

Since both cases lead to a contradiction, the assumption must be false: therefore, a non-degenerate game has only a finite number of Nash equilibria. □

In summary, the relationship between generic games, non-degenerate games, and games with finitely many isolated equilibria can be expressed as follows:

$$\text{Generic games} \subset \text{Non-degenerate games} \subset \text{Games with finitely many equilibria} \quad (1.25)$$

This implies that the converse of Lemma 1.5 does not hold: there exist $2 \times 2 \times 2$ games with a finite number of equilibria that are nonetheless degenerate. Figure 1.5 illustrates such a case: the game has a single completely mixed equilibrium but does not satisfy condition (i) in the definition of non-degenerate games, as some payoff parameters of player III are zero.

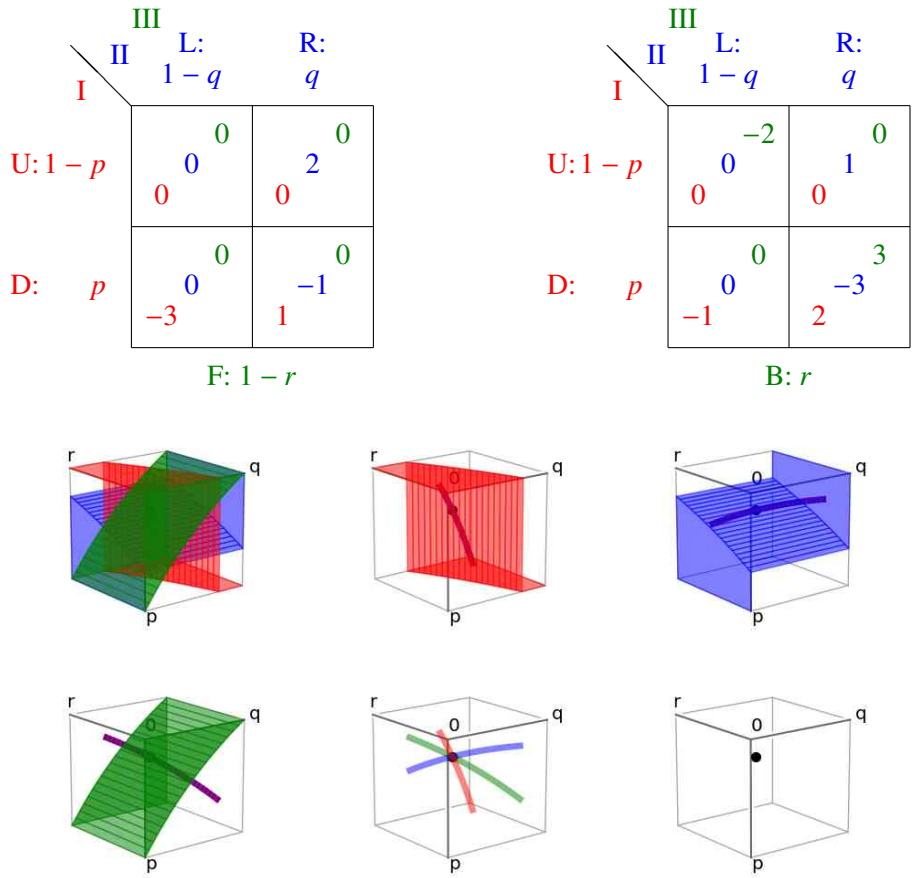


Figure 1.5.: Example of a degenerate game with one single equilibrium (Test 2 in the Python code).

Moreover, there exist non-degenerate games that are not generic. For example, games where the indifference surface takes the form of a degenerate hyperbola in the base plane are non-generic, as this form changes with the smallest perturbations in the payoff parameters. However, they are not necessarily degenerate. Figure 3.1 in the Appendix shows an example of such a game.

1.3. Enumeration of Nash Equilibria

So far, we have discussed the types of Nash equilibria in $2 \times 2 \times 2$ games and introduced the concept of degeneracy in such games. In this section, we present our method for enumerating all Nash equilibria, whether the game is degenerate or non-degenerate.

Our approach divides the computation into two parts:

- Computation of *partially mixed* and *pure* equilibria.
- Computation of *completely mixed* equilibria.

We employ different algorithms for each part. The union of the equilibria obtained through these algorithms constitutes the full set of Nash equilibria for the game.

1.3.1. Computing Partially Mixed and Pure Equilibria

In an equilibrium, each player's strategy is a best response to the strategies of the other players. In a pure or partially mixed equilibrium, at least one player plays a pure strategy. We can exploit this property to identify such equilibria by examining six subgames, each obtained by fixing one of the three players' strategies to a pure strategy.

These equilibria arise when the indifference surfaces intersect the boundaries of the $[0, 1]^3$ cube. On these boundaries, the best-response correspondence of the pure-strategy player takes the form of strict surfaces, while the best responses of the other players appear as projections of their indifference surfaces onto the corresponding face of the cube. Consequently, this algorithm examines each face of the cube to identify such equilibria.

In each subgame, we fix the strategy s_i of one player ($i = 1, 2, 3$) to be 0 or 1. Fixing one player's strategy results in a 2×2 game, for which we compute all equilibria. This can be done by simple case distinction for 2×2 games or using the lrsNash algorithm [3], which enumerates all equilibria of any two-player game.

In the next step, for each equilibrium component of the subgame, denoted as s_{-i}, the strategies of the two players in the subgame, along with the pure strategy of the fixed player, form the strategy profile for the three-player game. We then change the fixed player's strategy to the opposite value (0 if $s_i = 1$ and 1 if $s_i = 0$). If the original strategy profile yields a payoff that is at least as high for the fixed player as the modified strategy profile, then the original strategy profile is a partially mixed or pure Nash equilibrium (PMNE) of the game.

Algorithm 1 provides a simplified pseudo-code.

Algorithm 1 Finding partially mixed and pure equilibria

Input: Payoff table (1.1) of a $2 \times 2 \times 2$ game
Output: Set of partially mixed and pure equilibria of the game

```
PMNE ← ∅
for each player i do
    for s_i ∈ {0, 1} do
        SG ← 2 × 2 game when player i plays s_i
        cand ← all Nash equilibria of SG          ▷ using lrsNash or case distinction
        for each s_{-i} ∈ cand do                 ▷ strategy pair of the other two players
            if U_i(s_i, s_{-i}) ≥ U_i(1 − s_i, s_{-i}) then ▷ U_i is the expected payoff for player i
                add (s_i, s_{-i}) to PMNE
            end if
        end for
    end for
end for
return PMNE
```

Lemma 1.6. *Assume P_a is the set of equilibria found using Algorithm 1 and P_m is the set of all partially mixed and pure equilibria of the game. Then, $P_a = P_m$.*

Proof. We start with proving $P_a \subseteq P_m$. Suppose $x \in P_a$ is found using the algorithm. Then, $\exists i$, such that $x_i \in \{0, 1\}$ and $x = (x_i, x_{-i})$, where i is the player from the first `for` loop, and x_i is the fixed strategy. Clearly, since x_i is a pure strategy, if x is an equilibrium, then it would be either partially mixed or pure. Thus, it is sufficient to show that x is an equilibrium.

Towards a contradiction, assume that x is not an equilibrium. Then, representing the expected payoff function of player i as U_i we have:

$$\exists j \quad \exists \bar{x_j} \in [0, 1] : \quad U_j(\bar{x_j}, x_{-j}) > U_j(x_j, x_{-j}).$$

Player j cannot be any index other than i because x_{-i} is an equilibrium of the 2×2 game (line 4 of algorithm). From the expected payoff function U_i of player i, and line 7 of the algorithm, we have:

$$\begin{aligned} U_i(\bar{x_i}, x_{-i}) = \quad & \bar{x_i}.U_i(0, x_{-i}) + (1 - \bar{x_i})U_i(1, x_{-i}) \quad > \quad U_i(x_i, x_{-i}) \\ \implies \quad & max(U_i(0, x_{-i}), U_i(1, x_{-i})) \quad > \quad U_i(x_i, x_{-i}) = max(U_i(0, x_{-i}), U_i(1, x_{-i})) \end{aligned}$$

which is a contradiction. So, the assumption does not hold and $x \in P_m$.

For the second part of the proof, we assume $x \in P_m$. According to the definition of partially mixed equilibria, $\exists i : x_i \in \{0, 1\}$, a player who plays a pure strategy. Since x is an equilibrium, it will be an equilibrium for the 2×2 games when we fix player i's strategy to be x_i. This means $x \in$ `cand` in line 5 of the code, when player $= i$ and $s_i = x_i$. Then because x is an equilibrium,

$$\forall i \quad \forall \bar{x_i} \in [0, 1] : \quad U_i(x) \geq U_i(\bar{x_i}, x_{-i}).$$

This means the condition in line 7 of the algorithm holds, and x will be added to PMNE. So $x \in P_a$. Combining the two parts of the proof, we have $P_a = P_m$. □

Algorithm 1 also applies when the 2×2 subgame is degenerate and `cand` is an infinite component of equilibria. Such a component may be a line segment or even the entire face of the cube.

The case where `cand` spans an entire face of the cube implies that all points on player i's strict surface on that face are equilibria of the game. Since the intersection of a face of the cube with the best-response correspondence of the player has parametrised boundaries, either straight lines or hyperbolic branches (see Section 1.1.4), this case is straightforward to handle.

In the remaining case, where `cand` contains a line segment, the algorithm identifies the intersection of player i's indifference surface with `cand`, as this is where player i's best response may change. This intersection is efficiently computed using the fact that

equilibrium line segments in 2×2 games are either horizontal or vertical. As a result, one player's strategy remains fixed along the segment, while only a single variable changes, making the intersection point simple to compute.

1.3.2. Computing Completely Mixed Equilibria

In this section, we assume that all partially mixed equilibria have been computed using the previous algorithm. Here, we focus on computing the completely mixed equilibria, if they exist.

In Section 1.1.4, we illustrated the geometric representation of indifference surfaces. The intersection of these surfaces for all players corresponds to the completely mixed equilibria of the game. Therefore, in Section 1.1.5, solving the indifference equation system led us to three quadratic equations, one for each player's mixed strategy.

The quadratic equation of each player, unless it simplifies to $0 = 0$, has at most two solutions for the player's mixed strategy parameter. These solutions must lie within $(0, 1)$ to represent valid mixed equilibrium strategies.

An important result derived from the quadratic equations is that a non-degenerate (and therefore generic) $2 \times 2 \times 2$ game has at most two completely mixed equilibria, as proved in more complex ways by [11] and [44].

Theorem 1.7. *Any non-degenerate* $2 \times 2 \times 2$ *game has at most two completely mixed equilibria.*

Proof. According to Definition 1.4 of non-degenerate games, either the quadratic equation of one player has no solution in $(0, 1)$, or the quadratic equations of all players have at most one or two solutions in $(0, 1)$ (but never infinitely many).

In the first case, the game has no completely mixed equilibria, since the coordinates of any such equilibrium must satisfy all players' quadratic conditions. This aligns with the statement of the theorem.

In the second case, a solution to one player's quadratic equation determines their mixed strategy, which then reduces the indifference equations of the other two players to linear equations with at most one solution for their respective mixed strategies.

For instance, in the indifference equation system (1.16), fixing a value of p reduces the second and third equations to linear equations in r and q, respectively, with at most one solution for each. In a non-degenerate game, these reduced equations cannot yield a trivial equation $0 = 0$, as that would correspond to a degenerate game where the original quadratic equation also has infinitely many solutions.

Therefore, for each solution of the quadratic equation, at most one equilibrium can be defined, which is feasible only if the other mixed strategies can be derived from this solution and all mixed strategies lie within the $[0, 1]^3$ cube. Hence, the number of completely mixed equilibria is at most two. □

To compute the completely mixed equilibria of a $2 \times 2 \times 2$ game, we begin with the quadratic equations. If any of these equations has no solution in the interval $(0, 1)$, then the game admits no completely mixed equilibrium. By examining the solutions of the quadratic equations, we can already determine whether the game is non-degenerate.

When the quadratic equations do admit solutions in $(0, 1)$, we substitute these values into the players' indifference equations to compute the remaining mixed strategies. If all resulting parameters lie within the interval $(0, 1)$, the solution constitutes an equilibrium.

It is worth noting that if p, q, and r are the solutions to the quadratic equations, they may be irrational numbers in a completely mixed equilibrium. These values can be represented as approximate floating-point numbers or symbolically as algebraic numbers involving square roots, assuming the payoffs are given as rational inputs.

In degenerate games, the quadratic equations may hold trivially in the form $0 = 0$, which can indicate an infinite solution set. For example, equation (1.20) becomes trivial in case (a) when $A = K = L = M = 0$.

Moreover, even if equation (1.20) has two real solutions for p, the third equation in system (1.16) may simplify to $0 = 0$ for one or both values of p, leaving q undetermined. In such cases, equation (1.21) may also reduce to $0 = 0$, resulting again in an underdetermined solution.

Another source of infinite solutions arises when any of the mixed strategy variables p, q, or r take boundary values 0 or 1. In such cases, the corresponding player plays a pure strategy and is not required to satisfy an indifference condition, allowing the indifference equations to hold trivially. These cases are addressed in the analysis of partially mixed equilibria, as discussed in the previous section.

To compute all completely mixed equilibria, we begin by solving the quadratic equations. If any of these equations has no solution in the interval $[0, 1]$, then the game does not admit any completely mixed equilibria. Determining the solutions to the quadratic equations also reveals whether the game is non-degenerate.

For non-degenerate games, for a quadratic equation that has solutions in $[0, 1]$, we substitute these into the indifference equations of the players to compute the other mixed strategies (if possible). If all resulting parameters lie within $[0, 1]$, then the resulting strategy profile defines a completely mixed equilibrium.

A noteworthy point is that if p, q, and r are the solutions to the quadratic equations, they may be irrational numbers. In such cases, these can be represented approximately using floating-point numbers or symbolically using square roots as algebraic numbers, assuming rational payoffs as input in our code.

In degenerate games, quadratic equations may reduce trivially to $0 = 0$, which can indicate an infinite set of solutions. For instance, equation (1.20) becomes trivial in case (a) when $A = K = L = M = 0$.

Furthermore, even when equation (1.20) has two real solutions for p, the third equation in system (1.16) might simplify to $0 = 0$ for one or both values of p, leaving q undetermined. In

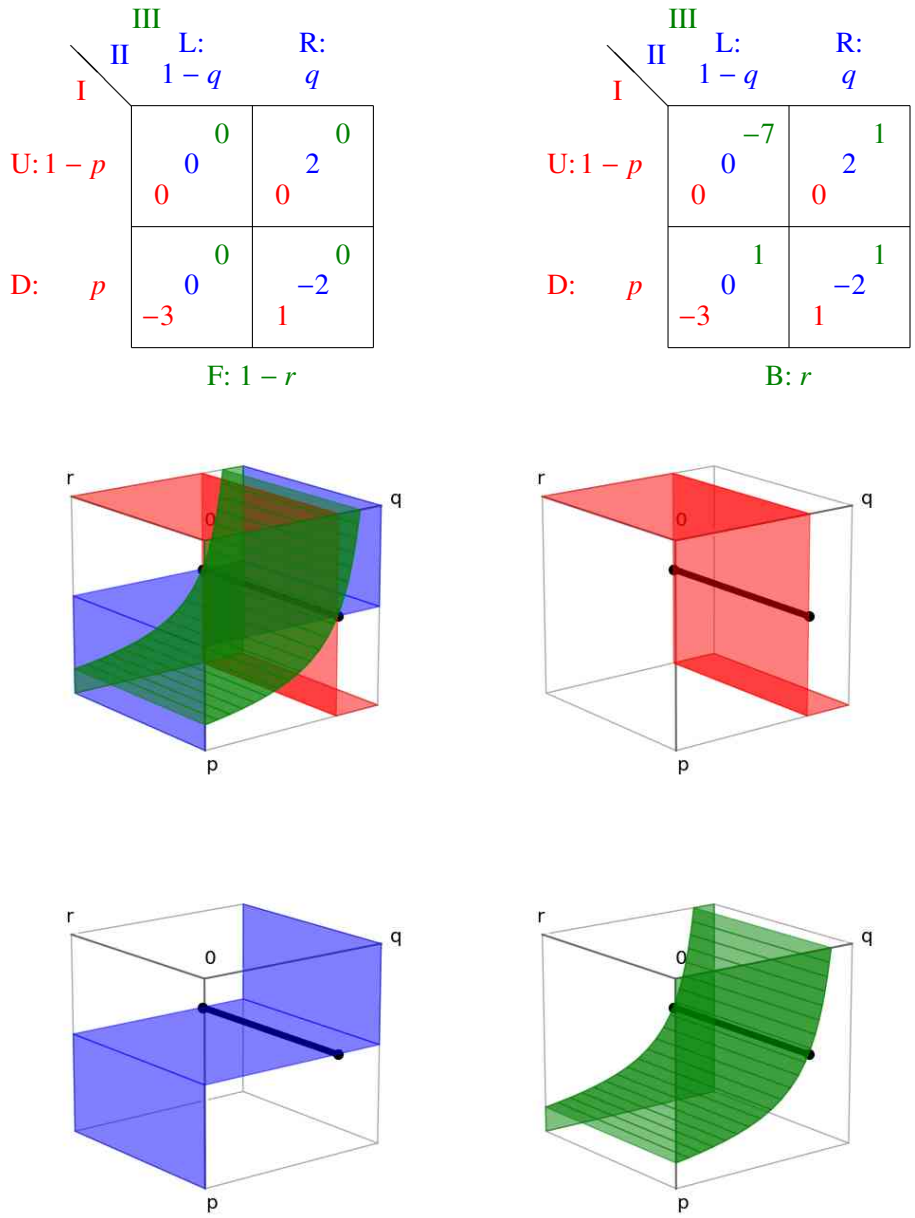


Figure 1.6.: Example of a game with a line of mixed equilibria, when the quadratic equations have infinitely many solutions (Test 0 in the Python code).

such cases, equation (1.21) often also reduces to $0 = 0$, again leading to an underdetermined system.

Another source of infinite solutions arises when any of the strategy parameters p, q, or r take boundary values 0 or 1. In such cases, the corresponding player uses a pure strategy and is not required to satisfy an indifference condition, allowing the indifference equations to hold trivially. These cases are covered in the analysis of partially mixed equilibria, as discussed in the previous section.

To compute all completely mixed equilibria in degenerate $2 \times 2 \times 2$ games, one must consider not only the quadratic equations but also the indifference surface equations that express the relations between each pair of mixed strategy parameters.

An example of this case is the game shown in Figure 1.6. In this game, all the quadratic equations yield $0 = 0$, which, according to condition (ii) of Definition 1.4, implies the game

is degenerate. To determine the nature of this degeneracy and compute all completely mixed equilibria, we examine the functional relations between each pair of strategy parameters:

$$IS_1(q,r): \quad 0qr + 4q + 0r - 3 = 0$$
$$IS_2(r,p): \quad 0rp + 0r - 4p + 2 = 0$$
$$IS_3(p,q): \quad -8pq + 8p + 8q - 7 = 0$$

$$p = f(q) = \tfrac{-8q+7}{-8q+8}, \qquad q = f(p) = \tfrac{-8p+7}{-8p+8},$$
$$p = f(r) = \tfrac{1}{2}, \qquad q = f(r) = \tfrac{3}{4}.$$

As the coefficients of the r terms are zero in IS_1 and IS_2, r cannot be written as a function of the other variables (as this would require division by zero). This also implies that during the derivation of the quadratic equation in terms of r, multiplication by zero occurred. Therefore, the parameter r is unrestricted, while p and q are fixed. This defines a line of equilibria in the r-direction, where $p = \frac{1}{2}$ and $q = \frac{3}{4}$, as observed in each player's best-response correspondence in Figure 1.6.

Taking into account all possible edge cases that arise in the quadratic equations and the indifference surfaces, we compute all completely mixed equilibria using our Python code. Below, we explain key ideas in our implementation.

When all payoff parameters of a player are zero (see Figure 1.2 (a)), the equilibrium computation reduces to finding the intersection of the other players' best-response correspondences. All points in this intersection are equilibria.

Additionally, when a player's indifference surface has the form of a degenerate hyperbola in the base plane, we decompose it into its two asymptote planes (see (1.15)) and compute the intersection of each plane with the other players' indifference surfaces. The union of these intersections yields the equilibria. This method is preferred because degenerate hyperbolas introduce edge cases in the formation of quadratic equations, and the resulting equilibrium components may take different forms. Considering the asymptote planes separately simplifies the interpretation.

Our analysis determines the number of unrestricted parameters and their relations to the others, which define the structure of the infinite components of completely mixed equilibria. The union of this set with the pure and partially mixed equilibria yields the complete set of Nash equilibria. The infinite components of equilibria in a $2 \times 2 \times 2$ game can be:

- 3-dimensional (e.g., the entire cube),
- 2-dimensional (e.g., parts of best-response correspondences, Test 37),
- 1-dimensional (e.g., lines, Test 0, or curves, Test 32), or
- 0-dimensional (singleton points, Test 34),

where, for non-degenerate games, only 0-dimensional components (singleton points) are possible.

You can run the test cases mentioned above in our Python code.

1.4. Graphical Representation of the Game

All the figures used in Chapters 1 and 2 of this thesis are generated by our Python code. By specifying the payoff parameters of a $2 \times 2 \times 2$ game, our program computes the game's equilibria, provides relevant details in text format, and generates an interactive 3D representation of the game surfaces and equilibria, which can be rotated and examined in detail.

We have also included a set of interesting test cases in the code, each corresponding to a predefined game and identified by a test number. Throughout these chapters, we refer to these test numbers for both the examples discussed and other notable cases not covered in detail. You can run any of these test numbers in our program to view the corresponding game.

In this section, we explain the graphical representation of the game (the output produced by our code), which applies to all the figures used in this project.

Each graphical presentation consists of six subplots, arranged either horizontally (e.g., Figure 1.8) or vertically (e.g., Figure 2.1). The cube of mixed strategies, introduced in Section 1.1.1, is shown in all these plots.

In each subplot, we display some of the best response correspondences and the curves defined earlier, allowing us to examine them individually. The subplots are numbered row by row, and we explain each one below:

1. The first subplot displays the best-response correspondence of all players along with all equilibria. Due to the difficulty in distinguishing them, we also provide separate plots in the subsequent subplots.
2. In the second subplot, the best-response correspondence of player I is plotted in red. IOS_1, the intersection of IS_2 and IS_3, is represented in purple. The completely mixed equilibria, found at the intersection of IOS_1 and BR_1, are displayed in black, while partially mixed (including pure) equilibria are shown in grey. Similar representations are employed for other players.
3. The third subplot displays the best-response correspondence of player II in blue, along with IOS_2 in purple, and equilibria in black and gray.
4. In the fourth subplot, the best-response correspondence of player III is depicted in green, along with IOS_3 in purple, and equilibria in black and gray.
5. The fifth subplot illustrates the intersection curves: IOS_1 in red, IOS_2 in blue, and IOS_3 in green. Equilibria are also represented in black and grey.
6. The final plot exclusively features equilibria, with completely mixed equilibria in black and partially mixed (including pure) equilibria in grey.

To conclude this chapter, we work through a classic example from the literature.

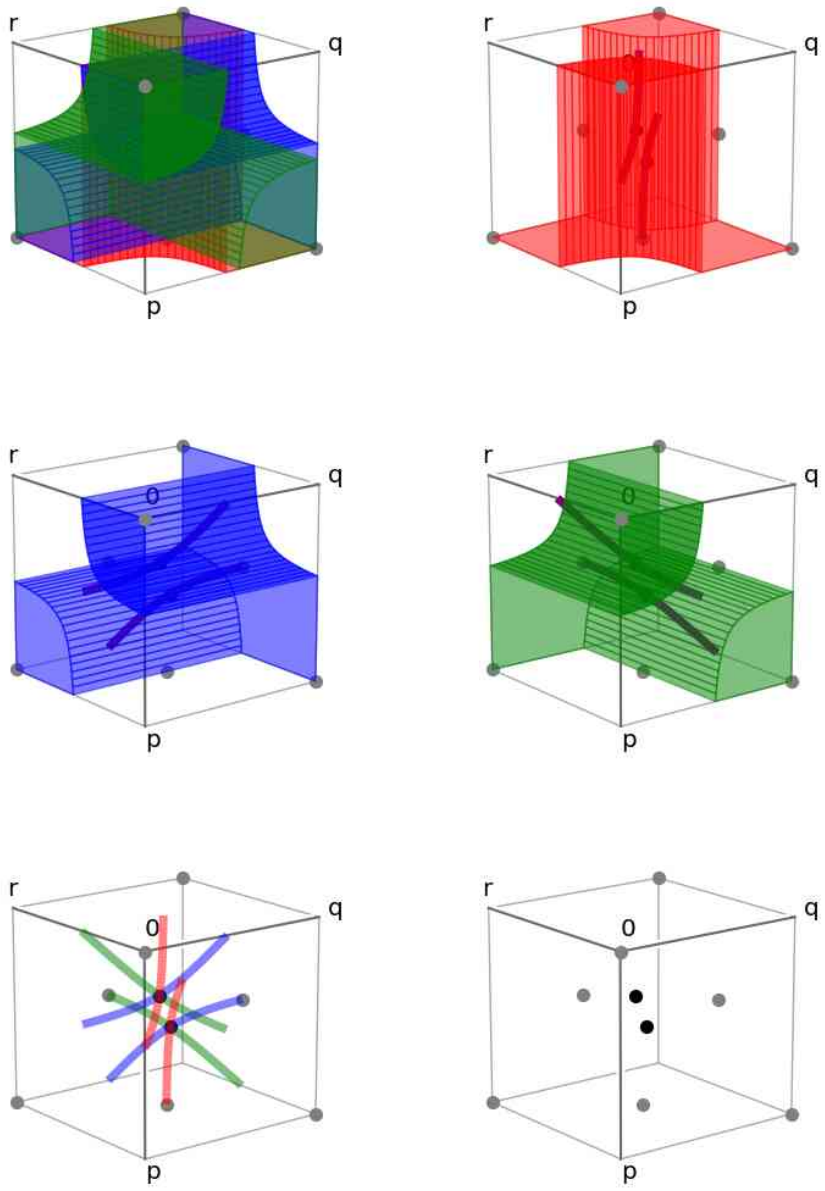


Figure 1.7.: Example of best-response surfaces of a game with 9 equilibria: 4 pure, 3 just partially mixed (not pure), and 2 completely mixed equilibria (Test 12 in the Python code).

1.4.1. Selten's Horse, a Well-Known Example

The extensive-form game shown in Figure 1.8 is a well-known example from Selten [61, Figure 1]. Due to the shape of its game tree, it is commonly referred to as "Selten's horse." The corresponding strategic form is displayed on the right.

This game is known to exhibit two segments of partially mixed equilibria and no completely mixed equilibria. The best-response correspondences and equilibrium segments are plotted below. These segments include the pure strategy equilibria ULF and DRB.

This example illustrates how degeneracy can arise naturally from extensive-form games.

In the computation of completely mixed equilibria for this game, the quadratic equation in q (1.21) simplifies to $2 = 0$, indicating that no real solution exists. This confirms that the game admits no completely mixed equilibria.

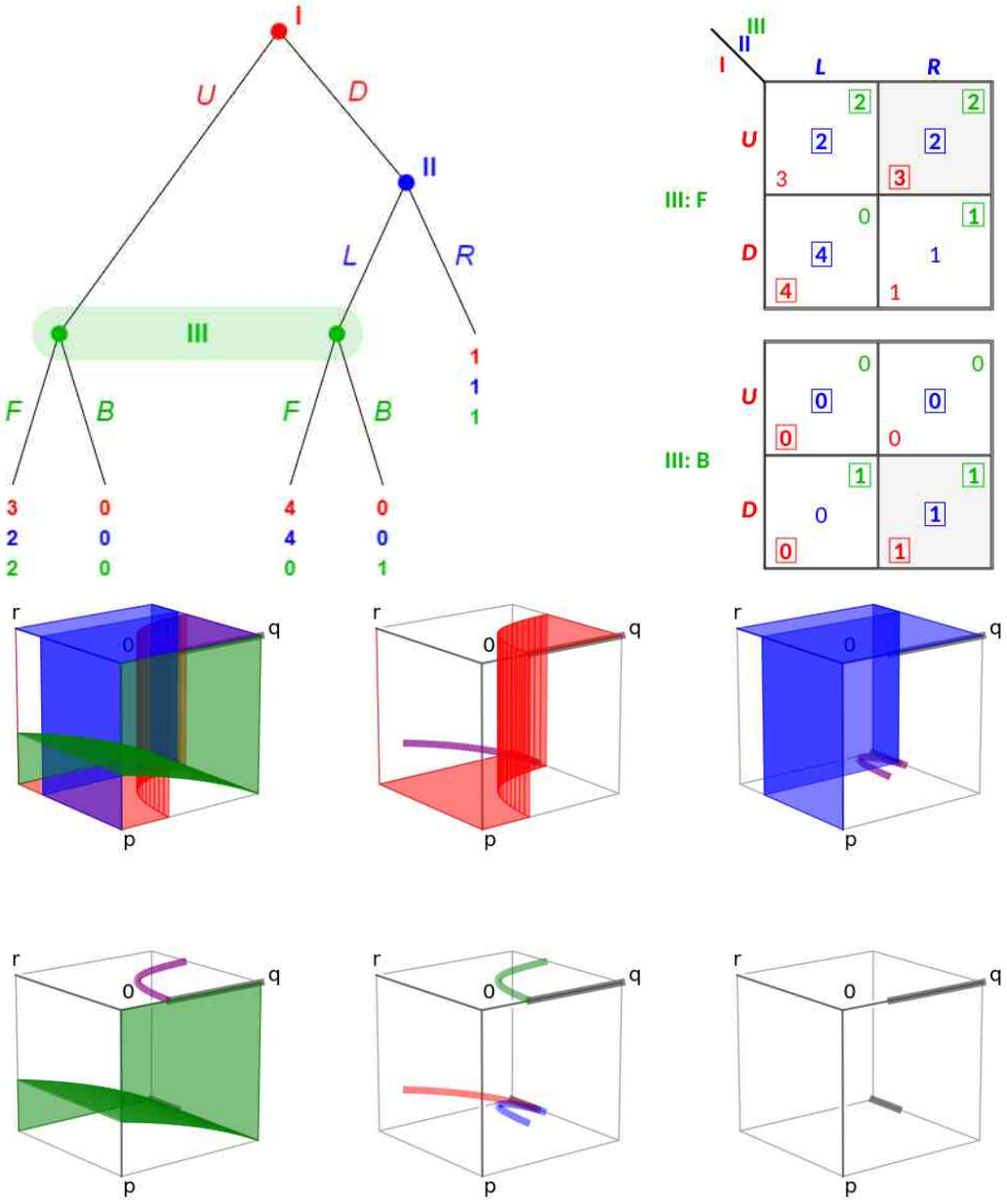


Figure 1.8.: A well-known example: "Selten's horse", see Section 1.4.1 (Test 1 in the Python code).

2. Upper Bound on the Number of Equilibria in $2 \times 2 \times 2$ games

In this chapter, we investigate bounds on the number of isolated equilibria in $2 \times 2 \times 2$ games. As explained through several examples in the previous chapter, components of infinite equilibria can arise in degenerate games when the best response correspondences of the players intersect in specific ways. However, we do not address such cases here. To establish a bound on the number of isolated equilibria, we restrict our attention to non-degenerate games, as defined in Definition 1.4. In Lemma 1.5, we proved that non-degenerate games do not contain any components of infinite equilibria.

As shown by Nash [47], every finite game has at least one equilibrium in mixed strategies. The example in Figure 2.1 shows a non-degenerate game with exactly one (completely mixed) equilibrium point. This demonstrates that the lower bound on the number of equilibria cannot be improved.

We now turn to studying upper bounds on the number of isolated equilibria in such games.

2.1. Preliminary Lemmas

In Theorem 1.7, we proved that non-degenerate games can have at most two completely mixed equilibria. In the following, we present lemmas that will help us in the subsequent proofs.

Lemma 2.1. *No non-degenerate* $2 \times 2 \times 2$ *game has more than one equilibrium on any edge of the cube.*

Proof. Towards a contradiction, assume there are two isolated equilibria E_1 and E_2 on an edge. Without loss of generality we assume $E_1 = (p', q', r'_1)$ and $E_2 = (p', q', r'_2)$ where $p', q' \in \{0, 1\}, r'_1, r'_2 \in [0, 1]$ and $r'_1 < r'_2$. We want to prove that the line segment $[E_1, E_2]$ is on all players' best response correspondences.

Since E_1 and E_2 are equilibria, they are located on all players' best-response correspondences. Dependent on the coordinates of the points, the expected payoff of players I and II are equal, greater or less than zero (from best response functions 1.3 and 1.4) but

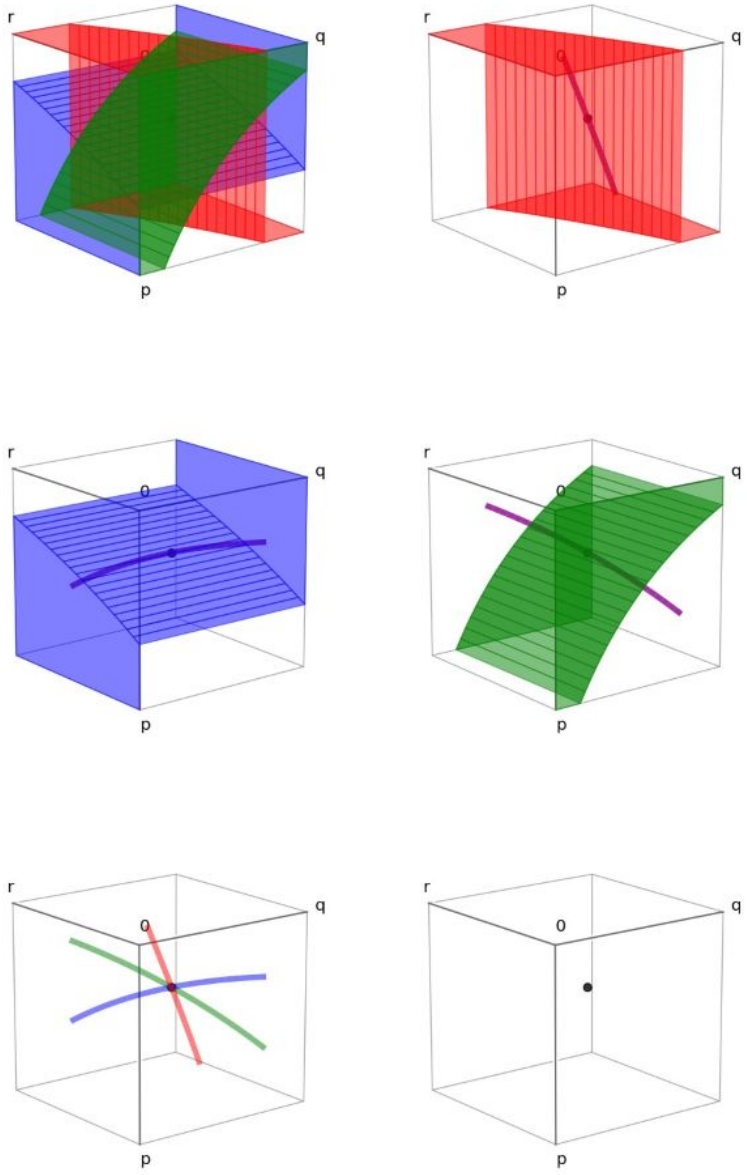


Figure 2.1.: Example of a non-degenerate game with only one equilibrium (Test 43 in the Python code).

the same for both points since the first two coordinates are equal. However, since the third coordinate is different between the two points, both points have to be on the indifference surface of player III (as it extends in the direction of the third coordinate). Without loss of generality, We assume $S(q', r_1'), S(q', r_2') \leq 0$ and $s(r_1', p'), s(r_2', p') \geq 0$, then using the

notation in (1.9) we have:

$$\begin{array}{rcrcrcrcl} Mq'r_1' & + & Kq' & + & Lr_1' & + & A & \leq & 0 \\ Mq'r_2' & + & Kq' & + & Lr_2' & + & A & \leq & 0 \\ mr_1'p' & + & kr_1' & + & lp' & + & a & \geq & 0 \\ mr_2'p' & + & kr_2' & + & lp' & + & a & \geq & 0 \\ \mu p'q' & + & \kappa p' & + & \lambda q' & + & \alpha & = & 0 \end{array}$$

We define E_t to be on the convex combination of E_1 and E_2.

$$E_t := tE_1 + (1-t)E_2 = (p', q', tr_1' + (1-t)r_2'), \quad t \in [0,1]$$

by multiplying first and third equation by t, second and fourth equation by $(1-t)$, and summing them up,

$$\begin{array}{rcrcrcrcl} Mq'(tr_1' + (1-t)r_2') & + & Kq' & + & L(tr_1' + (1-t)r_2') & + & A & \leq & 0 \\ m(tr_1' + (1-t)r_2')p' & + & k(tr_1' + (1-t)r_2') & + & lp' & + & a & \geq & 0 \\ \mu p'q' & + & \kappa p' & + & \lambda q' & + & \alpha & = & 0 \end{array}$$

This proves that E_t is located on the best response correspondence of players I and II and on the indifference surface of player III. Therefore, E_t, for all $t \in [0,1]$, is also an equilibrium. Hence, the points on the segment $[E_1, E_2]$ form a component of infinite equilibria, which contradicts the non-degeneracy of the game. □

Corollary 2.2. *Any* $2 \times 2 \times 2$ *non-degenerate game has at most 4 pure Nash equilibria.*

Proof. This can be inferred from Lemma 2.1, because if we have more than 4 pure equilibria, i.e., at least 5 pure equilibria, then at least 2 equilibria must be located on the same edge of the cube, which, according to the previous lemma, results in degeneracy. □

We explained what hyperbolas are in Definition 1.1, as the set of points (x, y) satisfying the following equation:

$$(x-a)(y-b) = c \tag{2.1}$$

where $a, b, c \in \mathbb{R}$.

Building on this definition, we now introduce the notion of a hyperbola function.

Definition 2.3. A *hyperbola function* h is a function $h : \mathbb{R} \setminus \{a\} \to \mathbb{R} \setminus \{b\}$, defined by

$$x \mapsto h(x) = y = b + \frac{c}{x-a} \tag{2.2}$$

for a non-degenerate hyperbola, that is, when $c \neq 0$.

The following lemmas are straightforward to prove.

Lemma 2.4. *The hyperbola function h in (2.2) is a bijection from $\mathbb{R} \setminus \{a\}$ to $\mathbb{R} \setminus \{b\}$. Its inverse is the hyperbola function h', given by*

$$y \mapsto h'(y) = x = a + \frac{c}{y - b}\,. \tag{2.3}$$

Lemma 2.5. *Let $d, e, f, g \in \mathbb{R}$ with $f \neq 0$ and $ef \neq dg$. Then the function*

$$x \mapsto y = \frac{dx + e}{fx + g} \tag{2.4}$$

is a hyperbola function of the form

$$x \mapsto y = \frac{d}{f} + \frac{(ef - dg)/f^2}{x + g/f}\,. \tag{2.5}$$

The condition $ef \neq dg$ in Lemma 2.5 ensures that the function in (2.4) is not constant.

We observe that the composition of two hyperbola functions is either linear or a hyperbola function, except for being undefined on some points where one function takes values of the respective asymptotes. This gives the formulation for IOS curves, which are the intersection of Indifference Surfaces (IS), explained in Section 1.1.6.

Lemma 2.6. *Let $a, b, c, A, B, C \in \mathbb{R}$, $c \neq 0$, $C \neq 0$, and consider the hyperbola functions $x \mapsto y$ for $x \neq a$ in (2.2) and $y \mapsto z$ for $y \neq A$ in*

$$y \mapsto z = B + \frac{C}{y - A}\,. \tag{2.6}$$

Then the composition $x \mapsto z$ of these functions is either a line, if $b = A$, given by

$$x \mapsto z = \frac{C}{c}x - \frac{Ca}{c} + B \qquad (x \neq a), \tag{2.7}$$

or, if $b \neq A$, is the hyperbola function

$$x \mapsto z = B + \frac{C}{b - A} + \frac{-Cc/(b - A)^2}{x - a + c/(b - A)} \qquad (x \neq a,\ \ x \neq a + \frac{c}{A - b})\,. \tag{2.8}$$

Proof. If $b = A$, then (2.7) holds as factors in b and A in the denominator of the equation below cancel each other.

$$z = B + \frac{C}{b - A + \frac{c}{x - a}}$$

If $b \neq A$ then (2.8) is obtained by substituting y from (2.2) into (2.6) as above and applying Lemma 2.5 with $x - a$ instead of x and $d = C$, $e = 0$, $f = b - A$, $g = c$. The inequality $x \neq a + \frac{c}{A-b}$ prevents the case $y = A$ and a zero denominator in (2.8). □

The next lemma may appear simple, but it plays a key role in the proof of our main theorem.

Lemma 2.7. *Any two distinct non-degenerate hyperbolas intersect in at most two points.*

Proof. Suppose $y = b + \frac{c}{x-a}$ and $y = B + \frac{C}{x-A}$ are two non-degenerate hyperbolas, where $a \neq A$, $b \neq B$, $c \neq 0$, and $C \neq 0$ (to exclude degenerate cases).

The intersection points are the solutions to

$$b + \frac{c}{x-a} = B + \frac{C}{x-A},$$

which can be rearranged to

$$(B-b)(x-A)(x-a) + C(x-a) - c(x-A) = 0.$$

This is a quadratic (or possibly linear) equation in x, which has at most two real solutions. Therefore, the hyperbolas intersect in at most two points. □

Finally, the next lemma shows that the branches of a hyperbola never meet.

Lemma 2.8. *In a non-degenerate hyperbola with* $c \neq 0$ *as in (refhyp), the branches maintain a positive distance of* $2\sqrt{2c}$ *from each other.*

Proof. Without loss of generality, assume the hyperbola is centred at the origin, so that $a = b = 0$ in (refhyp). The equation becomes:

$$xy = c.$$

The two branches of this hyperbola are symmetric with respect to the origin. The shortest distance between them occurs along the line $y = x$ if $c > 0$, or $y = -x$ if $c < 0$.

Consider the case $c > 0$; the case $c < 0$ follows by symmetry. Substituting $y = x$ into the equation $xy = c$ gives:

$$x^2 = c \quad \Rightarrow \quad x = \pm\sqrt{c}, \quad y = \pm\sqrt{c}.$$

Hence, the points of intersection are

$$\begin{pmatrix} \sqrt{c} \\ \sqrt{c} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} -\sqrt{c} \\ -\sqrt{c} \end{pmatrix}.$$

Thus, the minimum distance between the branches is $2\sqrt{2c}$. □

2.2. Upper Bound Theorem

The main result of this chapter is the following theorem.

Theorem 2.9. *Any generic* $2 \times 2 \times 2$ *game has at most nine equilibria.*

The proof of Theorem 2.9 requires several preliminary discussions and results, which we present first. The complete proof is provided towards the end of this section in Section 2.2.2.

From Theorem 1.7 and Corollary 2.2, we have established that each non-degenerate game contains at most two completely mixed and four pure equilibria. To derive the upper bound on the total number of equilibria in a non-degenerate game, we focus on the scenario in which a game attains the maximum number of completely mixed and pure equilibria. We then analyse the possible existence of partially mixed equilibria.

Theorem 2.10. *Let G be a non-degenerate $2 \times 2 \times 2$ game with four pure Nash equilibria. On the cube of mixed strategies, define player i's faces as those on which player i's strict surfaces lie. Then, for any IOS_i curve of player $i \in \{1, 2, 3\}$ (as defined in Section 1.1.6), if both endpoints of IOS_i that lie on player i's faces correspond to partially mixed equilibria, then the game G cannot have two completely mixed equilibria.*

Proof. Starting with the assumption of four pure equilibria, Lemma 2.1 establishes that no other equilibrium points can exist on the edges of the cube, as each edge already contains one pure equilibrium.

Thus, the pure equilibria must be non-adjacent and located on opposite sides of each face of the cube. There are only two possible configurations for the four pure equilibria. Without loss of generality, we assume the four pure equilibria are:

$$(1, 0, 0), \quad (0, 1, 0), \quad (0, 0, 1), \quad \text{and} \quad (1, 1, 1).$$

For the other configuration, the structure of the proof remains the same.

Moreover, we exclude indifference surfaces that take a linear form, as these can be viewed as simplified hyperbolas containing only a single branch. In non-degenerate games, such linear cases yield fewer equilibria and are therefore excluded from our analysis.

In games where the indifference surface of at least one player takes the form of a degenerate hyperbola on the base plane, the IOS curves and corresponding strict zones reduce to linear forms. This simplification makes the enumeration of partially mixed equilibria more tractable. Therefore, we analyse these cases separately in Appendix 3.2.

Consequently, in this theorem, we focus on games in which the best-response correspondences of all players take the form of non-degenerate hyperbolas in the base plane. We present the proof for the IOS_3 curve, noting that the analysis for the other IOS curves proceeds analogously.

Consider the base plane of player III, which we denote by the set $Z = [0, 1] \times [0, 1]$ representing the mixed strategies (p, q) of the opponents (players I and II). As described and illustrated in Section 1.1.2, the indifference surface IS_3 of player 3 divides the set Z into two *strict zones*, denoted Z_0 and Z_1 (short for Zone^0 and Zone^1), corresponding to the regions where the best response lies on the $r = 0$ and $r = 1$ planes, respectively:

$$Z_0 = \{(p, q) \in Z \mid \sigma(p, q) < 0\}, \qquad Z_1 = \{(p, q) \in Z \mid \sigma(p, q) > 0\}. \tag{2.9}$$

Since IS_3 projected on its base plane is $I_3 = \{(p, q) \in R \mid \sigma(p, q) = 0\}$, we have that Z is the disjoint union of Z_0, I_3, and Z_1. By our assumption on pure equilibria, $(1, 0)$ and $(0, 1)$ belong to Z_0, and $(0, 0)$ and $(1, 1)$ to Z_1.

The best-response correspondence BR_3 for player 3 is given by the set

$$BR_3 = (Z_0 \times \{0\}) \cup (I_3 \times [0, 1]) \cup (Z_1 \times \{1\}), \tag{2.10}$$

as shown in Figure 2.2 as an example.

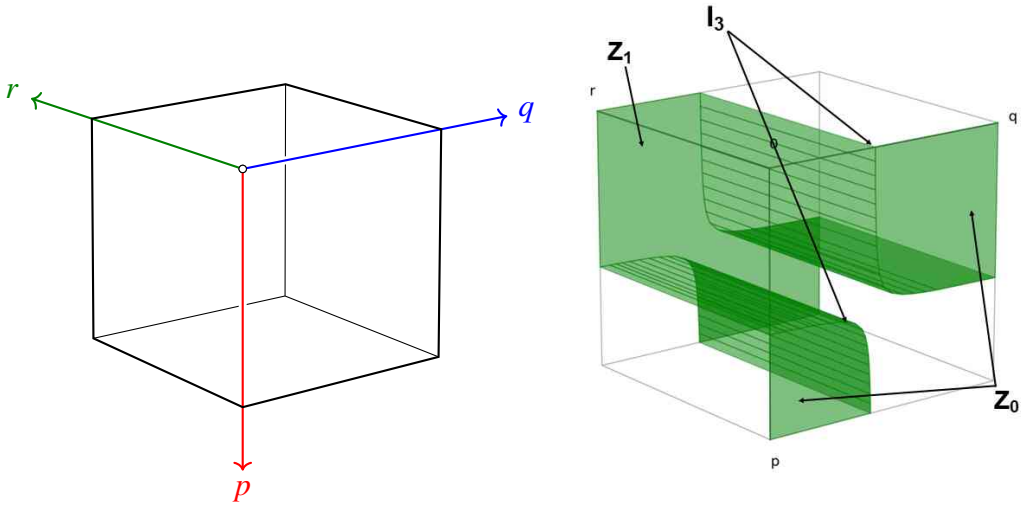


Figure 2.2.: Left: Cube of mixed-strategy probabilities (p, q, r) drawn (as in game table 1.1) down, right, and backwards. Right: A best-response surface of player 3.

We consider the indifference surfaces, projected on their base planes, I_1, I_2, I_3 as hyperbola functions h_1, h_2, h_3, respectively, restricted to those arguments where their domain and range are in $[0, 1]$.

From the assumption of four pure equilibria, it follows that the hyperbola functions h_1, h_2, and h_3 must each have two branches in their respective base planes, in order for the best-response correspondences to intersect at the specified pure equilibria. This assumption excludes simpler cases with fewer branches that can potentially lead to a smaller number of partially mixed equilibria. For example, if I_1 does not intersect the edge $r = 1$ of the top face, then IOS_3 cannot have a partially mixed equilibrium on the face $r = 1$ of the cube since it is not defined there.

Therefore, the hyperbola function h_i for $i = 1, 2, 3$ is restricted to a pair of suitable intervals $[0, u_i] \cup [v_i, 1]$ with $0 < u_i < v_i < 1$ such that $\{h_i(u_i), h_i(v_i)\} = \{0, 1\}$. We consider h_1 and h_2 as functions of r, and h_3 as a function of p (this is arbitrary), as follows:

$$\begin{aligned} q &= h_1(r), & I_1 &= \{(h_1(r), r) \mid r \in [0, u_1] \cup [v_1, 1]\}, \\ p &= h_2(r), & I_2 &= \{(h_2(r), r) \mid r \in [0, u_2] \cup [v_2, 1]\}, \\ q &= h_3(p), & I_3 &= \{(p, h_3(p)) \mid p \in [0, u_3] \cup [v_3, 1]\}. \end{aligned} \tag{2.11}$$

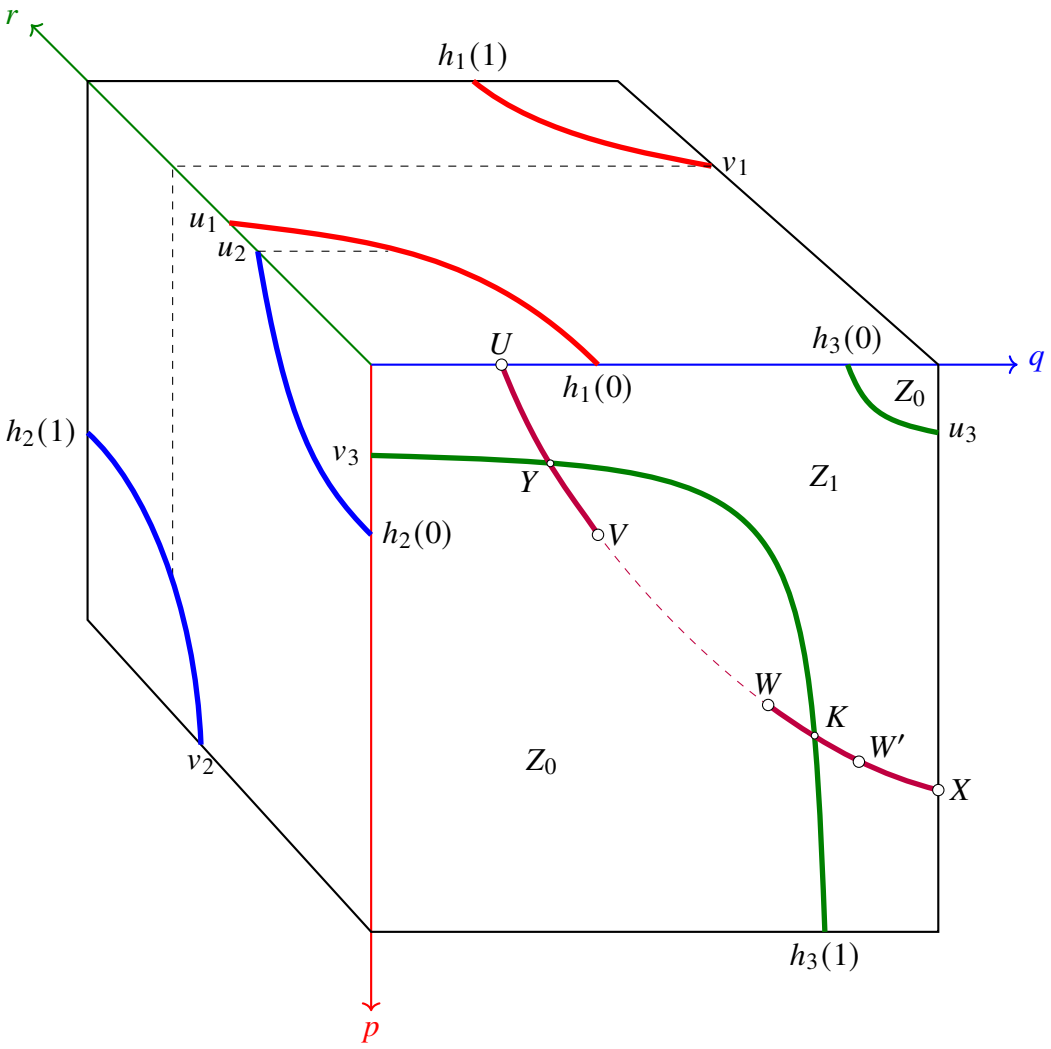


Figure 2.3.: The indifference sets I_1 (red branches, top face of the cube), I_2 (blue branches, left face of the cube) and I_3 (green branches, front face of the cube) drawn as hyperbola functions $h_1(r)$, $h_2(r)$, and $h_3(p)$, respectively. The purple branch segments of a hyperbola on the front face of the cube are the projections $(p, q) = (h_2(r), h_1(r))$ of the curve $\text{IOS}_3 = (h_2(r), h_1(r), r)$ that is the intersection of indifference surfaces of players 1 and 2, here for $r \in [0, u_2] \cup [v_1, 1]$ because $u_2 < u_1 < v_2 < v_1$.

Figures 2.3 and 2.4 show examples of these hyperbola functions, drawn on the sides of the mixed-strategy cube for players 1, 2, 3 in the colours red, blue, and green, respectively.

We show how the *completely mixed* equilibria of the game can be completely identified by considering the two-dimensional square Z, drawn as the front face of the cube (where $r = 0$) in the figures and projecting IOS_3 on this face.

The Nash equilibria of the game are the elements of the intersection of the three best-response correspondences $BR_1 \cap BR_2 \cap BR_3$. The strict surfaces of different players can only intersect on the edges of the cube as they are located on different faces. In this setting, they can only intersect in the pure equilibria, as there are no equilibria on edges. Therefore, to find the partially mixed equilibria, we study the intersection of opponents'

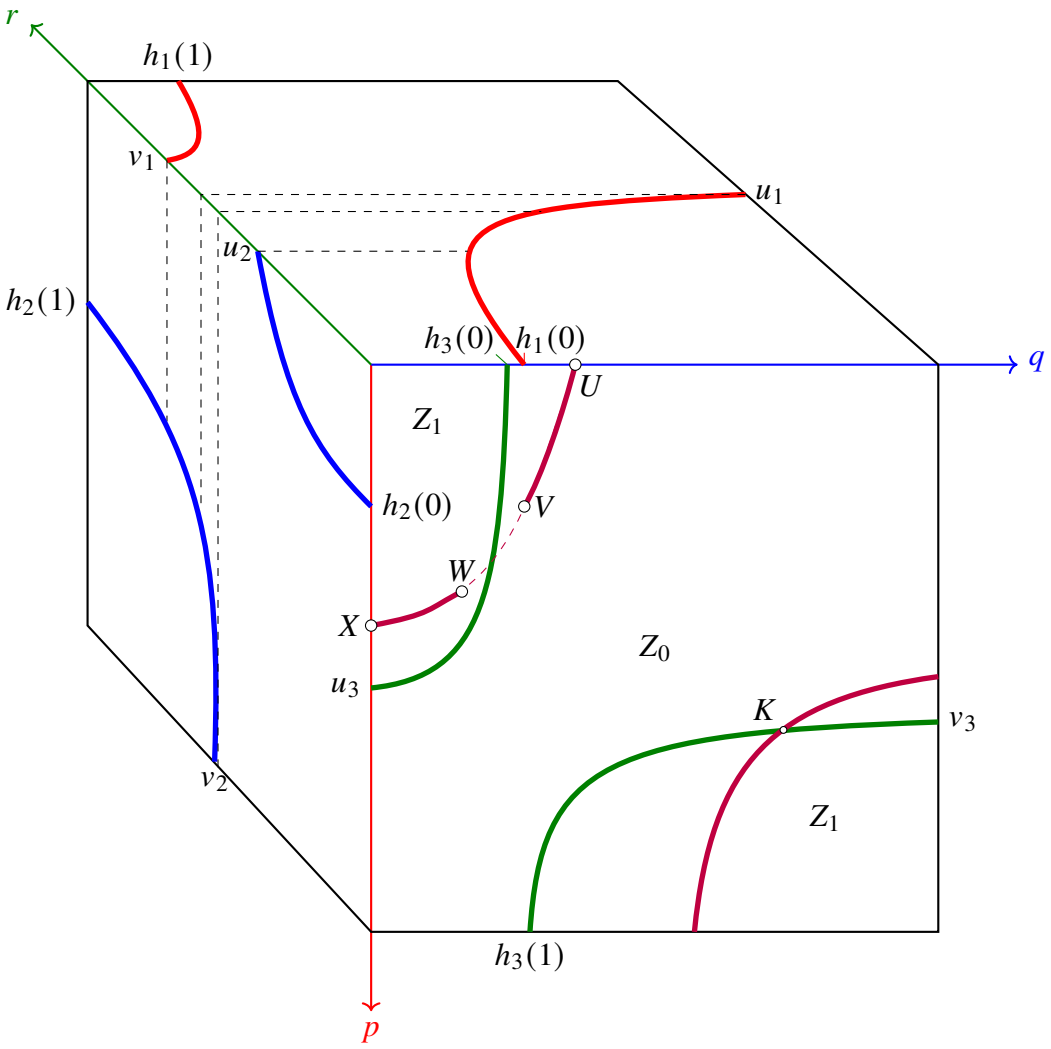


Figure 2.4.: Here, the intersection of the best-response correspondences of players 1 and 2 consists of three curve segments, given $(h_2(r), h_1(r), r)$ for $r \in [0, u_2] \cup [v_2, u_1] \cup [v_1, 1]$ because $u_2 < v_2 < u_1 < v_1$. Projected to the front face, the endpoints V and W of the first and last segment are in different zones Z_0 and Z_1 and define partially mixed equilibria for $r = 0$ and $r = 1$, respectively. There is only one completely mixed equilibrium, shown with its projection K.

indifference surfaces (IOS curves) with the players' strict surfaces. The completely mixed equilibria are the intersection of all players' indifference surfaces.

The set $\mathrm{IOS}_3 = \mathrm{IS}_1 \cap \mathrm{IS}_2$, which is the intersection $BR_1 \cap BR_2$ restricted to the interior of the cube, is the curve parametrised by r given by $(p, q) = (h_2(r), h_1(r))$.

Hereby, the IOS_3 curve (including its endpoints for $r = 0, 1$) consists of up to three continuous parts, where r is defined both for h_2 and h_1 so that the hyperbolas are restricted to $[0, 1]^2$, that is, for

$$r \in R = ([0, u_2] \cup [v_2, 1]) \cap ([0, u_1] \cup [v_1, 1]), \tag{2.12}$$

which may be a union of two intervals (as in Figure 2.3) or three intervals (as in Figure 2.4), depending on the relative order of u_1, v_1, u_2, v_2, where u_1, v_1 are the endpoints of I_1 and u_2, v_2 are the endpoints of I_2.

The key observation is that the projection of $IOS_3 = (h_2(r), h_1(r))$ to Z is again a *hyperbola* (or a line), which we denote by H, with the additional restriction (2.12). Namely, as a function of p, this composition is given by

$$q = h_1(h_2^{-1}(p)) =: H(p), \tag{2.13}$$

which, within Z, is either a hyperbola or a linear function, as established by Lemmas 2.4 and 2.6.

Note that the hyperbola function $H(p)$ is a function defined on all of $\mathbb{R}$ except for up to two points for p where $h_2^{-1}(p)$ and $h_1(h_2^{-1}(p))$ are undefined.

Projected on Z, this hyperbola H is drawn on the front face of the cube in Figures 2.3 and 2.4 in purple (including the dashed curve). The intersection of H with the (green) hyperbola I_3 is given by the pairs (p, q) where $q = H(p) = h_1(h_2^{-1}(p)) = h_3(p)$, which are the projections of completely mixed equilibria onto the front face. In Figure 2.3, these are the points Y and K. The third-dimensional coordinate of these points is $r = h_2^{-1}(p)$, which defines the completely mixed equilibrium (p, q, r).

A line and a non-degenerate hyperbola have at most two points in common, and so do two hyperbolas by Lemma 2.7. Hence, a generic $2 \times 2 \times 2$ game has at most two completely mixed equilibria, which are intersections of H and h_3. We emphasise here that these two hyperbolas and their intersections are not limited to the range $[0, 1]$.

Imagine the endpoints of IOS_3 on the front and back faces of the cube are $(p', q', 0)$ and $(p'', q'', 1)$. These points correspond to partially mixed equilibria if, by the definition (2.10) of BR_3, the points are in different zones, since they are on different faces of the cube. We define their projections V and W to Z as

$$V = (p', q') = (h_2(0), h_1(0)) \in Z_0\,, \qquad W = (p'', q'') = (h_2(1), h_1(1)) \in Z_1\,. \tag{2.14}$$

Examples of these points V and W are shown in Figures 2.3 and 2.4, projected onto the front face, although $(W, 1)$ is actually located on the back face in the 3D representation. Only in Figure 2.4 we have that $V \in Z_0$ and $W \in Z_1$ and hence the equilibrium property. This is not the case for Figure 2.3 as only $(V, 0)$ is an equilibrium and $(W, 1)$ is not.

For V and W to represent equilibria and therefore be in different zones, they must be *separated by a branch of* h_3. For example, if W in Figure 2.3 were located at the position of W', then $W' \in Z_1$. The claim is that, in this case, we would lose the completely mixed equilibrium (K, r) for a suitable r.

The main idea behind the rest of the proof is that the configuration of H (purple) and h_3 (green) depicted in Figure 2.5, where V and W are in different zones and the game has two completely mixed equilibria K and Y, is not possible because V and W *are on the same*

branch of H. This then results in the loss of the mixed equilibrium that would exist between them if the game were not restricted to the $[0, 1]^3$ cube, on the dashed line in Figure 2.4.

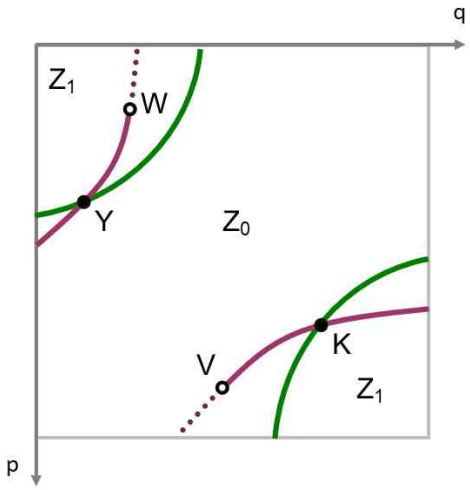


Figure 2.5.: The impossible intersection of H and h_3.

Because V and W are of the form $(h_2(r), h_1(r))$ for some r and therefore of the form $(p, h_1(h_2^{-1}(p))$ (where $r = h_2^{-1}(p)$), we have argued that V and W are part of the hyperbola or line H on the front face Z, by Lemma 2.6. We now show that they belong to the same branch of H, with a missing part (shown as a dashed curve in Figures 2.3 and 2.4) where H *does not intersect* with the hyperbola h_3 if V and W are in different zones (as in Figure 2.4). Because this intersection is missing, it accounts for the lost completely mixed equilibrium.

As part of a branch of H, consider the path (projected to the square Z) of pairs $(h_2(r), h_1(r))$ which starts at V for $r = 0$ and is *increasing* in r. In both Figure 2.3 and 2.4, $p = h_2(r)$ is a decreasing function of r and becomes zero when $r = u_2$ (by our assumption on pure equilibria, $u_2 < 1$, otherwise $(0, 0, 1)$ would not be a pure equilibrium).

In Figure 2.3, $q = h_1(r)$ is decreasing in r, and and in Figure 2.4, $h_1(r)$ is increasing in r. When $r = u_1$, we obtain $h_1(r) = 0$ in Figure 2.3 respectively $h_1(r) = 1$ in Figure 2.4. In both cases we plotted here $u_1 > u_2$, so the path $(h_2(r), h_1(r))$ ends when $r = u_2$ at the point U at the top edge of the front square Z of the cube, where $q = h_1(u_2)$. This could equally have ended on the left or right edge, depending on the order of u_1 and u_2 or if h_2 was increasing.

A similar consideration applies to the path of pairs $(h_2(r), h_1(r))$ when started at the point W for $r = 1$ and *reducing* r, except that the path now goes in the opposite direction and terminates at some other edge of Z, at the point X. Here in both figures $X = h_2(v_1)$, because $v_1 > v_2$ and $h_1(r), h_2(r) \in [0, 1]$ when $r \in [v_1, 1]$; if $v_1 < v_2$ then that path of pairs $(h_2(r), h_1(r))$ would end at the bottom edge of Z when $r = v_2$.

To summarise, the segments of the IOS_3 curve correspond to the intervals in (2.12). The endpoints of these segments on the faces of the cube are determined by the monotonicity of h_1 and h_2 (increasing or decreasing), as well as the order of u_1, v_1, u_2, and v_2. For instance, in Figure 2.4, the segment between X and W corresponds to values of $r \in [v_1, 1]$, while the

segment between V and U corresponds to values of $r \in [0, u_2]$. The third segment, which may contain the completely mixed equilibrium not discussed in this proof, corresponds to values of $r \in [v_2, u_1]$.

We prove that the points V and W are on the same branch of the hyperbola H, defined by the pairs $(h_2(r), h_1(r)) = (p, q)$ where $q = H(p))$ when p is also allowed to take values between $h_2(0)$ and $h_2(1)$. This interval corresponds to the part of the hyperbola h_2 that is not located on the $[0, 1]^2$ face on the left, because $r = h_2^{-1}(p)$ becomes negative or greater than 1. Consequently, for the hyperbola $H = h_1(h_2^{-1}(p))$, this part is shown as the dashed line, as it lies outside the $[0, 1]^3$ cube.

Let

$$p \mapsto r = h_2^{-1}(p) = b + \frac{c}{p - a}, \qquad r \mapsto q = h_1(r) = B + \frac{C}{r - A}\,. \tag{2.15}$$

Note that the asymptotes a, b, A, B of both hyperbolas in (2.15) are between 0 and 1. If $c > 0$ then $h_2(0) < h_2(1)$ as in Figures 2.3 and 2.4; if $c < 0$ then $h_2(0) > h_2(1)$.

First, suppose that $b = A$. According to Lemma 2.6, the composition $p \mapsto q = h_1(h_2^{-1}(p)) = H(p)$ is then a linear function. The corresponding line (obtained when $p \in \mathbb{R} - \{a\}$) contains the points V and W, which determines the line uniquely (H as a line has only one branch). This line, which can have at most two intersections with I_3, loses the gap of the line between W and V in Z when

$$p \in (h_2(0), h_2(1)) - \{a\} \;\text{ (if } c > 0), \quad p \in (h_2(1), h_2(0)) - \{a\} \;\text{ (if } c < 0), \tag{2.16}$$

respectively. This implies that the completely mixed equilibrium between V and W does not exist. While this intersection point of H and h_3 must exist for V and W to be in different zones, it is not a completely mixed equilibrium of the game because, for values in the interval (2.16), r does not lie within $[0, 1]$.

Now assume $b \neq A$. Then, as $r \to -\infty$ and $r \to \infty$, the value of $p = h_2(r)$ approaches a arbitrarily closely. Consequently, $q = h_1(r)$ approaches B arbitrarily closely.

This implies that the gap between W (as $r \to \infty$) and V (as $r \to -\infty$) disappears, which is only possible if V and W lie on the same branch of H. According to Lemma 2.8, the two branches of a hyperbola have a positive distance between them.

Therefore, similar to the line case, the intersection point of h_3 and H, $(p, q) = (a, B)$, which explains why V and W are in different zones, is not a mixed equilibrium of the game. This is because, for p being in the interval (2.16), we have that $r \notin [0, 1]$.

The two hyperbolas H and h_3, when unrestricted, intersect in at most two points, but one of these intersection points is on the said gap of the branch of H between W and V as we saw in the cases above. Hence, H and h_3 lose one of the intersections, and the game has at most one completely mixed equilibrium, as claimed. □

Corollary 2.11. *Any non-degenerate* $2 \times 2 \times 2$ *game with four pure and two completely mixed equilibria has at most three partially mixed equilibria.*

Proof. Theorem 2.10 establishes that, in the presence of two completely mixed equilibria, no two opposite faces of the cube can both contain non-pure partially mixed equilibria. Therefore, a game with four pure and two completely mixed equilibria can have at most one non-pure partially mixed equilibrium per IOS curve, which is one along each axis, yielding a maximum of three partially mixed equilibria. □

Corollary 2.11 proves an upper bound on the number of partially mixed equilibria by assuming the maximum number of pure and completely mixed equilibria, which gives an upper bound of nine on the total number of equilibria.

In general, the projected IOS curves on any face of the cube form hyperbolas (according to Lemma 2.6). Therefore, they have at most two intersections on the two opposing faces of the cube. These intersections could be partially mixed equilibria, namely if they are also on the best-response correspondence of the remaining player. Thus, the three IOS curves can lead to at most 6 partially mixed equilibria in non-degenerate games.

To better understand these hypothetical partially mixed equilibria, assume that our $2 \times 2 \times 2$ game has four pure equilibria. Each face of the cube is given by one player playing a pure strategy and defines a 2×2 game between the other two players with two pure and one mixed equilibrium. This mixed equilibrium defines a partially mixed equilibrium of the three-player game if the fixed player's pure strategy is a best response.

Harsanyi [24] proved that the total number of Nash equilibria in a generic game is odd. Corollary 2.11 leaves open the possibility that there exists a partially mixed equilibrium on each face of the cube. Together with the four pure and one completely mixed equilibrium, this would yield a total of $4 + 6 + 1 = 11$ equilibria.

We show that this configuration is not possible by invoking the concept of the *index* of an equilibrium. This concept also rules out the scenario in which there are four pure and five partially mixed equilibria, but no completely mixed equilibrium (which would still yield only nine equilibria).

2.2.1. Concept of Equilibrium Index

For any finite game, its set of Nash equilibria in mixed strategies is the union of closed connected sets, called the (topological) equilibrium *components*; this follows from the fact that equilibria are the fixed points of a suitable continuous map on the set of mixed-strategy profiles, such as the map used by Nash [47]. In a generic game, all equilibria are isolated points; that is, all components are singletons.

The *index* is an integer that is assigned to every equilibrium component. It has the following properties for any *generic* game (we deliberately talk about generic rather than non-degenerate games in the remainder of this section).

1. Every equilibrium is isolated and has index $+1$ or -1.

2. In a two-player game, the endpoints of any path computed by the algorithm of Lemke and Howson [42] are equilibria of opposite index, including an *artificial equilibrium* (which is not a Nash equilibrium) that has index -1.
3. Every pure-strategy equilibrium has index $+1$.
4. The sum of the indices over all Nash equilibria is $+1$.
5. The index only depends on the payoffs in the equilibrium support.
6. In a 2×2 game with two pure equilibria, the mixed equilibrium has index -1.

For two-player games, the definition of the index, and proof of these properties, is due to Shapley [63]. For a streamlined account, see von Stengel [66], with a summary in [32]. Gül, Pearce and Stacchetti [22] showed that these properties extend to n-player games by defining the index via the fixed points of a suitable map, and a corresponding definition of an index for such fixed points.

In a mixed equilibrium of a generic two-player game, the players' mixed strategies have supports of equal size, and, assuming positive payoffs, the sub-matrices of the two players' payoff matrices for these supports have nonzero determinants. The index of the equilibrium is defined as the product of the signs of these determinants for each player, times -1 if the support size is even. It is easy to see that this implies properties 3., 5., and 6. above.

Property 2. proves the important Property 4.İt also implies Property 3. as every pure equilibrium can be reached from the artificial equilibrium via a Lemke–Howson path by selecting an appropriate missing label.

Properties 1. and 4. together imply that there is exactly one more Nash equilibrium of index $+1$ than of index -1. Moreover, the total number of Nash equilibria in a generic game is odd, since each Lemke–Howson path connects two unique equilibria of opposite index, and the artificial equilibrium should not be counted. The odd number of equilibria was shown independently by Harsanyi [24] for n-player games.

For an n-player game, the definition of the index is more complex. We think that it is possible to give a much simpler self-contained definition and proof of the above properties for a $2 \times 2 \times 2$ game. We discuss this further below. For now, we assume the above properties also for 3-player games.

The index of partially mixed equilibria

Consider a generic $2 \times 2 \times 2$ game with 4 pure equilibria. Then if one player plays a pure strategy, the resulting 2×2 subgame between the other two players has two pure equilibria, and a mixed equilibrium of index -1 by property 6. (Note that our normalisation in (1.1) allows for positive and negative payoffs; to apply the definition of the index in terms of signs of determinants, all payoffs need to be shifted to become positive, but this is otherwise immaterial.)

The mixed equilibrium of the 2×2 subgame defines a partially mixed equilibrium of the $2 \times 2 \times 2$ game if the pure strategy of the fixed player is a best response. Crucially, the index of this partially mixed equilibrium (with two further pure equilibria on the same face of the mixed-strategy cube) remains -1. This is a consequence of property 5., because the pure-strategy player is inactive. This leads to the following lemma.

Lemma 2.12. *No generic* $2 \times 2 \times 2$ *games can have 4 pure, 6 partially mixed equilibria, and 1 completely mixed equilibrium, nor can it have 4 pure, 5 partially mixed, and 0 completely mixed equilibria.*

Proof. For a game with 4 pure equilibria, all partially mixed equilibria have index -1. The described possibilities have too many equilibria of negative index to achieve an overall sum of 1 over all indices according to property 4. If there are 6 partially mixed equilibria with index -1, then the remaining 5 equilibria will not lead to sum 1 even if they all have index $+1$. The same applies to a game with 4 pure, 5 partially mixed, and 0 completely mixed equilibria. □

2.2.2. Completing the Proof of Main Theorem

Using the results of the previous sections, we now conclude the proof of Theorem 2.9. For $2 \times 2 \times 2$ games, we have established the following:

1. In Corollary 2.2, we showed that the number of pure equilibria in non-degenerate games cannot exceed four. The main idea was that if there are two equilibria on the same edge, any point between them must also be an equilibrium. This argument also applies to the class of games with finitely many equilibria.
2. In Theorem 1.7, we proved that the number of completely mixed equilibria in non-degenerate games is at most two.
3. For partially mixed equilibria, the endpoints of the IOS curves on the faces of the cube serve as potential candidates. Since there are three IOS curves, there can be at most six partially mixed equilibria. In Corollary 2.11, we proved that if a game has four pure and two completely mixed equilibria, then there can be at most three partially mixed equilibria, giving a total of nine equilibria. This result holds for non-degenerate games.

4. The fact that the number of equilibria must be odd was established by both Harsanyi [24] and Wilson [73], under their respective definitions of n-player non-degenerate games.
5. The concept of equilibrium index is defined for generic games. In Lemma 2.12, based on prior work on index theory, we proved that the combination of four pure, six partially mixed, and one completely mixed equilibrium is impossible in generic games.

All the results above apply to generic games, as generic games are a subset of both non-degenerate games and games with finitely many equilibria. Therefore, we have completed the proof of the upper bound of nine equilibria in generic games, as no other combination of equilibrium types can yield a higher number.

Furthermore, from the equilibrium index theory, we saw that this upper bound can only be attained through the following combinations:

- 4 pure, 3 partially mixed, and 2 completely mixed equilibria, or
- 4 pure, 4 partially mixed, and 1 completely mixed equilibrium.

□

2.2.3. Discussion and Ongoing Work

The results used in the proof of the upper bound of 9 equilibria in Section 2.2.2 indicate that many of the arguments extend beyond the class of generic games. In particular, to generalise Theorem 2.9 to non-degenerate games and establish the same upper bound for this broader class, it suffices to verify parts 4. and 5. for non-degenerate games.

Part 4., concerning the oddness of the number of equilibria, has already been established for an alternative definition of non-degenerate games. To extend this result to our setting, it remains to demonstrate either the equivalence or inclusion of our definition of non-degeneracy within the framework introduced by Wilson [73].

Regarding part 5., for the extension of the index properties to non-degenerate games, further investigation is required.

Much of the literature on the equilibrium index (which we do not survey here) relies on advanced concepts from topology and fixed point theory. One key reason for this is that the index is often studied in the context of equilibrium components, which may be non-singleton sets when the game is not generic. In such cases, the index of a component is defined by perturbing the game to a nearby generic one and summing the indices ($+1$ or -1) of the resulting nearby equilibria. (It must be shown that this definition is well-defined, i.e., independent of the specific perturbation, which is not trivial.)

One question that arises in this context concerns the "strategic stability" of an equilibrium component, that is, whether the component can vanish entirely, with no nearby equilibrium, under a small perturbation of the payoffs. A necessary (though not sufficient) condition for such instability is that the component has index zero (see, e.g., [15]).

Another question is the relationship between the index and the properties of evolutionary dynamics. For example, the index can provide insights into which equilibria are dynamically stable; see DeMichelis and Germano [14] and Demichelis and Ritzberger [15]. In the case of generic games, Hofbauer [30] has shown that only equilibria with index +1 can exhibit dynamic stability.

For our $2 \times 2 \times 2$ games, an accessible geometric intuition of the index should be the *orientation* of the normal vectors to the three best-response surfaces that meet at the equilibrium. This orientation (right-handed or left-handed) corresponds to the sign of the determinant of these normal vectors. It is easy to assign a normal vector to a player who plays a pure strategy, namely the respective unit vector. However, normal vectors are not uniquely defined where the best-response surfaces are not differentiable, and the partially mixed equilibria appear exactly at the "edges" of these surfaces. A proper definition should ignore the unused payoffs of the pure-strategy player, in line with property 5. of index in Section 2.2.1.

The fact that the overall index sum is +1 (property 4. of index) should then be the consequence of applying Lemke-Howson paths in our three-player setting. These paths have been generalised by Rosenmüller [56] and Wilson [73] to N-player games. It would be useful to demonstrate them for our $2 \times 2 \times 2$ games and then to show property 2. of index, which should not be too difficult.

For games with more than two players, Lemke–Howson paths are no longer line segments but curves. In this context, the equivalence of the two definitions of non-degeneracy, as mentioned in the discussion on extending part 4, becomes crucial. This is because non-degeneracy is required for the Lemke–Howson algorithm, not only at equilibria but along the entire path. The corresponding technical condition is likely to be the "transversality" of best-response correspondences, meaning that their intersections have the expected dimension reduction, similar to the intersection of linearly independent hyperplanes. For a recent discussion, see Hertling and Vujic [29].

An important point in the proposed extension is that our Definition 1.4 of non-degeneracy allows for games where indifference surfaces take the form of degenerate hyperbolas in the base plane (which are clearly not generic). These can still yield isolated equilibria, provided the asymptotes differ across players. Such games are considered by Hertling and Vujic [28] in the context of n-player games with two actions per player, as their mixed equilibria are relatively tractable and analytically accessible. For the case $n = 3$, we study these games in Theorem 3.2 in the Appendix.

Conclusions to Part I

The computational complexity of finding Nash equilibria is often concerned with asymptotic properties such as PPAD-hardness. In Section 6.3.3 in Part II of this thesis, we discuss that PPAD-hardness, which concerns the problem of finding one Nash equilibrium, does not imply intractability for games with a few hundred strategies. In contrast, finding *all* Nash equilibria of a game definitely requires exponential effort, because the game may have exponentially many Nash equilibria.

However, many concrete games are small and would benefit from a complete analysis of all their Nash equilibria. This has been done for two-player games, but it is significantly more difficult for general games with more than two players. Solvers based on solving polynomial equations and inequalities often fail in degenerate cases, which can have an infinite number of equilibria [57].

We considered the simplest multiplayer game, namely three players with two strategies each, which had not been done before in this generality.

In Chapter 1, we studied the best-response correspondence of each player and its representation in 3D. We investigated the intersection of best-response correspondences algebraically and streamlined the corresponding expressions using determinants in the quadratic equations (1.20), (1.21), and (1.22), exploiting the symmetry of the setup. We took advantage of the fact that mixed-strategy profiles can be displayed in a three-dimensional cube.

We extended the definition of non-degeneracy from 2-player games to our setting, allowing identification of edge-case games based solely on the players' payoffs. We showed that non-degenerate games can only have finitely many singleton equilibria. Moreover, we proved that in such games, the number of completely mixed equilibria does not exceed two.

To compute all equilibria of a general $2 \times 2 \times 2$ game, we presented two algorithms. The first algorithm computes the pure and partially mixed equilibria by investigating all six subgames. An important insight is that computing partially mixed equilibria is a fruitful approach. For larger games, this involves reducing the number of players and analysing the subgames, which can already yield useful information about equilibria with relatively little additional effort.

The second algorithm enumerates the completely mixed equilibria by identifying the type of degeneracy (if any). Implementing this algorithm required a large number of case distinctions for different forms of degenerate games.

Another contribution of this chapter is a "proof of principle" that the complete equilibrium set can be computed and visualised in full, regardless of how degenerate the game is. We implemented a Python tool to compute the equilibria and visualise the games using 3D graphics, which is available for public use [34].

In Chapter 2, we focused on bounds on the number of equilibria in $2 \times 2 \times 2$ games. For non-degenerate games, we established an upper bound of nine on the number of equilibria under the assumption that there are four pure and two completely mixed equilibria. This result was achieved through a step-by-step analysis of the indifference surfaces and their intersection curves.

We showed that this upper bound can be realised in two different ways, and we explicitly demonstrated one case with 4 pure, 3 partially mixed, and 2 completely mixed equilibria. Furthermore, we discussed that by changing the payoffs, one of the completely mixed equilibria (of negative index) may move towards a face of the cube and become a partially mixed equilibrium.

Using results on the concept of the equilibrium index, we completed the proof of the upper bound of nine equilibria for the class of generic games. We also noted that most of our results apply to a broader class of non-degenerate games. The next step toward fully establishing the upper bound of nine equilibria for all non-degenerate games is to prove further properties of the index in this broader setting.

For analysing the Nash equilibria of larger games, it seems advisable to proceed incrementally in the same manner: identifying degenerate cases, computing partially mixed equilibria via subgame analysis, and employing algebraic solvers such as those used by Datta [13], under the assumption of non-degeneracy.

The main question in this context is what types of multiplayer games researchers aim to address. Restricting attention to models like polymatrix games, which rely on pairwise interactions [8, 21, 31], may be a promising next step in this direction.

3.

Appendix to Part I

3.1. Excluded Case of the Upper Bound Theorem

In our proof of Theorem 2.10, we excluded the case in which the indifference surface of at least one player takes the form of a degenerate hyperbola on the base plane.

Since these forms are unstable under small perturbations (as they change into non-degenerate hyperbolas), one might think that such cases do not require separate consideration because the game is not generic. However, our proof of Theorem 2.10 applies to the broader class of non-degenerate games, as defined in Definition 1.4. Non-degenerate games may include indifference surfaces that take the form of degenerate hyperbolas, as these do not necessarily lead to degeneracy in the game.

In fact, these cases were our starting point for proving the upper bound in Theorem 2.9, as their special structure simplifies the study of IOS curves. Nonetheless, degenerate hyperbolas can be viewed as the limiting case of hyperbolas in which the branches approach their asymptotes.

The following lemma assists us in the analysis of these cases.

Lemma 3.1. *In non-degenerate games, completely mixed equilibria do not lie on a line parallel to any coordinate axis within the cube of mixed strategies. In other words, the completely mixed equilibria cannot have two common coordinates.*

Proof. The reasoning is similar to that of Lemma 2.1, with the key distinction that, in this case, equilibria must lie on all indifference surfaces. Therefore, the inequalities used in Lemma 2.1 are replaced by equalities.

It follows similarly that if two completely mixed equilibria lie on a line parallel to any coordinate axis, then every point on the convex combination of these equilibria would also satisfy the equilibrium conditions, leading to degeneracy, which contradicts the assumption. □

We now study the cases involving degenerate hyperbolas in the following theorem.

Theorem 3.2. *Let G be a non-degenerate $2\times 2\times 2$ game with four pure and two completely mixed Nash equilibria, in which the indifference surface of at least one player takes the*

form of a degenerate hyperbola. Then, for any IOS_i curve of player $i \in \{1, 2, 3\}$, at most one endpoint of IOS_i corresponds to a partially mixed equilibrium.

Proof. We begin with the case where the indifference surfaces of all players are degenerate hyperbolas on their respective base planes. Then, we progressively replace one of these surfaces at a time with a non-degenerate hyperbola to study the subsequent cases.

As we focus solely on indifference surfaces of hyperbolic form in this theorem, we rewrite the expected payoff equations in a "product form" to highlight the degenerate cases of hyperbolas.

Thus, by considering $M, m, \mu \neq 0$ (i.e., only hyperbolas) in the expected payoff equations (1.9), and using the hyperbolic representation introduced in (1.14) with renamed parameters for clarity, we obtain:

$$\begin{aligned} S(q, r) &= (q - q_1) \times (r - r_2) - K_1 \\ s(r, p) &= (r - r_1) \times (p - p_2) - K_2 \\ \sigma(p, q) &= (p - p_1) \times (q - q_2) - K_3 \ . \end{aligned} \tag{3.1}$$

This formulation expresses the expected payoff equations as hyperbolas, with asymptotes given by $q = q_1$ and $r = r_2$ for player I, $r = r_1$ and $p = p_2$ for player II, and $p = p_1$ and $q = q_2$ for player III. This reflects the symmetry of the formulation, which is based on the cyclical order of the players as before.

Importantly, if $K_j = 0$ for any $j \in \{1, 2, 3\}$, then the corresponding indifference surface IS_j projected onto the base plane takes the form of a degenerate hyperbola.

As we study the upper bound on the intersections of indifference surfaces, we assume that all players' hyperbolic asymptotes lie within the open interval $(0, 1)$. Otherwise, some of the indifference surfaces would lie entirely outside the mixed strategy cube and, therefore, would not yield feasible equilibria. Hence, we assume $(p_i, q_i, r_i) \in (0, 1)^3$ for $i = 1, 2$, and $K_j \in \mathbb{R}$ for $j = 1, 2, 3$. The edge cases where the asymptotes are equal to 0 or 1 are already excluded by the assumption that the game is non-degenerate.

Moreover, the non-degeneracy of the game implies that $p_1 \neq p_2$, $q_1 \neq q_2$, and $r_1 \neq r_2$, as otherwise components of infinitely many equilibria could form in the interior of the cube, which would violate condition (ii) in Definition 1.4 of non-degenerate games.

Now, for each of the following cases, we show that under the assumption of having four pure and two completely mixed equilibria, any IOS_j curve, for $j \in \{1, 2, 3\}$, intersects the strict surface of player j (SS_j) in at most *one* partially mixed equilibrium.

Case 1: Degenerate hyperbolas for all players

In this case, we assume that the indifference surfaces of all players have the form of degenerate hyperbolas. Consequently, in equation (3.1), we have

$$K_1 = K_2 = K_3 = 0\,.$$

In this case, the intersection of any two indifference surfaces consists of three mutually perpendicular lines. Therefore, the IOS curves have the following symmetric form, with an example illustrated in Figure 3.1:

$$\mathrm{IOS}_1 = \begin{cases} (\mathbf{p}, q_2, r_1) & p \in [0,1] \\ (p_1, \mathbf{q}, r_1) & q \in [0,1] \\ (p_2, q_2, \mathbf{r}) & r \in [0,1] \end{cases} \quad \mathrm{IOS}_2 = \begin{cases} (\mathbf{p}, q_2, r_2) & p \in [0,1] \\ (p_1, \mathbf{q}, r_2) & q \in [0,1] \\ (p_1, q_1, \mathbf{r}) & r \in [0,1] \end{cases}$$

$$\mathrm{IOS}_3 = \begin{cases} (\mathbf{p}, q_1, r_1) & p \in [0,1] \\ (p_2, \mathbf{q}, r_2) & q \in [0,1] \\ (p_2, q_1, \mathbf{r}) & r \in [0,1] \end{cases}$$

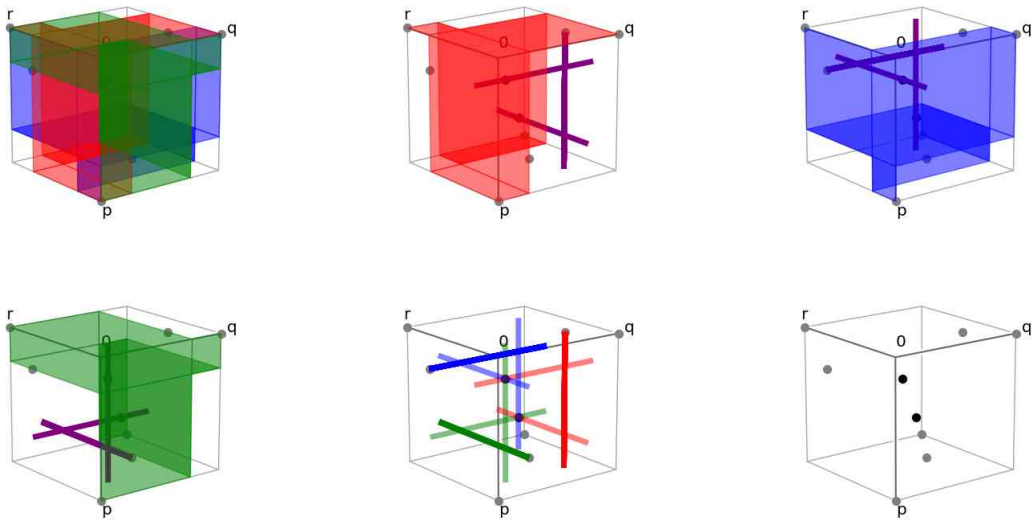


Figure 3.1.: Example of a game with degenerate hyperbolas as indifference surfaces of all players

Each IOS_j consists of three lines parallel to the coordinate axes, with exactly one line parallel to the j-th axis, along which player j's indifference surface (IS_j) extends. This line can intersect player j's strict surface SS_j at exactly one point, since both endpoints of the line lie within the same strict zone for player j. Consequently, each intersection of two indifference surfaces can produce at most one partially mixed equilibrium (PME).

Case 2: Two degenerate hyperbolas

Now, without loss of generality, we assume that IS_2 and IS_3 take the form of degenerate hyperbolas, while IS_1 is a non-degenerate hyperbola. That is, $K_1 \in \mathbb{R} \setminus \{0\}$ and $K_2 = K_3 = 0$ in (3.1).

$$S(q,r) = (q - q_1) \times (r - r_2) - K_1$$

$$s(r,p) = (r - r_1) \times (p - p_2)$$

$$\sigma(p,q) = (p - p_1) \times (q - q_2) \quad .$$

Figure 3.2 illustrates an example of this configuration.

The curve IOS_1 remains the same as in the previous case, consisting of three perpendicular lines. The argument from the previous case still applies: only one of these lines aligns with the direction of IS_1. Since this is a straight line, it can intersect the strict surface SS_1 in at most one PME.

Now, we turn our attention to the other two intersection curves.

$$IOS_3 = IS_1 \cap IS_2 = \begin{cases} (\mathbf{p}, \frac{K_1}{r_1 - r_2} + q_1, r_1) & p \in [0,1] \\ (p_2, \mathbf{q}, \mathbf{r}) & (q - q_1) \times (r - r_2) = K_1, \quad q, r \in [0,1] \end{cases}$$

$$IOS_2 = IS_3 \cap IS_1 = \begin{cases} (\mathbf{p}, q_2, \frac{K_1}{q_2 - q_1} + r_2) & p \in [0,1] \\ (p_1, \mathbf{q}, \mathbf{r}) & (q - q_1) \times (r - r_2) = K_1, \quad q, r \in [0,1] \end{cases}$$

Each of the curves IOS_2 and IOS_3 consists of:

(i) a vertical line parallel to the p-axis,

(ii) a hyperbola fully contained on a plane with fixed p-coordinate.

We now prove the claim for IOS_3; the same argument applies to IOS_2.

Line (i) of IOS_3 is parallel to the p-axis, so it cannot intersect SS_3. Therefore, any possible endpoints of IOS_3 that intersect SS_3 must lie on the hyperbola part (ii).

Moreover, the game has two completely mixed equilibria (CMEs). According to Lemma 3.1, at most one of these equilibria can lie on the line (i). Consequently, the hyperbola (ii) must pass through at least one CME. This hyperbola lies on the plane $p = p_2$ and therefore cannot intersect the IS_3 on the plane $p = p_1$ (since $p_1 \neq p_2$). Thus, the remaining CME must lie on the intersection of this hyperbola with the plane $q = q_2$ of IS_3.

We claim that no indifference surface IS_3 in the form of a degenerate hyperbola can intersect the hyperbola part (ii) of IOS_3 in more than one PME.

Now, consider the two endpoints of this hyperbola on the faces $r = 0$ and $r = 1$ of the cube, which are the possible candidates for PMEs. The hyperbola is entirely located on one side of the asymptote plane $p = p_1$ of IS_3. Therefore, considering the strict zones of IS_3 (which is a degenerate hyperbola), the two endpoints of hyperbola (ii) lie in different zones only if the plane $q = q_2$ lies strictly between the two branches of the hyperbola (ii), without touching either branch.

This contradicts the existence of a CME on the intersection of the hyperbola and the plane $q = q_2$. Hence, the endpoints of IOS_3 on the faces $r = 0$ and $r = 1$ project to the same strict zone of player III, and therefore at most one of them can be a PME.

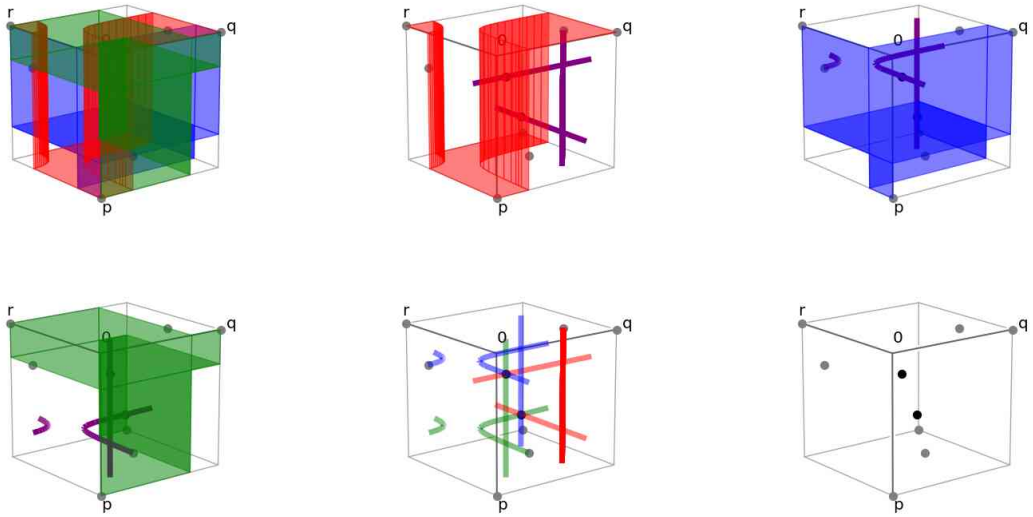


Figure 3.2.: Example of a game with two degenerate hyperbolas and one non-degenerate hyperbola as indifference surfaces.

Case 3: One degenerate hyperbola

Here, we assume that players I and III have indifference surfaces that are non-degenerate hyperbolas on their respective base planes. Thus, we define:

$$\begin{aligned} S(q,r) &= (q-q_1) \times (r-r_2) - K_1 \\ s(r,p) &= (r-r_1) \times (p-p_2) \\ \sigma(p,q) &= (p-p_1) \times (q-q_2) - K_3\,, \end{aligned}$$

where $K_1, K_3 \in \mathbb{R} \setminus \{0\}$, and $K_2 = 0$.

The IOS_1 and IOS_3 curves have the same structure as IOS_3 in the previous case, consisting of one hyperbola and one line. We therefore adapt the same idea to this case, where the indifference surface of the player has changed to a non-degenerate hyperbola:

$$\mathrm{IOS}_1 = \mathrm{IS}_2 \cap \mathrm{IS}_3 = \begin{cases} (i): & (p_2, \frac{K_3}{p_2-p_1} + q_2, \mathbf{r}) & r \in [0,1] \\ (ii): & (\mathbf{p}, \mathbf{q}, r_1) & (p-p_1)\times(q-q_2) = K_3, \quad p,q \in [0,1] \end{cases}$$

$$\mathrm{IOS}_3 = \mathrm{IS}_1 \cap \mathrm{IS}_2 = \begin{cases} (i): & (\mathbf{p}, \frac{K_1}{r_1-r_2} + q_1, r_1) & p \in [0,1] \\ (ii): & (p_2, \mathbf{q}, \mathbf{r}) & (q-q_1)\times(r-r_2) = K_1, \quad q,r \in [0,1] \end{cases}$$

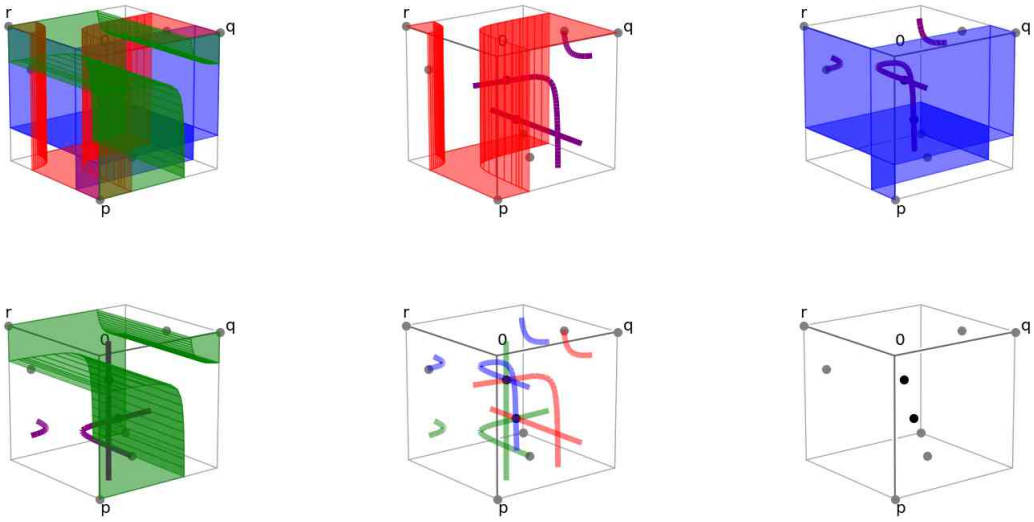


Figure 3.3.: Example of a game with one degenerate hyperbola and one non-degenerate hyperbola as indifference surfaces.

We now consider $\text{IOS}_1 \cap \text{IS}_1$, to illustrate how the proof adapts when applied to a different player. The same reasoning applies to $\text{IOS}_3 \cap \text{IS}_3$.

As in the previous case, the linear component (i) of IOS_1 cannot intersect IS_1 in more than one CME. Therefore, the hyperbolic component (ii) of IOS_1 must intersect IS_1 in at least one CME to satisfy the assumption that the game has two CMEs. Additionally, the line (i) cannot intersect SS_1, since it is parallel to the r-axis.

Note that IS_1 is a hyperbola on the $q \times r$ base plane, extended in the p direction. Hence, specifying a value of r uniquely determines a corresponding value of q on this hyperbola, while p can vary freely in $[0, 1]$, resulting in a vertical line segment parallel to the p-axis on IS_1.

Because the hyperbola (ii) of IOS_1 lies on the plane $r = r_1$, it can intersect IS_1 in at most one CME point (p', q', r_1). This is because fixing $r = r_1$ uniquely determines q' on IS_1, and this q' in turn uniquely determines p' on the hyperbola (ii) of IOS_1.

Therefore, following the same logic as in the previous case, the endpoints of hyperbola (ii) on the $r = 0$ and $r = 1$ faces of the cube will lie in the same strict zone of player I. They are both on the same side of the CME, which lies on the boundary between strict zones. Thus, IOS_1 can intersect IS_1 in at most one PME.

The only remaining analysis in this case concerns $\text{IOS}_2 \cap \text{IS}_2$. Here, IOS_2 represents the intersection of two non-degenerate hyperbolas. The projection of IOS_2 onto the $p \times q$ plane has the form of IS_3, and the projection onto the $q \times r$ plane has the form of IS_1, as IOS_2 is the intersection of these surfaces. Since both IS_1 and IS_3 are non-degenerate hyperbolas, specifying the value of any one coordinate on this curve uniquely determines the other two coordinates via the corresponding hyperbola functions.

By assumption, IOS_2 intersects IS_2 at two CMEs. Note that IS_2, being a degenerate hyperbola, is the union of two planes: $r = r_1$ and $p = p_2$. By setting $r = r_1$ or $p = p_2$, the

coordinates of a CME on IOS_2 can be uniquely determined. Therefore, IOS_2 intersects each of these planes at exactly one CME.

Now consider the endpoints of IOS_2 on the $q = 0$ and $q = 1$ planes, which are potential PME candidates. Towards a contradiction, suppose that IOS_2 intersects SS_2 in two PMEs, meaning these two endpoints lie in different strict zones of player II. Given the structure of strict zones in non-degenerate hyperbolas, this can only happen if the endpoints lie on the same side of one of the planes $r = r_1$ or $p = p_2$, and on opposite sides of the other, as these planes define the zone boundaries. However, if the endpoints lie on opposite sides of one of these planes, then IOS_2 would not intersect that plane, contradicting the assumption that IOS_2 intersects IS_2 in exactly one point on each of the planes $r = r_1$ and $p = p_2$.

This concludes the proof of the theorem. □

An interesting observation from the proof of Theorem 3.2 is that when one of the completely mixed equilibria approaches a face of the cube and becomes a partially mixed equilibrium located on an indifference surface of the player, it leads to a boundary case where the two endpoints of an IOS curve lie in different strict zones. This is possible because the indifference surface on the base plane forms the boundary between strict zones and can be considered as belonging to both. This transition reveals a new configuration of Nash equilibria, where a completely mixed equilibrium is effectively replaced by a partially mixed one. The resulting configuration consists of 4 pure, 4 partially mixed, and 1 completely mixed equilibrium, still yielding the same upper bound of 9 equilibria.

3.2. Instructions for Software

In this section, we provide a comprehensive guide to using our 3D graphics and Nash equilibrium computation software, available at [34]. The instructions are designed to ensure clarity and ease of use, while also serving as a resource for users interested in further developing the software.

To use the software, follow the steps below to configure the desired settings. Commands can be specified as arguments when running the `RUN.py` file. Alternatively, the file can be executed without arguments, prompting an interactive mode where the instructions are displayed, and commands can be entered dynamically.

```
Please enter the payoff in one of the following formats:

    Normalised form payoff:     -n [A,B,C,D] [a,b,c,d] [alpha,beta,gamma,delta]
    K-coefficients payoff:      -k [M,K,L,A] [m,k,l,a] [mu,kappa,lambda,alpha]
    Product payoffs (p-p1)(q-q2)-K1 etc.:-r [p1,q2,K1] [r1,p2,K2] [q1,r2,K3]
    Defined tests (0 to 40):    -t testID/No
    From file (default game.stf):   -f inputfile

Options can be specified as follows:
```

```
    Verbose (default 1: print, 0: no print):           -v 0
    Save as GIF (default 0: do not save, 1: save):     -g 1 name
    Orientation (default 1: horizontal, 0: vertical):  -o 0
    Write output to file (default: out.txt):           -w outputfile
    Show plot (default 1: show, 0: do not show):       -p 0
    Save plot (default 0: do not save, 1: save):       -s 1
    Only plot the best response of one player (1,2,3):     -b player
[Enter options, -q to quit]
```

The game payoffs can be input as a file in the `.stf` format, compatible with the Game Theory Explorer [19]. Alternatively, payoffs can be entered using one of the supported formats outlined below:

- Normalised payoff, as formulated in (1.2).
- Hyperbolic representation, as described in (1.9).
- Product representation, defined in (3.1).

Additionally, 47 predefined test cases are available for execution. These tests cover a variety of interesting scenarios, including both degenerate and non-degenerate cases. The test cases explore diverse sets of best response correspondences and degenerate equilibria of varying dimensions, as well as numerous non-degenerate cases with different numbers of equilibria, many of which have been presented above.

Several options are provided for saving outputs, including saving plots as GIF or PNG files and writing output to a file. Default values are specified alongside each command; to modify them, the desired values must be explicitly provided.

The resulting output plot, displayed after entering the game commands, is a 3D interactive plot. Each subplot can be rotated to view it from different angles. While the interactive 3D plot cannot be saved in its original format, it can be saved as a GIF file that rotates the plot to display views from different directions.

3.2.1. Code Structure

The `RUN.py` file serves as the main entry point for executing the software. Upon receiving the payoff matrix, it initialises an instance of the `PartialAlg` class. This class computes the K-coefficients of the payoffs, which are the coefficients of the expected payoff function as outlined in (1.9). These coefficients form the foundation for subsequent computations. The `PartialAlg` class also implements the algorithm described in Section 1 to compute all pure and partially mixed equilibria of the game. Furthermore, it solves the roots of the quadratic equations (1.20), (1.21), and (1.22), and provides detailed output in written form.

To compute all completely mixed equilibria and generate the 3D visual representation, an instance of the `GamePlot` class is created using the K-coefficient matrix as input.

The `GamePlot` class performs the following key tasks:

- Defines the best-response correspondence as the union of polygons representing indifference and strict surfaces.
- Computes the IOS curves, which result from the intersection of two indifference surfaces.
- Identifies completely mixed equilibria by computing the intersection of the IOS curve with the remaining indifference surface.
- Represents these intersections using the `IntersectionComponent` class, which generalises points, curves, or even 2D surfaces as equilibrium components.

Each `IntersectionComponent` is characterised by the following attributes:

- Base parameter: This can be p, q, or r.
- Base parameter interval: The range or value of the base parameter.
- Set of unrestricted parameters: Parameters that are not constrained in the given component.
- List of functions: For each parameter that is neither basic nor unrestricted, it is expressed as a function of the base parameter.

Some examples of `IntersectionComponent` for degenerate or non-degenerate cases:

- In **non-degenerate cases**, each `IntersectionComponent` corresponds to a single equilibrium point. Here, the base parameter interval consists of a single value, and the other parameters are expressed as functions of the base parameter. The set of unrestricted parameters is empty.
- In **degenerate cases**, such as when an entire plane represents equilibria (e.g., the plane $p = 0$), the base parameter is p with an interval of zero, and the set of unrestricted parameters includes both q and r.

The `GamePlot` class further provides the following functionalities:

- Generates all visual representations, including subplots, of the best-response correspondences and equilibria.
- Produces a written description of all completely mixed equilibria in terms of the `IntersectionComponent` structure.
- Displays a 3D interactive plot of the equilibria, which can be rotated for different viewing angles.

The workflow of our code can be summarised as follows:

(i) Receives the payoff matrix and initialises an instance of the `PartialAlg` class.

(ii) Computes the K-coefficients of payoffs required for further calculations.

(iii) Implements Algorithm 1 to determine all pure and partially mixed equilibria and solve the roots of the quadratic equations (1.20), (1.21), and (1.22).

(iv) Creates an instance of the `GamePlot` class to compute all completely mixed equilibria, using the quadratic equation results and analysis of degenerate cases, and generates the 3D representation of the game.

(v) Constructs the best-response correspondence and identifies IOS curves and completely mixed equilibria.

(vi) Produces the 3D interactive plots and provides a written description of the equilibria.

Part II.

Empirical Game-Theoretic Analysis of a Pricing Game

Introduction to Part II

This part investigates the Empirical Game-Theoretic Analysis (EGTA) of a multi-round duopoly pricing game using model-free Reinforcement Learning (RL) methods. We apply the Policy-Space Response Oracles (PSRO) framework to approximate the infinite-strategy "hyper-game" that encompasses all possible pricing strategies available to the players in the pricing game. In this framework, single-agent reinforcement learning is used to compute best-response pricing strategies against previous Nash equilibrium strategies of the approximating game. Using this approximation, we study the equilibria to which the learning dynamics converge and the behaviour that emerges from RL-trained pricing strategies.

Central to our study is a multi-round pricing game of a duopoly with demand inertia, which belongs to the class of games introduced by Selten [62]. This pricing game is asymmetric, non-zero-sum, allows for varying degrees of cooperation among players, and is analytically intractable.

Selten analysed this class of games by introducing the concept now known as *subgame perfect equilibrium (SPE)*, which was cited in his Nobel Memorial Prize award [49]. The unique SPE of these games prescribes very competitive behaviour between the two price-setting firms. This equilibrium, explained in Section 4.3.1, although computed by Selten, is considered too complicated for general players to reach. He mentioned that, due to bounded rationality and other psychological factors, this SPE does not model actual human behaviour.

Another important behaviour in this game is that firms could cooperate by colluding and repeatedly setting high prices. Similar to a finitely repeated prisoner's dilemma, this is not a Nash equilibrium of the multi-period game because it *unravels* under backward induction. Nevertheless, such price collusion may occur between players who do not look too far into the future, where this unravelling begins. If this collusion occurs, it can yield higher returns for both firms compared to the SPE.

Reinhard Selten was a pioneering analyst of the strategic interaction of both fully rational players (game theory) and real human beings with *bounded rationality* (experimental economics) [50]. Selten's reference to different human behaviour in his work is presumably one motivation for the work of his PhD student, Claudia Keser, who experimentally studied his multi-period pricing game in 1990. She used the *strategy method*, where game theorists submitted strategies for this game in the form of flowcharts for both firms, determining the next price for each period. (The two firms were not symmetrical to avoid an easy focal point

for price collusion.) These strategies were then played against each other in a tournament. Afterwards, participants were informed about their performance and invited to submit a revised strategy for a second tournament. Both tournaments are described and analysed in Keser [37, 38].

My supervisor and co-author, Bernhard von Stengel, was a successful participant in Keser's two tournaments. His success was due to a detailed understanding of the game mechanics, particularly the trade-off between long-term and short-term profits, and—in particular, for the second tournament—his exploitation of more naive opponents. Due to von Stengel's familiarity with the game, we chose it for our experiments with machine learning techniques, not least to assess whether the learned strategies performed well.

We study the same pricing game explored in Keser's experiment, framing it as a machine learning problem to model the competitive dynamics of firms. Specifically, we represent the pricing game as a reinforcement learning environment, where the firms are learning agents that seek to optimise their policies to perform effectively against an opponent strategy. Since the primary objective of firms in this game is to maximise cumulative payoffs, the RL environment is designed to provide feedback in the form of payoffs corresponding to the agents' chosen actions (prices). This feedback is used to iteratively refine their strategies.

The aim of this project is, first, to leverage reinforcement learning algorithms to train strategies for playing a complex pricing game that presents significant challenges, even for human decision-makers; and second, to explore the equilibria in the space of strategies, analysing their behaviour in terms of collusion and competition.

The emergence of collusive behaviour among machine learning-trained models has been studied across various classes of pricing games. Beginning with repeated Cournot oligopoly games, Waltman and Kaymak [70] used computer simulations to show that players trained using Q-learning can learn to collude.

In a notable recent study, Calvano, Calzolari, Denicolo and Pastorello [9] observed and analysed collusion between pricing strategies learned by Q-learning algorithms in a repeated Bertrand oligopoly game. They interpreted the agents' behaviour as collusion with punishments for deviation, consistent with classical theoretical results on repeated games. Furthermore, they found that collusion was less likely in asymmetric pricing games, where agents struggled to coordinate on a socially optimal solution.

In more recent work on Q-learners, Bertrand, Duque, Calvano and Gidel [4] theoretically proved the conditions under which agents learn a specific cooperative strategy when playing the repeated Prisoner's Dilemma.

Askenazi-Golan, Cecchelli and Plumb [1] argue that collusion should be understood as an emergent behaviour of trained agents arising from learning dynamics. They investigate the convergence properties of various learning dynamics in general repeated games. Crucially, for some learning dynamics, they show that certain Nash equilibria have a basin of attraction, meaning that if the initial strategies lie within a sufficiently small neighbourhood of such an equilibrium, the learning process will converge to it.

Brown and MacKay [7], analysing competition in different pricing games, observed that high-frequency pricing algorithms can reduce competition and lead to higher profits for the firms in equilibrium. They suggested regulations to prohibit firms from conditioning prices on rivals' prices, or limiting the storage of rival firms' prices to avoid collusive behaviour.

In Miklós-Thal and Tucker [46], the authors studied pricing algorithms currently in use and argued that the risk of collusion may be overstated in recent works. They explained that algorithms with better demand forecasting capabilities may reduce collusive behaviour, as accurate demand predictions make sustaining collusive prices more difficult. Conversely, collusive behaviour is more likely in pricing algorithms that allow firms to peg their prices to rivals' prices. These findings suggest that the potential for collusive behaviour heavily depends on both the type of pricing algorithm and its implementation. In a recent paper by Douglas, Provost and Sundararajan [16], the repeated game of Prisoner's dilemma was analysed using bandit algorithms. The authors argued that a class of deterministic bandit algorithms can settle on collusive behaviour without conditioning on rivals' prices. This suggests that prohibiting algorithms from conditioning on rivals' prices is not sufficient to avoid collusion.

Our approach differs from the previously mentioned papers both in terms of the structure of the pricing game and the learning methodology employed.

In prior work, the pricing game studied is a repeated game, whereas ours is not. We consider a fixed-length, 25-round game, where the demand potential at the start of each round depends on the pricing history of previous rounds. This setup introduces unique complexities. Agents' actions depend not only on the opponent strategy they face but also on their current position within the game. For instance, in the final rounds, agents no longer need to compete over future demand potential, as there are no subsequent rounds. Punishment strategies also lose effectiveness in later stages. Furthermore, unlike many previous models, our action space is continuous, and agents can choose from an infinite set of prices. These differences significantly increase the learning complexity of the game.

The increased complexity necessitates a different learning approach from those used in the aforementioned studies. The common methodology in multi-agent reinforcement learning is known as *independent contemporaneous learning*, where all agents interact and learn simultaneously by updating their respective policies. In contrast, we adopt the *Policy-Space Response Oracles* (PSRO) [40] framework, which is rooted in *Empirical Game-Theoretic Analysis* (EGTA) [72].

In PSRO, agents are trained sequentially; only one agent learns at a time while its opponent remains fixed. This approach is motivated by two main factors: first, our aim to approximate a "hyper-game" that spans the space of all possible pricing strategies; and second, the need for stable training due to the complexity of our pricing game. Since the strategy space is infinite, direct mathematical analysis and explicit equilibrium computation have their limitations. PSRO enables us to study a reduced subgame, referred to as the *meta-game*, which evolves iteratively to approximate the full strategy space.

To approximate a Nash equilibrium of the hyper-game, we use the equilibrium of the meta-game to guide the generation of new strategies, following the standard EGTA methodology. The meta-game is represented as a bimatrix game, with each row corresponding to a strategy for the low-cost agent and each column to one for the high-cost agent. This game is initialised with one or more base strategies. At each iteration, we compute at least one Nash equilibrium of the current meta-game using an algorithmic implementation of the Harsanyi–Selten tracing procedure [26]. New agents are then trained against this equilibrium profile. If a new strategy achieves a higher payoff than the equilibrium payoff, it is added to the strategy pool, allowing the meta-game to evolve.

Despite adopting a different learning approach, we employed many ideas from previous studies. These works also inspired us to analyse not only the meta-game equilibria but also the emergence of collusion among trained agents, in addition to their competitive behaviour.

Unlike prior work that observes emergent collusion and proposes regulation to prevent it, our analysis takes a reverse-engineered approach. Early experiments did not reveal signs of collusion. Instead, we actively explored factors that might foster collusive behaviour. Our objective was to identify such factors so they can be deliberately avoided in future market designs.

In this research, we address the following key questions:

- Is there an alternative equilibrium in the specified pricing game that yields higher expected payoffs for both firms compared to the subgame-perfect equilibrium of the game?
- Which reinforcement learning algorithms can effectively train agents that optimally compete in the pricing game against a fixed opponent? Furthermore, how do these requirements evolve when agents face a mixed strategy of opponents, as arises in the PSRO framework?
- Do agents trained within the PSRO framework naturally learn to collude in a pricing game of this complexity?
- What factors lead to cooperation or competitiveness among agents trained using this framework?
- Does the equilibrium selection method employed in the PSRO framework significantly influence the final outcomes?
- How do agents behave when they converge to an equilibrium that is more cooperative?

In Chapter 4, we begin by explaining the framework, the pricing game, and the PSRO setting. We then provide an overview of the foundations of reinforcement learning algorithms, which form the basis for the algorithms we implement in Chapter 5, or utilise methods from the Stable-Baselines3 framework in Chapter 6.

In Chapter 5, we explain and implement basic policy gradient methods. We detail the technical aspects and explain the reasoning behind each decision made during the implementation. We define preliminary goals and highlight the limitations of these methods, which led to the development of the next algorithm. Finally, we demonstrate that these basic

models cannot adapt to playing against mixed strategies of opponents, as required in the PSRO setting.

In Chapter 6, we adopt more advanced reinforcement learning algorithms implemented in the Stable-Baselines3 framework, namely Soft Actor-Critic (SAC) and Proximal Policy Optimisation (PPO). These algorithms enable us to focus more on the meta-game as they can capture the complexity of PSRO settings. We explain the two algorithms, outline their differences, and conduct experiments using agents trained with both methods. We compare their performance in the meta-game in Sections 6.4.2 and 6.4.3. In our final experiments, we study the effects of different equilibrium selection methods and initial strategies in the meta-game in detail. We further analyse the equilibria found in these experiments, referred to as the *empirical mixed equilibrium*, studying the behaviour of strategies within this equilibrium in Section 6.4.4 and comparing it to the subgame-perfect equilibrium in Section 6.4.1. Finally, in Section 6.4.5, we present the results of replicator dynamics in the final meta-games.

The main contributions of this work include demonstrating that two key factors significantly influence the emergence of cooperative behaviour between agents trained in PSRO (Policy Space Response Oracles) settings: the choice of Nash equilibrium for training the next agents, and the specific initial strategies of the meta-game.

In our results, we show that selecting a Nash equilibrium that maximises social welfare strongly promotes collusive behaviour among new agents, provided that collusive strategies are introduced to the agents during the initial iteration of the meta-game, where new strategies are developed and compete against them. Our experiments indicate that these two factors must be combined, as neither factor independently leads to cooperation between agents.

Another new contribution of this work is the application of the Harsanyi-Selten tracing procedure for equilibrium computation. This method proves computationally efficient for the medium-sized meta-games that are created. We demonstrate that the equilibrium most frequently identified using this Bayesian approach, where the tracing procedure is started from random priors of agents' beliefs, does not lead agents trained in the PSRO setting toward cooperative behaviour.

Furthermore, we observe that when collusive behaviour emerges, the agent's payoffs of the mixed equilibrium approached in the limit are rather similar across all experiments. The strategies in these equilibria exhibit specific behaviours: a weakly cooperative behaviour for the low-cost player and a highly cooperative behaviour for the high-cost player. This empirically derived equilibrium yields higher returns for both agents in the pricing game compared to the subgame-perfect equilibrium, and is closer to the Pareto frontier.

In addition, we present a framework, available online for public use [35], for studying two-player pricing games, which can be extended to other pricing game scenarios in future research. Our framework is compatible with the Stable-Baselines3 reinforcement learning library, which supports a wide range of value-based and policy-based reinforcement learning algorithms. It includes multi-processing capabilities for efficient computation, database

creation for recording various aspects of the game (such as meta-game equilibria and the details of trained agents and their competitions), and comprehensive logging of every aspect of the game under study.

4. Foundations and Modelling

In this chapter, we begin by introducing the foundational concepts of single-agent reinforcement learning, followed by an overview of the core algorithms that form the basis of our approach. We then explain how these algorithms are integrated into a broader framework to approximate a more complex multi-agent game. Subsequently, we detail the structure and unique challenges of the pricing game examined in this study. Finally, we bring these components together to present our proposed learning framework, illustrating how both theoretical foundations and game-specific characteristics shape its design.

4.1. Learning Fundamentals

Machine learning techniques enable the development of agents that learn from experience and generalise to previously unseen situations. These agents can act effectively even in environments they have not directly encountered. Among these approaches, Reinforcement Learning (RL) allows agents to discover (semi-)optimal actions through trial and error, guided by feedback from repeated interactions with their environment. A key strength of RL is its ability to uncover underlying patterns and structures without requiring an explicit model of the environment. In the single-agent setting considered here, the objective is to train an agent to learn an approximate best-response strategy in an environment whose dynamics are influenced by the agent's own actions, for instance, when playing a multi-round game against a fixed opponent.

In this section, we introduce the foundational concepts of RL and the key algorithms employed in this work, following the notations and framework introduced by Sutton and Barto [64].

4.1.1. Single-Agent Reinforcement Learning

This section is adapted from [64, Chapter 1].

Reinforcement learning is a subclass of machine learning that relies on trial-and-error interactions with an environment. In the single-agent setting, each learning episode proceeds as follows: the agent observes the current state of the environment, selects an action based on its model parameters and the observed state, and then executes that action (we elaborate on these concepts shortly). The action causes the environment to transition to a new state,

and the agent receives a corresponding reward. This process continues until a terminal state is reached, marking the end of the episode. The agent then updates its model parameters based on the actions taken and rewards received during the episode.

Through a large number of interactions with the environment, the agent learns how to make decisions by trying different actions and using the rewards as feedback. These rewards show how effective the chosen action is with regard to the final goal. An important aspect of this process is that the action taken not only influences the immediate reward but also affects future states and rewards. Therefore, the agent should consider the long-term effects of its actions as well. By playing in the environment repeatedly and learning from each step, the agent enhances its strategy to achieve the final goal.

In reinforcement learning, the balance of two concepts plays a crucial role in the efficiency and performance of the model. Firstly, when the agent finds a series of actions that lead to a high overall reward, then these actions should be employed more to get closer to optimal behaviour. This concept is known as *exploitation*. Secondly, the agent cannot find a sequence of actions with high rewards unless it explores new actions in the action space. This concept is called *exploration*. For the agent to learn efficiently, both concepts should be balanced. If a model only explores without exploiting successful strategies, then it will never learn an optimal strategy since it does not learn from experiences. Moreover, if a model uses only exploitation, then the probability of finding the optimal strategy on the first try is very low. Also, with low exploration, the model cannot move beyond local optima to find global optimum points. The trade-off between exploration and exploitation has high importance in reinforcement learning and should be controlled by tuning the hyperparameters of learning methods. We discuss the hyperparameters for each of the implemented algorithms in the next chapters.

We next explain each component of reinforcement learning within the context of our setting, where agents play the pricing game. The details of the pricing game are presented in Section 4.3. In short, this is a 25-round duopoly pricing game in which two firms simultaneously set the prices of their products in each round. The total demand is then split in favour of the agent offering the lower price. Each agent receives a reward based on its sales in that round, and the game proceeds to the next round.

Now, if the opponent is fixed, each agent can treat the opponent and their actions as part of the dynamics of a single-player game. Within this framework, the components of reinforcement learning can be described as follows:

- **Agent**: An agent, or player, is the decision-maker that interacts with the environment by observing its state and taking actions that transition the environment to a new state. Through repeated interactions and feedback, the agent learns an optimal policy to maximise its cumulative reward over each episode, i.e., the sum of rewards from all rounds of the pricing game. In our setting, each player (firm) participating in the pricing game is an agent.
- **Environment**: The environment is the system with which the agents interact. It defines the game dynamics, including how rewards are computed and how states

transition in response to the agents' actions. In our experiments, the pricing game itself represents the environment. Based on the prices selected by the players, the environment updates the game state, computes payoffs, and moves to the next round.

- **State**: The state (or observation) provides all the information an agent needs to make a decision. It includes parameters of the game that are relevant for decision-making. The design of the state representation is crucial: if too simple, the agent may not distinguish between situations that require different actions, leading to poor performance; if too complex, learning becomes slower and more difficult due to the high dimensionality. We describe the specific state representations used in each model later.
- **Action**: Agents interact with the environment by taking actions, which move the game to a new state. The action space may be continuous or discrete; in this project, we consider both types and specify the choice in each model. In our setting, the action corresponds to the price each agent sets in the pricing game, which in turn affects demand and determines the reward received in that round.
- **Reward**: After selecting an action, the agent receives a reward from the environment. This scalar value indicates the agent's performance in that particular round. In our experiment, the reward corresponds to the player's payoff at that stage of the pricing game.
- **Policy**: A policy, denoted by π, is the reinforcement learning equivalent of a **strategy** in game theory. It defines the agent's behaviour by mapping states to actions. The agent's objective is to learn an optimal policy that maximises expected cumulative rewards.

The mathematical framework to model these agents' interactions with the environment is the *Markov Decision Process (MDP)*, which we explain in the next section.

4.1.2. Markov Decision Process

This section is adapted from [64, Chapter 3].

A Markov Decision Process (MDP) provides the mathematical framework for representing decision-making scenarios where an agent interacts with an environment across discrete time steps. At each time step, the agent observes the current state, selects an action, and the system transitions to a new state. This process continues until a terminal state is reached. This model allows us to systematically compute the optimal policy required in reinforcement learning (RL).

Definition 4.1. A Markov Decision Process (MDP) is characterised by the following components:

- S - the set of feasible states describing the system's situation. S_0 denotes the initial state, and S_T is the set of terminal states at the last time step T.
- A - the set of possible actions available at each state.

- R - the set of all possible rewards the agent can receive when the system transitions to a new state.
- $P : S \times R \times A \times S \rightarrow [0, 1]$ - is the *dynamics function* of the system. $P(s', r \mid s, a)$ is the (possibly deterministic) probability of the system transitioning to state s' and receiving reward r when action a is taken in state s. Building on this definition, the *transition function* $P : S \times A \times S \rightarrow [0, 1]$ can be defined as $P(s' \mid s, a) = \sum_{r \in R} P(s', r \mid s, a)$. The function $P(s' \mid s, a)$ represents the probability of transitioning to state s' when action a is taken in state s. We use the same notation for both of these functions, as both define the probabilities of transitions. They can be distinguished by the number of arguments.

An important property that MDPs have is known as the *Markov property*, which states that the next state the system transitions to depends only on the current state and the chosen action. It is independent of the sequence of states and actions in previous time steps. If S_t, a_t represent the system's state and chosen action at time t, according to the Markov property, we have:

$$P(s_{t+1} \mid s_t, a_t) = P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) \,.$$

This means the state should include all information that affects the future. This property simplifies decision-making and is essential for many reinforcement learning algorithms because it allows the use of the current state to make decisions, instead of the trajectory of all previous events.

Another important point to highlight is that the model we implement in this project corresponds to a Partially Observable Markov Decision Process (POMDP). In the next chapter, we will see that while each agent receives observations representing the state of the pricing game, the transition probabilities remain uncertain for the agent. This uncertainty arises because the transitions depend not only on the agent's own actions but also on the opponent's actions, which are not observable at the time the agent acts. Instead, the opponent's current behaviour may sometimes be inferred through the observation of past behaviour.

4.1.3. Value Functions

This section is adapted from [64, Chapters 3 and 4].

The goal of RL is to maximise the cumulative rewards. This is only possible if we have an estimation of future rewards following each path. In reinforcement learning, this estimation is done by approximating value functions. The value functions can be defined for each state or state-action pair.

The state-value $V_\pi(s)$ of policy π at state s is defined as the expected return from state s onwards :

$$V_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0 = s]$$

where R_t is the reward at stage t of the environment and γ is the discount factor which determines the current valuation of future rewards. Similarly, the action-value $Q_\pi(s,a)$ is defined as the expected future return starting at state s and choosing action a.

$$Q_\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0 = s, a_0 = a] \,.$$

These value functions can be written in terms of each other:

$$V_\pi(s) = \sum_{a \in A_t} \pi(a|s) Q_\pi(s,a) \,,$$
$$Q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V_\pi(s') \,.$$

In finite MDPs, the optimal policy π^* can be defined as the policy which has equal or higher state value than other policies, in all states:

$$V_{\pi^*}(s) \geq V_\pi(s), \quad \forall s \in S, \forall \pi \in \Pi$$

where π^* is not necessarily unique. The optimal state-value and action-value functions can be defined similarly:

$$V_*(s) = \max_\pi V_\pi(s), \qquad Q_*(s,a) = \max_\pi Q_\pi(s,a) \,.$$

Bellman optimality equations state that under an optimal policy, the value of a state is equal to the expected reward that follows the best action. Similarly, for action-values, the value of a state-action pair under an optimal policy is equal to the expected return of choosing the action that maximises the action-value function:

$$V_*(s) = \max_a \sum_{(s',r)} p(s',r|s,a)[r + \gamma V_*(s')] \,,$$
$$Q_*(s,a) = \sum_{(s',r)} p(s',r|s,a)[r + \gamma \max_{a'} Q_*(s',a')] \,.$$

Since these value functions already take the value of future rewards into account, it suffices to select the action that maximises these values. In finite MDPs, the system of equations for V_*, one equation for each state, has a unique solution. Once V_* is known, Q_* can be computed. Then, any policy that selects only the actions (not necessarily unique) maximising the action-values is an optimal policy.

Although it is ideal to compute the optimal policy by solving the Bellman equations directly, in most real-life problems, and in our experiment, this is not feasible due to the large number of equations involved, caused by a large state space.

In settings where the dynamics of the environment (i.e., transition probabilities and rewards) are completely known, the *policy iteration method* can be used to derive the

optimal policy. Policy iteration consists of two steps that are repeated until the optimal policy is found. These two steps are known as *policy evaluation* and *policy improvement*.

In the policy evaluation step, the value function for the current policy is calculated for all states. This involves determining the expected return (sum of discounted future rewards) of following the current policy from each state, using the Bellman equations:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(.|s),\ s'\sim P(.|s,a)}\left[r + \gamma\, V_\pi(s')\right].$$

In the policy improvement step, the current policy is updated by acting greedily with respect to the evaluated value function. This means that for each state, the policy is updated to select actions that maximise the value function, resulting in a new policy that is guaranteed to be at least as good as, if not better than, the previous policy:

$$\pi'(s) = \arg\max_a \mathbb{E}_{s'\sim P(\cdot|s,a)}\left[r + \gamma\, V_\pi(s')\right].$$

Policy iteration, which involves repeating these two steps, continues until the new policy is the same as the old policy. This indicates that the optimal policy has been reached.

However, in many cases such as our experiment, the dynamics of the game are not completely known, and policy iteration cannot be applied since the transition probabilities depend on the opponent's actions. In these cases, *Monte Carlo methods* are often used to estimate the policies.

4.1.4. Q-Learning

This section is adapted from [64, p. 131-133].

We did not use Q-learning in our early experiments. However, in the final experiments, we employ SAC (Soft Actor-Critic), which can be viewed as a more advanced version of Q-learning for continuous action spaces. Therefore, we first explain the basics of Q-learning here.

Q-learning [71] is one of the most successful value-based reinforcement learning algorithms, especially for discrete action spaces. In Q-learning, Q-values are maintained as pairs $Q(s,a)$ for state-action pairs (s,a), similar to $Q_\pi(s,a)$ in Section 4.1.3 above. They are iteratively updated to approximate the optimal action-values using the Bellman equation update rule. Once the Q-values converge, the optimal policy can be inferred by selecting the action that maximises the Q-value at each state.

An important property of Q-learning is that it is an off-policy algorithm. This means that the actions used to collect new observations (executed under the *behavioural policy*) may differ from the actions used to update the Q-values. The Q-values are updated based on a *target policy*, which is usually the greedy policy that selects the best actions (i.e., actions that maximise the Q-values).

This is how the Q-values are updated in Q-learning. Iterations take place at time steps $t = 0, 1, \ldots, T$, where the current state is s_t. At that state s_t, an action a_t is chosen according to the behavioural policy. One example of this policy is ε-greedy, which chooses the action with the highest value with probability $1 - \varepsilon$ and chooses a random action with probability ε. Then the process transitions to the next state s_{t+1} and receives the reward R_t. This reward is used to update the Q-value of the previous state s_t according to the following assignment, based on the target policy, which is shown here as the greedy policy:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big[R_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \Big], \tag{4.1}$$

where α is the learning rate, γ is the discount factor, and $\max_{a'} Q(s_{t+1}, a')$ represents the maximum Q-value of the next state s_{t+1} over all possible actions a'. This procedure continues for the next states.

In equation (4.1), the first two terms inside the brackets represent the new approximated Q-value based on the observed reward and the expected future value, while the last term is the previous Q-value. The Q-values are updated in the direction of the new observation, with α as the learning rate.

The off-policy property of Q-learning arises from the new Q-value approximation, $R_t + \gamma \max_{a'} Q(s_{t+1}, a')$. Here, the Q-value of the next state is not computed using the policy that generated the observation (behavioural policy). Instead, the update uses the greedy policy, which selects the action with the maximum Q-value.

The common choice for the behavioural policy is the ε-greedy policy, which exploits the action with the highest Q-value with probability $1 - \varepsilon$, and with probability ε, explores other actions uniformly at random.

Q-learning improves the Q-values based on the Bellman equation and is known for its strong convergence properties in discrete cases.

4.1.5. Policy Gradient Algorithms

This section is adapted from [64, Chapter 13].

Policy gradient algorithms, as the name suggests, are a *policy-based* class of RL algorithms. Unlike value-based methods that approximate state-action values and derive a policy from them, policy-based algorithms directly optimise the policy by moving in the direction of the gradient of the discounted cumulative reward. The goal is to find the policy that maximises the cumulative reward across all stages. In the episodic case, like our pricing game with an episode length T and assuming the policy π is parametrised by a parameter θ, the objective is to maximise J_θ:

$$J(\theta) = V_{\pi_\theta}(s_0) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t R_t \right],$$

where τ is a trajectory of states and actions.

The policy gradient theorem [64, p. 325] states that:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t\right], \tag{4.2}$$

where $G_t = \sum_{k=t}^{T} \gamma^{k-t} R_k$.

This theorem allows us to use different gradient-based optimisation methods, such as stochastic gradient descent, to move in the direction of the gradient in order to maximise $J(\theta)$. This is the basis for the algorithm we implemented in our early experiments in Section 5.1, which we call REINFORCE.

We explain more about our implementation of policy gradient methods in the next chapter, along with the modifications we make to the algorithm.

4.2. Policy-Space Response Oracles

In this project, we adopt the *Policy-Space Response Oracles* (PSRO) framework for learning, which is rooted in *Empirical Game-Theoretic Analysis* (EGTA).

Empirical Game-Theoretic Analysis (EGTA) [72] is a method for studying strategic interactions in complex games by approximating the full game through simulations. This approach is particularly useful when traditional game-theoretic analysis is infeasible due to factors such as the size or complexity of the game. The PSRO framework, introduced by Lanctot, Zambaldi, Gruslys, Lazaridou, Tuyls, Perolat, Silver and Graepel [40], combines ideas from empirical game theory and the Double Oracle method with multi-agent learning.

The Double Oracle algorithm, introduced by McMahan, Gordon and Blum [45], is a framework of EGTA that models planning in multi-agent Markov Decision Processes (MDPs) as a two-player zero-sum matrix game. It assumes that players are equipped with oracles capable of computing exact best responses to their opponents' strategies. The algorithm begins with an initial set of random strategies, which forms the first subgame used to approximate the full game. We refer to this expanding subgame as the *meta-game*, which is iteratively updated to better approximate the full game, referred to as the *hyper-game*.

At each iteration, the equilibrium of the current meta-game is computed, and both players determine their best responses to the opponent's equilibrium strategy using their oracles. These best responses are then added to the meta-game, and the process repeats. In finite games, this procedure eventually converges to the equilibrium of the hyper-game, although in the worst case, all strategies in the hyper-game may need to be added. Nevertheless, empirical results show that the Double Oracle algorithm often converges more quickly than solving the full game's linear programming formulation directly. In summary, the equilibria of the meta-games iteratively approximate the equilibrium of the hyper-game by expanding the strategy space through best responses.

PSRO [40] generalises the Double Oracle algorithm to multi-agent learning and non-zero-sum games. Unlike Double Oracle, which relies solely on Nash equilibria, PSRO

supports a variety of *meta-strategy solvers*, such as replicator dynamics or correlated equilibria [5]). In PSRO, the payoff from the meta-strategy solver's solution is used as a threshold: new best responses must exceed this value to be added to the game. This mechanism filters out the best responses that perform worse than previously added strategies. Moreover, in PSRO, learning algorithms act as oracles that compute (approximate) best responses to the equilibrium strategies in the meta-game. Unlike the Double Oracle method, where strategies are pure, PSRO allows for stochastic strategies to be trained and added to the meta-game.

In our project, we use the Nash equilibrium as the meta-strategy solver, similar to the original Double Oracle method. This means the Nash equilibrium of the current meta-game guides the training of new best responses, and its expected payoff serves as the threshold for incorporating them into the meta-game.

In finite games, PSRO typically converges to a Nash equilibrium, analogous to the convergence of the Double Oracle method in zero-sum settings.

Algorithm 2 presents the pseudocode for the instantiation of the PSRO framework in our pricing game setting. We refer to the two players as the *low-cost* and *high-cost* agents. The aim of this framework is to approximate the hyper-game and its equilibria through the expanding meta-game G. The hyper-game represents the game in which all possible strategies for playing the pricing game, employed by the low-cost and high-cost players, compete against each other.

Algorithm 2 Policy Response Oracles algorithm

$G = (\Pi_1, \Pi_2) \leftarrow$ Initialise the subgame as explained in Section 6.3.4
for each round i: **do**
    $NEs \leftarrow$ select the equilibria of G, as explained in Section 6.3.3
    for $(S_1, S_2) \in NEs$ with payoffs (x, y): **do**
        $L \leftarrow$ train low-cost agents against S_2
        $H \leftarrow$ train high-cost agents against S_1
        for $\pi \in L$ with payoff $p > x$: **do**
            $\Pi_1 \leftarrow \Pi_1 \cup \{\pi\}$
        end for
        for $\pi \in H$ with payoff $p > y$: **do**
            $\Pi_2 \leftarrow \Pi_2 \cup \{\pi\}$
        end for
    end for
end for

4.3. The Pricing Game

In this section, we explain the *pricing game*, which forms the core of the competitive interaction in our research. This game belongs to the class of games studied by Selten [62]. We adopt the same parameter settings used by Keser [37] in her experimental study, allowing us to draw insights from and compare our results with hers.

The game models a duopoly between two firms that produce the same product but at different production costs. The low-cost firm has a unit production cost of 57 (denoted by c_1), while the high-cost firm incurs a unit cost of 71 (denoted by c_2). The total market *demand potential* is fixed at 400 units, and both players begin the game with an equal initial demand potential of $D_1^0 = D_2^0 = 200$.

The only deviation from Keser's setup is the exclusion of a small discount factor, which she used to increase the accumulated profit by one percent in each round. We omit this component, as it does not appear to significantly affect the observed behaviour of the players.

In each round of the game, both players simultaneously set their product prices (p_i) in the market. The payoff (reward) to player i at each round is determined by

$$R_i(p_i, D_i) = (D_i - p_i)(p_i - c_i) = -p_i^2 + (c_i + D_i)p_i - c_i D_i. \tag{4.3}$$

The prices set by the firms have a second effect aside from the immediate payoffs in that round. This is a pricing game with *demand inertia*, meaning that the prices set by the firms affect their demand potential D_i with some delay, specifically, in the next stage of the game. The total demand potential remains constant in all rounds. However, the demand potential of each firm changes in favour of the cheaper firm. After each round, the demand potential for the next round is changed proportionally to the price difference between the two firms.

Specifically, the player who sets a higher price loses half of the price difference in terms of demand potential, which is then gained by the player who sets a lower price:

$$D_1^{t+1} = D_1^t + \frac{1}{2}(p_2^t - p_1^t),$$
$$D_2^{t+1} = D_2^t + \frac{1}{2}(p_1^t - p_2^t).$$

As is evident, the sum of the players' demand potential remains constant in all rounds.

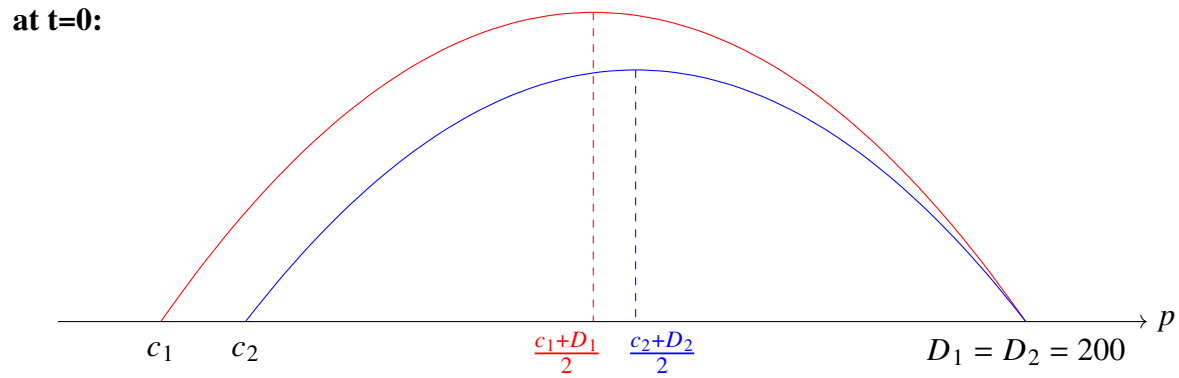


Figure 4.1.: Low-cost (red) and high-cost (blue) player payoff plotted as a function of price in the first stage of the pricing game.

In the payoff function of each round, the term $(D_i - p_i)$ can be considered as the quantity sold, and the second term $(p_i - c_i)$ is the profit from each unit. Each player's

objective is to maximise their cumulative payoff over the entire 25-round game, which is the sum of the payoffs earned in each round.

If a player attempts to maximise their payoff in the current round by setting a high price, they risk losing demand potential, which, in turn, could negatively impact their total payoff in future rounds. Hence, each firm aims to achieve a higher immediate payoff while also securing a larger proportion of the total demand potential to increase their payoff in the next round. From the demand perspective, customers prefer the firm that offers a lower price, as the products are of the same quality. However, unlike the setting in [11], for example, that preference manifests itself only in the next round due to the assumed demand "inertia".

For the one-stage game, the payoff function is quadratic and concave in terms of price, as shown in Figure 4.1. By taking the derivative of the current reward R_i with respect to the firm's price p_i, the optimal price p_i^* for that round is $\frac{(D_i+c_i)}{2}$, as the solution to

$$\frac{\partial R_i}{\partial p_i} = -2p_i + c_i + D_i = 0 \quad \Rightarrow \quad p_i^* = \frac{(D_i + c_i)}{2}. \tag{4.4}$$

We refer to this price p_i^* as the *myopic price* of firm i since it maximises the payoff in the current round without considering the effect on future demand potential. In the last round of the game, since there is no future demand potential to be affected, the myopic price is the optimal price to play. We refer to this principle, where playing myopically in the last round is optimal, as the "end-effect" of the game. Later in our study, we will attempt to train agents to learn and adapt to this end-effect.

Figure 4.1 shows that playing myopically in the first round leads to changes in demand potential for the next round since the optimal prices for the players are different. This raises the question of whether it is possible for both players to play myopically without worrying about demand changes, meaning the myopic prices of both players should coincide. In that case,

$$\frac{c_1 + D_1}{2} = \frac{c_2 + D_2}{2} \rightarrow D_1 - D_2 = c_2 - c_1 = 14 \overset{D_1+D_2=200}{\longrightarrow} D_1 = 207, D_2 = 193\,.$$

The equation above shows that if the players' demand potential is divided as $(207, 193)$, then the myopic prices for both players are the same, and they can achieve their maximum possible payoff for the round without changing future demand.

Another important point to consider is that this pricing game is not a repeated game, as the demand potential changes in each round, meaning the state of the game does not remain the same. Rather, it is an extensive-form game where, at each time step, a simultaneous game with continuous actions between two players is played.

In Figure 4.3, we present a 3D plot illustrating the payoff in a single-round pricing game as a function of the demand potential of Player 1 and the price of each player. This visualisation highlights the conflicting effect of the demand potential on the two players. At the start of each stage, the demand potential for that stage is known, defining the payoff parabola with respect to the price parameter as previously discussed. However, the demand potential changes in subsequent stages.

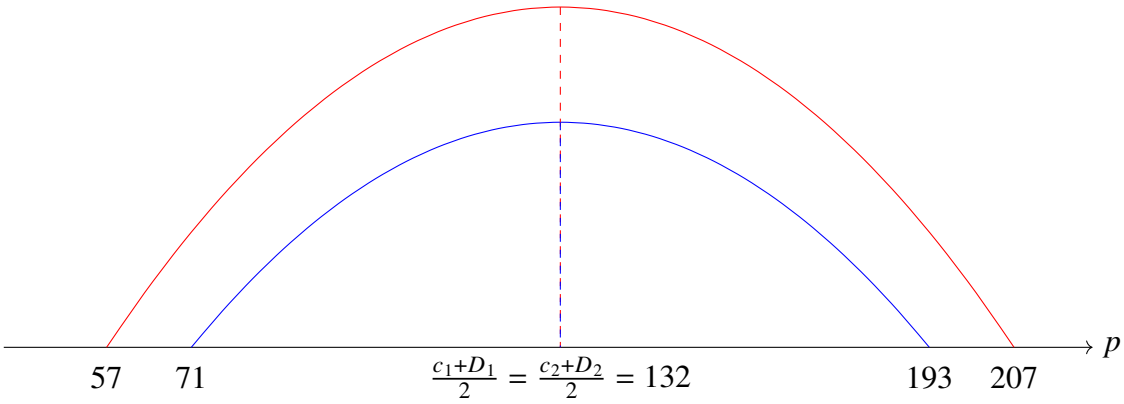


Figure 4.2.: The price at which low-cost and high-cost myopic prices match.

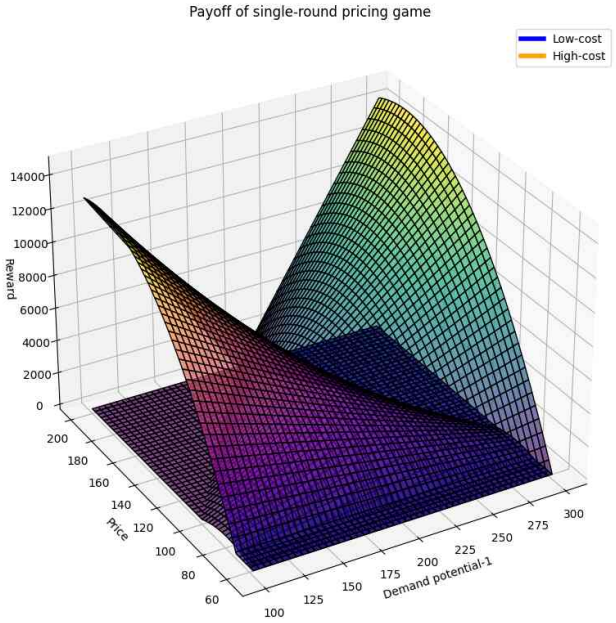


Figure 4.3.: 3D plot of the payoff in the single-round pricing game for low-cost and high-cost players, in terms of the price chosen by the respective player, and the demand potential of the low-cost player, which if given by D_1, determines D_2 as $400 - D_1$.

If the portion of total demand for Player 1 increases, the next stage begins on a parabola whose right endpoint is shifted right along the demand-potential axis. This shift alters the height of the payoff parabola, increasing the reward for the low-cost player while decreasing it for the high-cost player. The low-cost player has an advantage in this game, being able to push prices lower than the high-cost player. However, while this strategy increases demand potential for the next round, it sacrifices some profit in the current stage.

To analyse this trade-off, we assume $D_1 = 207$ and $p_1 = 132$, representing the demand that can be achieved through myopic play:

$$R_1(132, 207) = (207 - 132)(132 - 57) = 75^2.$$

If the low-cost player decreases the price by x units to gain more demand potential, they lose x^2 units of payoff in the current round:

$$R_1(132 - x, 207) = (207 - 132 + x)(132 - x - 57) = (75 + x)(75 - x) = 75^2 - x^2.$$

In the next round, if the high-cost player plays the myopic price, the added demand potential will yield an additional payoff of $\frac{x}{2}$(new price $-$ 57). Assuming the price returns to 132, the added reward is $37.5x$, which is higher than the loss from the previous round when $x < 37.5$, which is typically true. This shows that the strategy profile where both players play myopically is not an equilibrium, as deviation is profitable for them.

In infinitely repeated games, such deviations are generally not attractive due to punishment strategies employed by the other agent, which can result in several rounds of reduced future rewards. However, in our setting, the game has a fixed length, which reduces the effect of punishment strategies, especially in the final stages of the game.

4.3.1. Subgame Perfect Equilibrium

The concept of Subgame Perfect Equilibrium (SPE) was originally introduced as a solution concept for extensive-form games, in the very paper by Selten [62] that we study. SPE, an important concept in game theory, is a Nash equilibrium that is an equilibrium when restricted to any subgame of the extensive-form game. The set of SPE is a subset of the set of Nash equilibria.

Furthermore, in the same paper, Selten computed the SPE for a pricing game with general parameters (costs, demand potential, number of rounds, and even more than two players in the oligopoly version of the game). As the pricing game we consider in this project is a special case of these games, following Selten's formulation, we can compute the Subgame Perfect Equilibrium of the 25-round duopoly pricing game. This concept has several properties:

- The SPE is an equilibrium in each subgame of the game. Since this equilibrium is computed using backward induction, it remains an equilibrium from any point in the game until the end.
- In the backward induction process to compute the SPE, it is assumed at each step that the opponent also plays the SPE strategy. Therefore, the SPE strategy is a best response and is optimal against the opponent's SPE strategy.
- SPE is highly competitive. The player anticipates the opponent's future actions to be highly competitive and reacts competitively in turn. The optimality of actions at each step guarantees that there is no deviation at any point in time. However, this equilibrium is not necessarily the one with the highest payoff for all players.

In the SPE, the prices to be played by each firm at each stage can be computed as follows, with suitable parameters a^t, b^t and k^t (see [37], page 8):

$$p_i^t = a^t(D_i^t - D^t) + b^t + k^t(c_i - c) \quad \text{for } i \in \{1, 2\}, t \in \{1, 2, \ldots, 25\}$$

In this formulation, D_i^t represents the demand potential for player i at stage t and D^t is the average demand potential of all players in stage t; here $D^t = 200$ throughout. Similarly, c_i denotes the unit production cost of player i, and the average production cost is $c = \frac{c_1+c_2}{2} = 64$. The remaining parameters can be calculated recursively as follows. In our study, we set the discount factor γ as $\gamma = 1$.

$$\begin{aligned} a^t &= \tfrac{1-Y^{t+1}}{2-Y^{t+1}}, & b^t &= 132 - 0.25\gamma B^{t+1}, \\ k^t &= \tfrac{1-0.5\gamma K^{t+1}}{2-Y^{t+1}}, & A^t &= 0.25 + z^t A^{t+1}(1-a^t), \\ B^t &= 68 + z^t B^{t+1}, & K^t &= -0.5 + z^t(K^{t+1} - 2A^{t+1}k^t), \\ z^t &= \gamma(0.75 - 0.5a^t), & Y^t &= 0.25\gamma + z^t(1-a^t)Y^{t+1}. \end{aligned}$$

with initial values defined as below.

$$\begin{aligned} a^{25} &= 0.5, & b^{25} &= 132, & k^{25} &= 0.5, \\ A^{25} &= 0.25, & B^{25} &= 68, & K^{25} &= -0.5, \\ Y^{25} &= 0.25\gamma. \end{aligned}$$

Following the earlier formulation, we have plotted the prices and demand potential of both players in Figure 4.4. As depicted in the graph, prices tend to increase in the final stages. In the last stage, players adopt a myopic strategy, which maximises their stage reward since there is no next stage to consider for future demand. We can calculate the *myopic price*, which represents the best response of the players in the last stage, as defined in equation 4.4.

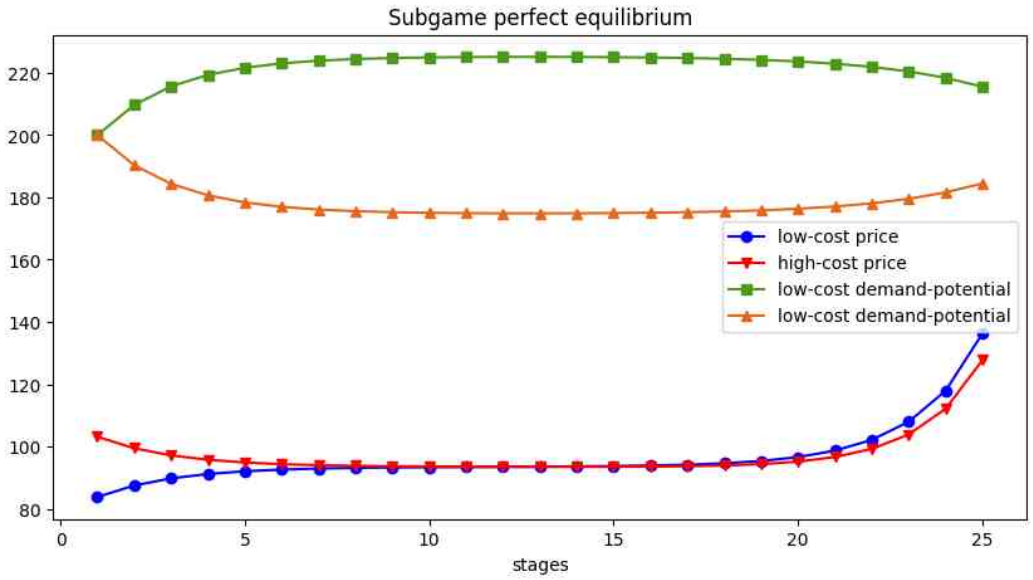


Figure 4.4.: Price and demand potential of players over 25 stages of the game in the subgame perfect equilibrium.

Playing the SPE strategy by both players over 25 rounds results in a total return of 120,810 for the low-cost player and 52,810 for the high-cost player. In the subsequent experiments, we divide the returns by 1,000 for easier interpretation and use the rounded return (121, 53) from the SPE as a benchmark to compare the effectiveness of collaborative strategies.

4.3.2. Initial Deterministic Strategies

In this section, we introduce a set of pricing strategies that serve two main purposes: they assist in the learning process and provide benchmarks for evaluating the performance of other strategies in terms of cooperation and payoff. We define several deterministic strategies that vary in their degree of cooperation. These strategies offer insights into the structure of the game and are also used to train our models, enabling agents to better understand the game dynamics and environment.

Myopic

When playing the myopic strategy, the agent maximises the stage's payoff by playing the myopic price. This strategy focuses on short-term payoffs and does not aim to gain higher demand potential for a larger overall payoff at the end. As defined in equation (4.4), player i (1 for the low-cost agent, 2 for the high-cost agent) at stage t plays:

$$\text{myopic price} : p_i^* = \frac{D_i^t + c_i}{2}$$

where D_i^t is player i's demand potential at stage t.

Constant-x

Another simple strategy is to play a constant price x in all stages. This strategy does not cooperate or compete, but it is straightforward for opponents to exploit.

Guess-x

The Guess strategy, suggested by Bernhard von Stengel in the experiment conducted by Keser in 1992 ([37]), was successful during the first phase of the experiment but less so in evolutionary settings. We hypothesised that its lower success in the evolutionary phase resulted from the lower payoff opponents received when playing against this strategy. Guess is a sophisticated strategy that anticipates the opponent's price and allows some margin for cooperation.

Algorithm 3 illustrates how Guess-x works. In the basic Guess strategy *myopic_range* is set to 7 and *max_price* is set to 125 for the low-cost agent and 130 for the high-cost agent.

The strategy plays price x in the first round and the myopic price in the last round. In other rounds, it considers a demand potential goal. If it has reached this goal, it cooperates by playing the last price within a *myopic_range* distance from the myopic price. If the price is further from the myopic price than the range, it moves the price toward the myopic price. It also never plays higher than the myopic price because it is not rational. However,

Algorithm 3 Guess-x Strategy for Player j

Input: initial price x
Parameters: $\texttt{myopic\_range} \leftarrow 7$, $\texttt{max\_price} \leftarrow [125, 130]$
if first stage **then**
    Play price x
else if last stage **then**
    Play $\texttt{myopic\_price}$
else
    $\texttt{demand\_goal}_i \leftarrow \frac{\texttt{total\_demand}}{2} + c_i - c_{-i}$ for $i = 1, 2$
    $\texttt{price\_goal} \leftarrow \frac{\texttt{total\_demand}+c_1+c_2}{4}$
    $\texttt{sale\_goal}_i \leftarrow \texttt{demand\_goal}_i - \texttt{price\_goal}$ for $i = 1, 2$ ▷ Cooperate
    if $\texttt{current\_demand}_j \geq \texttt{demand\_goal}_j$ **then**
        if $\texttt{last\_price} \geq \texttt{myopic\_price}$ **then**
            Play $\texttt{myopic\_price}$
        else if $\texttt{last\_price} \geq \texttt{myopic\_price} - \texttt{myopic\_range}$ **then**
            Play $\texttt{last\_price}$
        else
            Play $0.6 \times \texttt{last\_price} + 0.4 \times \texttt{myopic\_price}$
        end if
    else ▷ Compete
        $\texttt{new\_sale\_guess} \leftarrow 0.5 \times \texttt{sale\_goal}_j + 0.5 \times (\texttt{current\_demand}_{-j} - c_{-j})$
        $\texttt{opponent\_price\_guess} \leftarrow \texttt{current\_demand}_{-j} - \texttt{new\_sale\_guess}$
        $\texttt{price}^* \leftarrow \texttt{opponent\_price\_guess} - 2 \times (\texttt{demand\_goal}_j - \texttt{current\_demand}_j)$
        Play $\min(\texttt{price}^*, \texttt{max\_price}_j)$
    end if
end if

if the goal demand potential has not been reached, it fights for the demand potential by estimating the opponent's sales and price, then lowering its price below the opponent's to attract demand potential.

The Guess-132 strategy, when playing against itself, reaches the return (142, 79) for the low-cost and high-cost player, which we used to compare to the next variations of this strategy.

Guess2-x

Guess2 modifies the Guess strategy with *myopic_range* set to 15 and *max_price* increased to 135 for the low-cost agent and 140 for the high-cost agent.

Compared to the basic Guess strategy, Guess2 allows getting further from the myopic price when the goal demand potential is reached, which makes the strategy more competitive compared to the basic Guess because the myopic strategy is the most cooperative. However, it also sets a higher upper bound on the price (*max_price*) when the demand is not reached, which makes the strategy more cooperative because the higher *max_price* allows the agent to play a higher price if it is beneficial.

The Guess2-132 strategy, when playing against itself, reaches the returns (137, 75) for the low-cost and high-cost players, which is a lower return for both players compared to the strategy pair (Guess-132, Guess-132), showing more competitiveness.

Guess3-x

Guess3 is a more cooperative variant of Guess with *myopic_range* set to 3 and *max_price* set to 120 for the low-cost agent and 125 for the high-cost agent.

This strategy is more cooperative since it does not allow getting far from the myopic price when the goal demand potential is reached. However, when it has not reached the target demand yet, it is more competitive than the basic Guess strategy by setting a lower upper bound on the prices, trying to compete for the demand potential more aggressively.

The Guess3-132 strategy, when playing against itself, reaches the returns (143, 81) for the low-cost and high-cost players, which is a higher return for both players compared to the strategy pair (Guess-132, Guess-132), showing more cooperation.

SPE

This strategy plays according to the Subgame Perfect Equilibrium of the game from the current stage to the end. It assumes the same behaviour for the opponent as well, maximising the payoff at each stage. This strategy can be computed using backward induction from the last stage back to the current stage. We apply the formulation by Selten to compute the prices at each stage, as explained in Section 4.3.1.

This strategy is highly competitive since it maximises its own payoff by considering the same strategy for the opponent. It does not consider any chance for cooperation and is an optimal strategy against itself, as it is defined in this way.

Imitation-*x*

The Imitation strategy plays price x in the first stage and the myopic price in the last stage. In the intermediate stages, it mimics the opponent's previous price. This one-round penalty strategy encourages opponents to avoid reducing prices, as any decrease will be mirrored in the next stage.

We expected the opponents to understand the pattern when playing against this strategy and to be able to exploit it. (Indeed, the constant and imitation strategies quickly became suboptimal against trained strategies, which shows that they were exploited by them, in the experiment 6.4.)

4.4. The Framework: Pricing Game, Reinforcement Learning and PSRO

In this section, we explain how the concepts introduced in previous sections connect together in our framework.

The main objective of our research is to study the equilibria and equilibrium strategies in the space of strategies of playing the pricing game. The game that includes all possible pricing strategies is referred to as the hyper-game. Since the number of pricing strategies is infinite, our mathematical analysis can only provide the subgame perfect equilibrium (SPE). This equilibrium tends to be highly competitive. Therefore, we aim to investigate whether

the hyper-game has other equilibria that offer potentially higher payoffs for both players. To do this, we adopt the PSRO framework, which approximates the hyper-game and its equilibria by iteratively expanding and analysing a meta-game.

As demonstrated in Algorithm 2, the meta-game is a bimatrix game where the pricing strategies of low-cost agents represent the row-player strategies, and the strategies of high-cost agents represent the column-player strategies. We initialise this game with one or more strategies for each player. In the next step, we compute the Nash equilibrium of this bimatrix game.

Following this, new best-response pricing strategies are trained against the Nash equilibrium strategies of the meta-game. We use different reinforcement learning algorithms to train these best-response strategies. Specifically, new low-cost agents are trained against the fixed (not necessarily pure) high-cost equilibrium strategy, and vice versa.

After training, we evaluate whether these new agents are successful enough to be added to the meta-game. A new strategy is considered successful if its payoff in the pricing game exceeds the expected payoff of the current equilibrium. This iterative process continues: new agents are trained against the updated equilibrium, and the meta-game is extended with those that meet the success criterion.

To compute the (approximate) best-response strategies, we model our pricing game as a multi-agent reinforcement learning game. Each firm (player) is represented by a neural network and is referred to as an *agent*. The pricing game defines the environment where these agents, low-cost and high-cost, interact and compete. At each stage (round) of the game, both agents simultaneously select their actions, which specify the price they play in the pricing game. These actions determine the state of the environment, which in turn determines the rewards for each agent.

For effective learning through trial and error, the agents interact over hundreds of thousands of episodes, each consisting of 25 stages. They are trained based on the feedback (rewards) they receive. At each stage, the agents' policies (strategies) are determined by the parameters of their neural networks. After each episode, reinforcement learning algorithms update these parameters to maximise cumulative rewards. The resulting trained agents serve as the best responses in each PSRO iteration.

The strategies of trained agents can be stochastic and non-deterministic, depending on the specific reinforcement learning algorithm used. Each agent is trained against a mixed strategy of its opponent, derived from the current equilibrium of the meta-game. In each episode, a single opponent strategy (potentially stochastic) is sampled from this mixed strategy and held fixed for the duration of the episode. A different opponent may be sampled in subsequent episodes. Algorithm 2 outlines the PSRO framework employed in our setting.

This training setup should not be confused with Independent Reinforcement Learning (InRL) in multi-agent reinforcement learning (MARL), as used in many prior works such as [9]. In InRL, all agents interact simultaneously, observe the state of the environment, choose and execute actions, transition to a new state, and all of them update their parameters based on the rewards at the end of each episode. In contrast, in our framework, we keep

the opponent fixed during training, and only the learning agent's parameters are updated while the opponent's network remains unchanged. Therefore, even though we train multiple agents across the PSRO iterations, we do not train them simultaneously. We adopt this approach to stabilise the learning process of the training agent, as the multi-round pricing game considered in our setting is significantly more complex than those studied in prior works.

Another natural question is why we use the Nash equilibrium as the meta-strategy solver in PSRO. The reason is that our goal is to study the Nash equilibria of the hyper-game, as we are interested in identifying stable solutions for comparison with the subgame perfect equilibrium (SPE). Non-equilibrium solutions are not stable under deviations and therefore fail to serve as suitable and durable market outcomes. Consequently, the Nash equilibria of the meta-game offer the best available approximation of stable solutions in the hyper-game.

An important challenge in using PSRO with the Nash equilibrium as the meta-strategy solver is the computational difficulty of computing equilibria in two-player games. This problem is known to be PPAD-hard [10, 12]. The lrsNash algorithm, developed by Avis, Rosenberg, Savani and von Stengel [3], is a well-known method for computing all Nash equilibria in bimatrix games. However, its exponential time complexity becomes prohibitive when the meta-game exceeds roughly 20×20 strategies in our experiments.

To address this, we adopted the tracing procedure algorithm introduced by Harsanyi [26] and implemented by my co-author, Bernhard von Stengel, as discussed in Section 6.3.3. The tracing procedure serves as both an equilibrium selection method and an efficient way to compute Nash equilibria of the meta-game.

For small to medium-sized meta-games (fewer than 200 strategies per player), the computational cost of equilibrium computation is negligible compared to the time required to train new agents. Additionally, the tracing procedure tends to yield equilibria that are dynamically stable. In PSRO, we do not need to compute all equilibria of the meta-game; therefore, this algorithm is well-suited to our purposes.

A known challenge with using PSRO and Nash equilibrium as the meta-strategy solver is overfitting to the opponent and the environment. We encountered this in our early experiments. To mitigate it, we introduced more stochasticity into the final agents and adopted learning algorithms with parameter settings that promote greater exploration.

5. Initial Reinforcement Learning Experiments

In the previous chapter, we explained that, as part of the PSRO framework (specifically lines 5 and 6 of Algorithm 2), we use reinforcement learning to train low-cost and high-cost agents as best responses to the meta-game Nash equilibrium strategies.

In this chapter, we focus solely on the modelling and reinforcement learning (RL) aspects of the framework, without yet incorporating the full PSRO procedure. Our goal is to identify the most suitable modelling specifications and the most effective RL algorithm for computing high-performing strategies. That is, we aim to find the algorithm that provides the best approximation of best responses in the eventual PSRO setup.

We describe our initial experiments in modelling the pricing game as a reinforcement learning problem and implementing *policy gradient* algorithms to learn pricing strategies by playing the game against a fixed opponent. The fixed opponents used for training are pure or mixed strategies of the predefined deterministic pricing strategies explained in Section 4.3.2, as this forms the starting point of the PSRO framework that we aim to prepare for.

We began with policy gradient methods because they are well-suited for episodic games, such as the pricing game we study. Moreover, these methods are relatively simple to implement and offer strong convergence properties. They can also be extended to continuous action spaces, which we intend to explore later. In contrast, such extensions are more challenging with alternative approaches like Q-learning.

In our models, the pricing game is implemented as the environment in which the firms, modelled as learning agents, interact and compete. We described the components of our RL model in Section 4.1.1.

We have implemented all the algorithms and the environment explained in this chapter from scratch, without using any RL framework models. Therefore, we explain the details of each model, the reason behind each modification, and, finally, the results that led us to the next model.

To represent our pricing game in the reinforcement learning framework, we treat each firm as an agent. These agents set prices by selecting their actions in the environment, which updates the game parameters and transitions the game to the next stage. The pricing game consists of 25 stages, which make one episode, from the initial state to the terminal state.

Hence, in our experiments, the episodes have a fixed length of 25. The training process involves these agents playing the underlying game for millions of episodes, with rewards serving as feedback to update their policies.

As described in Section 4.1.5, in the policy gradient algorithm, we optimise the policies by updating their parameters in the direction of the gradient of the cumulative reward function. A common practice in policy gradient methods, as in our model, is to use neural networks to parametrise the policy. This is due to their ease of use, their ability to approximate complex policies, and the computational efficiency of automatic gradient computation via backpropagation.

In this chapter, we discretised the action space for simplicity. Our implementation is based on the conventional method for policy gradient algorithms, as explained in [75]. The neural network receives the state as the observation, and the output of the neural network is a probability distribution over the set of actions, known as the *SoftMax distribution*. The agent follows the policy defined by the neural network by sampling an action from this distribution. At each policy update, the parameters of the neural network are updated based on the feedback received from playing an action and the probability that the action had in the distribution.

We use the notation π_θ for the policy defined by the parameter set θ from the neural network. Here, $\pi_\theta(a|s)$ denotes the probability that the policy assigns to playing action a in state s based on the SoftMax probability distribution that is output by the neural network.

Additionally, throughout this chapter, while evaluating the impact of various learning parameters, we occasionally experiment with a simplified version of the pricing game, referred to as the *3-stage pricing game*. This variant retains the same structure and mechanics as the main pricing game but limits it only to three stages, with four discrete actions available at each stage. This reduced setting allows for more efficient assessment of learning performance and parameter sensitivity, as it significantly reduces training time and facilitates easier tracking of the learning process.

This forms the basis for the models implemented in this chapter. In the following sections, for each defined model, we elaborate on the specific design choices, modifications, objectives, and results that motivated the progression to the next model.

5.1. REINFORCE Algorithm

The first model we implemented is known as the *REINFORCE* method [64, p. 326], which is the simplest form of policy gradient algorithms (see Section 4.1.5). As we progress with the experiments, we further develop this algorithm.

We broke down the training objective into multiple levels and gradually increased the complexity as the algorithms mastered the earlier ones. The ultimate goal, in preparation for the PSRO framework, is to train agents that can adapt their play against a mixed strategy of opponents. These opponents may play stochastically and might themselves have been trained by the reinforcement learning algorithms.

In the early experiments, we focus on training agents against one fixed deterministic opponent in order to monitor the learning.

Algorithm 4 REINFORCE Algorithm

Initialise neural network (NN) with random parameters θ
for each episode **do**
  Reset environment, set initial state s
  Initialise empty lists: `rewards` $\leftarrow$ [], `log_prob` $\leftarrow$ []
  for each stage t in range $T = 25$ **do**
    Sample an action from the distribution defined by the network:
    $a_t \sim \pi_\theta(\cdot|s)$
    Play action a_t, observe next state s' and reward r_t
    Store reward r_t: `rewards` $\leftarrow$ `rewards` $\cup \{r_t\}$
    Store log probability of action a_t:
    `log_prob` $\leftarrow$ `log_prob` $\cup \{\log \pi_\theta(a_t|s)\}$
    Update current state: $s \leftarrow s'$
  end for
  Compute the returns vector G where:
    $G_t = \sum_{k=t}^{T} \gamma^{k-t}$ `rewards`$_k$
  Loss function: the negative sum of the product of returns and the log-probability of the actions taken

$$\texttt{Loss} = -\sum_{t=0}^{T} G_t \cdot \texttt{log\_prob}_t$$

  Update parameters θ using gradient optimisation.
end for

We implemented the REINFORCE algorithm as stated in Algorithm 4. In the following, we explain how we defined the concepts of our RL model, explained in Section 4.1.1, which are necessary for implementing the algorithm, based on their roles in our modelled game and the rationale behind these definitions.

Action space

The agent's action at each stage should determine the price. Here, we use some information about the game to avoid prices that are inferior in all circumstances. Since we know that any price higher than the myopic price is not rational (as it reduces both the demand potential at the next stage and the profit at the current stage), we consider actions as how much *below* the myopic price the chosen price is. A lower price will attract more demand and consequently lead to higher rewards in later stages.

In the models presented in this chapter, we consider the action space to be discrete. At each stage, the price determined by the action is defined as follows: we allow actions $a \in \{0, 1, \ldots, 20\}$ with a step size of 3. This means that the price can be up to 60 units below the stage's myopic price.

For example, for the low-cost player ($c_1 = 57$), if $D_i = 180$ at the start of stage i and the chosen action $a = 5$:

$$\texttt{Myopic price: } P^* = \frac{180 + 57}{2} = 118.5 \quad \text{and} \quad P = P^* - (a \times \text{step}) = 118.5 - 5 \times 3 = 103.5.$$

We introduced the step size to reduce the number of possible action sequences, as the learning process was otherwise too slow.

The actions are determined by sampling from the SoftMax probability distribution over the action space, which is the output of the neural network.

The discretisation of the action space was introduced for simplicity in early experiments. However, a discrete action space may result in missed opportunities to find the optimal price point. In the experiments in the next chapter, the action space is instead considered continuous within the same range of prices below the myopic price.

State representation

The choice of state plays an important role in the agent's learning, as it provides all the information needed for making a decision at each stage, which is known as the *Markov Property*. However, if the state size is too large, it increases the computational complexity of the learning algorithm, as the agent requires more time to identify important features.

In this experiment, we define the state as follows:

encoding of stage	demand potential	last price	opponent's price history

with components for agent i and opponent $-i$:

- current stage t of the game
- agent's current demand potential D_i^t
- agent's price in the last stage P_i^{t-1}
- memory of opponent's prices $P_{-i}^{t-1-k}, \ldots, P_{-i}^{t-1}$

As we discussed before, in the last stage, the optimal price is the myopic price, but this does not necessarily hold for other stages. Therefore, the stage should be passed to the agent in order to play optimally and to understand where it stands in the pricing game. In the early implementations, we passed the stage of the game in the state as an integer representing the current stage. Hence, stage $t \in \{1, 2, \ldots, 25\}$.

The current demand potential shows how well the agent has played so far and also gives a hint about the payoff in the next stages. The current demand potential shows the agent where she is standing in the game. In our model, the agents do not have a perfect memory and can just remember *some* previous actions (prices) of the opponent. The reason we do not include all previous prices is to keep the size of the state restricted. (In our later models, we increase the number of previous prices in the state, both for the agent and the opponent.)

The agent's price in the previous state, in addition to demand potential and the opponent's previous price, provides sufficient information about the agent's place in the game.

At last, we pass the k last prices of the opponent to the agent. The main purpose of this memory is to lead the agent to construct a model of the opponent and, consequently, play accordingly against different opponents. In our early implementation, we set $k = 3$.

As evident from this representation, the agent perceives the opponent's effect as part of the changes in the environment and the next state. Moreover, during the training of the learning agent, the opponent's model remains fixed.

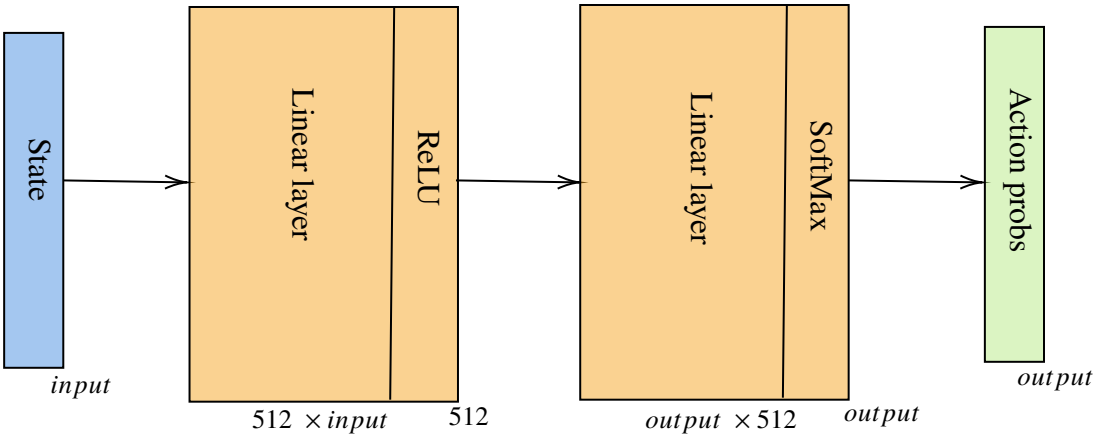
In later experiments, where the opponent is also a learning agent, the strategies become stochastic, increasing the complexity for the agent. Allowing both agents to train simultaneously would lead to a non-stationary environment, and we suspected that the agents would struggle to learn their opponent's strategy because it would constantly change.

Structure of the Neural Network

We employ neural networks as function approximators for the policy, where the parameters define the probability distribution to be played at each state. To avoid the complexities associated with intricate neural network architectures, we selected a simple model that captures non-linear behaviour while remaining straightforward and flexible.

Moreover, in the meta-game, we need to save, load, and manage numerous neural networks. Therefore, it is essential that these models are memory-efficient, as many of them will be stored and loaded simultaneously during each experiment.

In these initial experiments, we implemented the following structure:



The input to the neural network corresponds to the size of the state, and the output matches the size of the action space. The network processes the state through a linear layer followed by a rectified linear unit (ReLU) activation function. This is then passed through a second layer with a SoftMax activation function, which outputs a probability distribution over the possible actions. The agent samples from this distribution to select the action to play in the pricing game.

We explain these layers in detail below.

We denote the input of the neural network (which represents the state) by x and its output (which will be the SoftMax probability distribution) by y. There are two internal layers with intermediate vectors z_1 and z_2 of the relatively high dimension 512, and z_3 of

the same dimension as y, which is the number of actions. The process is as follows:

$$\begin{aligned} x \quad &\rightarrow \quad \Big[z_1 = W_1 x + b_1 \quad \rightarrow \quad z_2 = \texttt{ReLU}(z_1) = \max(z_1, 0) \Big] \\ &\rightarrow \quad \Big[z_3 = W_2 z_2 + b_2 \quad \rightarrow \quad y = \texttt{SoftMax}(z_3) = \tfrac{e^{z_3}}{\mathbf{1}^\top e^{z_3}} \Big] \quad \rightarrow \quad y \, . \end{aligned}$$

In detail:

(i) The state representation vector x is fed into the network.

(ii) In the first linear layer, we have the linear transformation of the input vector with weight matrix W_1 and the bias vector b_1. We defined the number of rows in the weight matrix to be 512, which is a relatively wide layer.

$$z_1 = W_1 x + b_1$$

(iii) z_1 will then be passed through the ReLU activation function. This activation function introduces non-linearity into the model, making the model more flexible and enabling it to understand more complex patterns. This function acts elementwise and maps negative values to zero, without changing the positive values.

$$z_2 = \texttt{ReLU}(z_1) = \max(z_1, 0)$$

(iv) The next layer is another linear transformation with weight matrix W_2 and the bias vector b_2. The output of this layer is a vector with the same size as the number of actions. Since it receives z_2 as input, the dimensions of W_2 are $output \times 512$.

$$z_3 = W_2 z_2 + b_2$$

(v) Finally, z_3 will pass through a SoftMax activation function. This function applies an exponential function to each element of z_3, and then divides by the sum of elements. Hence, the output forms a probability distribution over the set of actions, where all elements are positive and sum up to 1. Applying the exponential function puts more weight on the higher values, making the differences more noticeable. This helps direct our model to recognise the best action by assigning higher weights to it.

$$y = \texttt{SoftMax}(z_3) = \frac{e^{z_3}}{\sum_i e^{z_3^i}}$$

We emphasise that the reason we chose to use fewer but wider layers in our neural network is due to both simplicity and the fact that papers such as the following demonstrate that wide layers are sufficient to capture the complexity. Brightwell, Kenyon and Paugam-Moisy [6] proved that a function f defined on a compact set with not too many overlapping input regions can be learned with just one wide layer in the neural network. In a more recent paper, Jacot, Gabriel and Hongler [33] explain why wide networks often achieve

good generalisation and reliable training behaviour by describing the evolution of artificial neural networks using a kernel.

Loss function and policy update

After we collect the rewards for the whole episode, we define the returns at each stage based on the (in the general model discounted) rewards from that stage onwards: Assume that $r_1, r_2, \ldots, r_{25}$ are the rewards from this episode. The return vector G is defined as follows:

$$\begin{aligned} G_{25} &= r_{25} &&= \\ G_{24} &= r_{24} + \gamma r_{25} &&= r_{24} + \gamma G_{25} \\ G_{23} &= r_{23} + \gamma r_{24} + \gamma^2 r_{25} &&= r_{23} + \gamma G_{24} \\ &\ldots \\ G_1 &= r_1 + \gamma r_2 + \ldots + \gamma^{24} r_{25} &&= r_1 + \gamma G_2 \end{aligned} \tag{5.1}$$

The returns are computed backwards from the last stage. This definition of returns is compatible with the policy gradient theorem (4.2), as stated in Section 4.1.5. The returns demonstrate the effect of future state rewards on the current state. If we were to consider only the reward from the current stage, then the best price for the agent would always be the myopic price, as it maximises the one-stage reward. In general, the discount factor γ allows the agent to prioritise current payoffs while also considering future rewards. We initially experimented with different values for γ in the range of $[0.9, 1]$, but concluded that $\gamma = 1$ yielded the best learning results. In addition, setting $\gamma = 1$ matches the assumption about the pricing game where the game length is fixed and all stage rewards have equal weight for the total reward.

Finally, we define the feedback signal of each stage's action to update the policy parameters. The goal in RL is to maximise the cumulative rewards. Since optimisation algorithms are typically minimisers, we minimise the negative of our return signal. The loss function is defined as follows:

$$L = -\sum_{t=1}^{25} G_t \log(\pi_\theta(a_t \mid s_t))$$

This loss function computes the negative sum of returns weighted by the log-probability of the actions taken at each stage. By differentiating this loss with respect to the policy parameters θ, we obtain gradients that guide the update of the policy in the direction that increases the likelihood of actions yielding higher returns. The returns G_t effectively act as weights, reinforcing actions that led to better outcomes.

The optimisation algorithm we used to optimise the loss function is the Adam optimiser [39]. The Adaptive Moment Estimation (Adam) optimiser is a first-order gradient-based optimisation algorithm for stochastic functions, often more suitable for noisy settings than Stochastic Gradient Descent (SGD). Adam estimates the first and second moments of the

gradients, adapts learning rates for each parameter, and bounds the magnitude of parameter updates. It has shown particularly good results in training deep learning models due to its efficiency and robustness in handling large, complex datasets [39].

In our experiments, we employed the implementation of PyTorch Contributors.

Results

Although we tweaked our assumptions to improve the learning process, this model could not capture the complexity of the game. We trained this model against fixed opponent strategies for many iterations and plotted the total return over 25 rounds against the iterations. Figure 5.1 shows the return plot of low-cost agents against a fixed Myopic-strategy opponent on the left and a Fight-strategy (a simpler version of the Guess strategy) opponent on the right. As demonstrated by the plot, the agents did not learn effectively from the experiences. In the left plot, we observe that the agent converges to a strategy with low payoff, while in the right plot, the agent does not show any improvement after 1 million iterations.

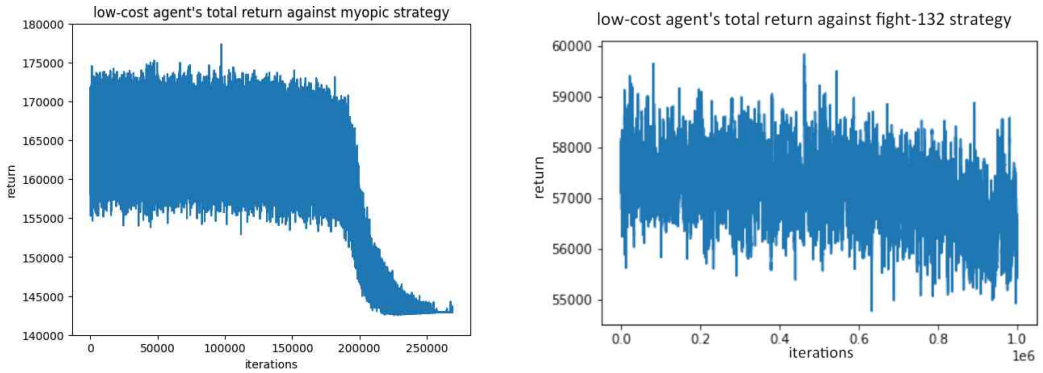


Figure 5.1.: Plot of total returns for a low-cost agent trained using the REINFORCE algorithm against the myopic-strategy on the left and the fight-strategy on the right.

This was not the behaviour we expected since we simplified many characteristics of the model. We repeated these experiments with different learning rates, discount factors, and opponent strategies. In some experiments, we observed better progress, but this seemed more due to chance, as repeating the experiment with the same parameters did not yield the same results.

Following these results, we chose to experiment with a reduced version of the pricing game, limited to just 3 stages, as it is simpler to learn and more tractable for performance assessment. In Figure 5.2, we observe experiments with different values of the discount factor γ in the 3-stage pricing game with 4 actions at each stage. Each experiment was repeated 3 times (3 trials), and the average return of these trials is plotted in Figure 5.2. All agents were trained against a myopic-strategy player, with the learning rate fixed at 0.0001.

Although the plots of the average return in the smaller pricing game suggest better learning compared to those in the main pricing game (Figure 5.1), we must remember that there are only $4^3 = 64$ different strategy profiles in the smaller game, making it excessive to train the agents for $500,000$ iterations just to explore these. In the main pricing game,

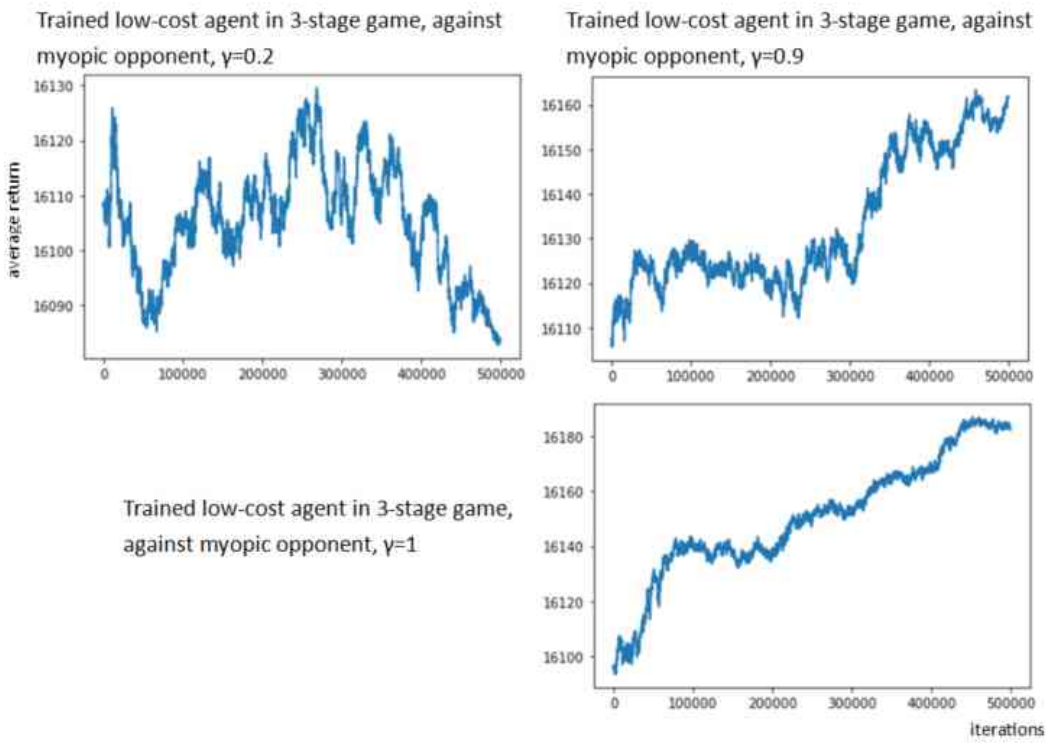


Figure 5.2.: Plot of total returns in the 3-stage pricing game with 4 actions at each stage, for 3 low-cost agents trained using the REINFORCE algorithm against myopic strategy with different γ values.

with discretisation and allowing 21 actions at each stage, there are 21^{25} possible strategies, which means that the current learning model is not effective. Therefore, we tried different techniques to enhance the model, as explained in the next sections.

Another point we observe in these plots is that a high value for γ helps the agents better account for the future effects of their actions.

5.2. REINFORCE with Baseline

Our first approach to improving the learning algorithm was to make the policy's feedback more informative to the agent. To this end, we introduced a baseline for the returns, allowing the agent to distinguish between below-average and above-average outcomes. By centering the feedback around this baseline, the signal becomes more nuanced, with smaller positive or negative values that help the agent more effectively identify and reinforce better-performing policies.

The first baseline we considered was the mean return for each stage. So, at iteration $iter$, the baseline is defined as follows:

$$\mathsf{Base}^{(iter)} = \frac{1}{iter} \sum_{i=1}^{iter} G^{(i)}$$

Then, the feedback for the returns is computed by subtracting the baseline from the returns, $A_t^{(iter)} = (G_t^{(iter)} - \texttt{Base}_t^{(iter)})$, which is known as the *advantage function*. Thus, the loss function is as follows:

$$L^{(iter)} = -\sum_{t=1}^{25}(A_t^{(iter)}) \cdot \log(\pi_\theta^{(iter)}(a_t|s)) \tag{5.2}$$

As defined, the advantage would initially be zero since $\texttt{Base}^1 = G^1$. However, as Base updates with the returns from the following iterations, the loss function takes positive values for returns above average and negative values for returns below average.

To test these changes to the learning algorithm, we first applied them to the 3-stage pricing game defined in the last section.

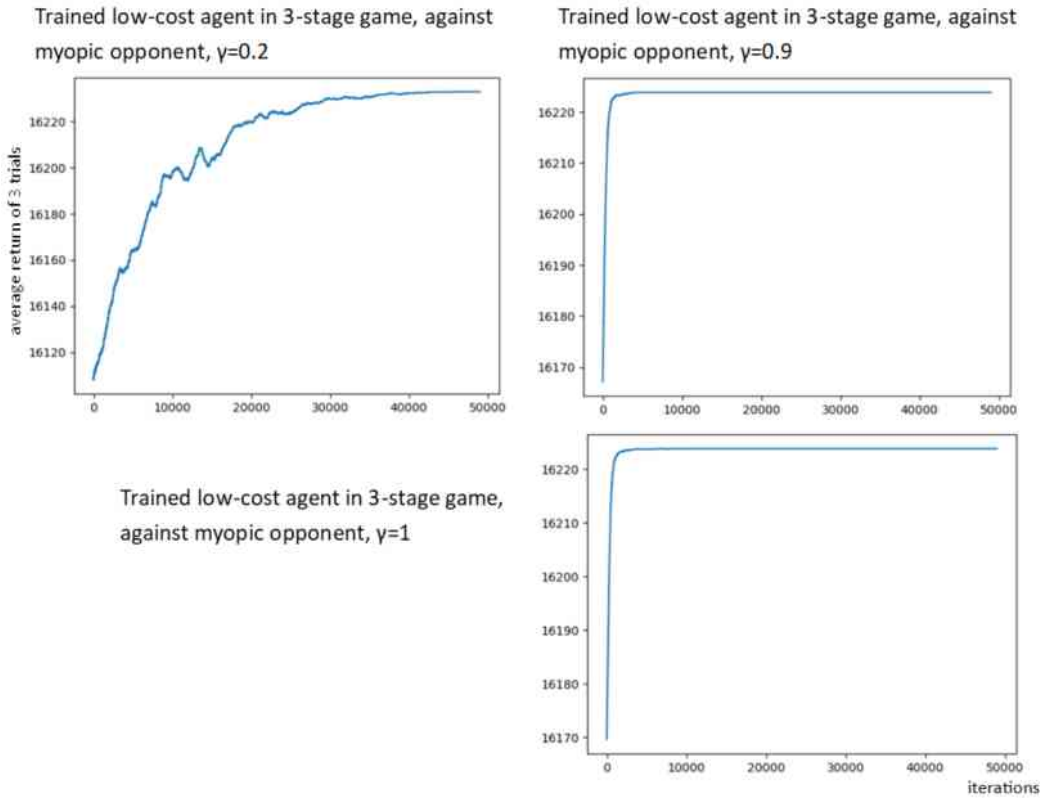


Figure 5.3.: Plot of the average returns over 3 trials of trained low-cost agents playing in the 3-stage pricing game with 4 actions at each stage. The agents are trained using the REINFORCE algorithm with a mean return baseline, competing against a myopic strategy with different γ values

Figure 5.3 shows the plots of the average return from 3 trials after adding the baseline. Comparing these plots to Figure 5.2, we observe how effective this change is for the agent in finding better policies. Notably, the number of iterations in the new case with a baseline is 50,000, which is one-tenth of those from the previous experiments. The learning rate remains fixed at 0.0001, and we can see the evolution of returns for different discount factors γ. As expected, higher values of γ are more effective, and since our pricing game has a

fixed number of stages, it makes sense to consider $\gamma = 1$, giving future rewards the same importance as current rewards, as suggested by the experimental results.

As shown in the plot for $\gamma = 1$, the agent is directed toward the optimal policy in fewer than 5000 iterations, making it over 100 times more efficient than the previous algorithm. This improvement significantly increased the efficiency of our learning algorithm. However, further adjustments are required to achieve a more efficient learning algorithm for handling the large state space of the main pricing game.

One-hot encoding

One measure we considered for evaluating the success of the trained strategies was learning the end-effect of the game, i.e., playing myopically in the last stage. The results from the first REINFORCE algorithm did not show this behaviour, which was understandable. However, even after adding the baseline in the short 3-stage game, learning the end-effect was inconsistent and occurred slowly, even in such a small game. Since the end-effect is specific to the last stage, we suspected that the agent cannot recognise the importance of the stage parameter in the state. Therefore, we decided to encode the stage in a way that was easier to learn.

In the basic model, the stage was initially represented as a floating-point number in the range $[0, 1]$. To make the end-effect more apparent, we decided to use *one-hot encoding* to represent the stage in the state.

Given that the game consists of 25 stages, we employ an array of size 25. In this representation, during each stage, the corresponding index in the array is set to 1, while the rest of the indices are set to 0:

	i_1	i_2	...	i_{24}	i_{25}
stage 1	1	0	...	0	0
stage 2	0	1	...	0	0
...	...	...	...	...	...
stage 24	0	0	...	1	0
stage 25	0	0	...	0	1

Learning backwards

Another technique we applied to emphasise the end-effect for the learning agent is the concept of *learning backwards*. The game follows the same rules as before, playing a full 25-stage episode, but with a key difference: during the first n learning episodes, only the network parameters corresponding to the action taken in the final stage, θ_{25}, are considered in the loss function to be optimised and updated. Then, over the next $2n$ learning episodes, the parameters corresponding to the last two actions, θ_{24} and θ_{25}, are optimised, and so on. In the final $25n$ episodes, the network parameters for all 25 actions are optimised.

n episodes	θ_1	θ_2	$\dots$	θ_{24}	θ_{25}
2 n episodes	θ_1	θ_2	$\dots$	θ_{24}	θ_{25}
3 n episodes	θ_1	θ_2	$\dots$	θ_{24}	θ_{25}
24 n episodes	θ_1	θ_2	$\dots$	θ_{24}	θ_{25}
25 n episodes	θ_1	θ_2	$\dots$	θ_{24}	θ_{25}

This approach allows the agent to focus initially on optimising its behaviour in the final stages, helping it to better understand and respond to the end-effect. The focus is then gradually expanded to earlier stages. As the number of parameters considered in the objective increases, we proportionally increase the number of learning episodes to allow sufficient time for optimising a larger set of parameters.

Bounded rewards

Finally, we divided all rewards by 1000 to reduce the range of possible reward values because receiving large positive values for the return can make it more challenging for the agent to differentiate between better results. Dividing the rewards bounded the returns approximately within the interval $[0, 200]$, depending on the player's and the opponent's strategies.

Myopic baseline

In the short 3-stage pricing game discussed earlier, we subtracted a mean return baseline from each stage's return, enabling our agent to recognise better policies. However, extending this approach to the main pricing game requires a baseline that reflects the expected return based on the demand potential (influenced by the previous action) at the start of each stage.

To clarify, a positive return after subtracting the mean return baseline does not necessarily indicate that the player followed a good policy; it simply means that, on average, across all possible returns after any play leading up to that stage, this policy yielded a higher return. Defining a baseline that considers the game parameters at the start of each stage will help the agent to distinguish more accurately between effective and less effective policies.

This is why we defined the *myopic baseline* for the more complex setting. This baseline calculates the expected return if, starting from the same game parameters, the agent were to play myopically for all future states, and then subtracts that return from the stage's actual return. In this way, the agent is incentivised to find policies that perform better than the myopic strategy, which sets a higher standard compared to the mean-return baseline.

Results of added baseline

Applying all the techniques mentioned above, we trained low-cost agents against various fixed, high-cost opponent strategies. Figure 5.4 shows the returns during the learning iterations, as well as the prices and demand potential of the final agent playing the pricing game against a fixed high-cost, myopic strategy. To observe the effect of adding the baseline and the other adjustments, we can compare the returns plot with the left plot in Figure 5.1. Here, we see that the average learning path shows increasing returns, which shows that the agent is effectively updating its parameters toward better policies. Moreover, there is no reduction in returns, which shows that the agent is learning effectively from experience.

The fluctuations displayed in Figure 5.4 are acceptable since exploration is necessary to prevent the agent from getting trapped in local minima. We also included the prices and demand potential plots to study the end-effect. In these plots, the blue line represents our trained low-cost agent, while the yellow line represents the myopic opponent. At the final stage of the pricing game, the demand potential for our agent is 257. With the production cost of 57, the myopic price for the low-cost agent is computed as 157. As seen in the plot of prices, the agent sets prices within the range $[100, 111]$ during the middle stages (stages 3 to 24), but in the final stage, it raised the price to 151. Although this price is still below the myopic value, it demonstrates an improvement in the agent's strategy by indicating an understanding of the end-effect of the game, where playing a higher price in the final stage becomes advantageous. This is an important progress compared to our previous agents.

This algorithm achieved its initial goal: training an agent to play effectively against a fixed opponent. However, in the meta-game, the agent will face a mixture of opponents, with one fixed for each learning episode but potentially changing in the next iteration. This change of opponent adds further non-stationarity, which can be confusing for our learning agent since the same actions in the same states can yield different returns depending on

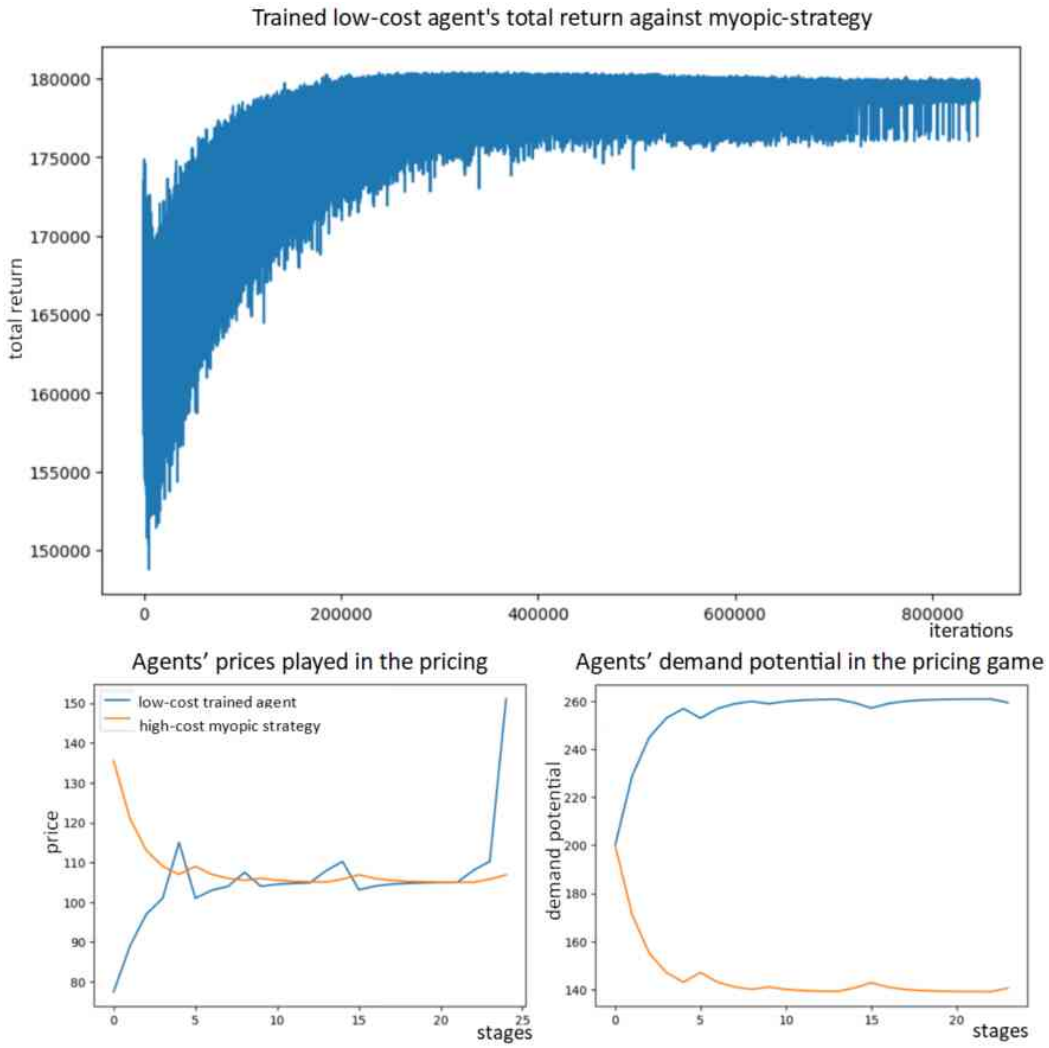


Figure 5.4.: Plot of returns over learning iterations, prices, and demand potential of a trained low-cost agent against a high-cost myopic strategy, with $\gamma = 1$. The agent is trained using the REINFORCE algorithm with a myopic baseline.

the opponent. Moreover, the opponents' strategies later in the meta-game are stochastic since they are trained strategies themselves, meaning that even the same opponent may act differently in similar scenarios.

Our agent receives no direct indication of which opponent it faces, so it must rely on the memory of previous prices played within each episode to understand the opponent's play. In previous experiments, we included memory in the state representation but did not focus on this point, as the opponent was fixed. At this stage, however, we want the agent to recognise different opponents and adjust its strategy accordingly. We need to point out that our goal for the agent is not to adopt a midpoint strategy that performs adequately against all opponents; rather, we want it to learn distinct strategies that perform well against each specific opponent.

Achieving this performance will be challenging, if even possible, because until the agent sees some of the play, it cannot know which opponent it is facing, even after supposedly being trained. The stochasticity of the opponent will add to the complexity on top of this.

To test the performance of the myopic-baseline policy gradient algorithm against mixed opponents, we set up the mixed opponent strategy as $\left(\frac{1}{3}\text{Myopic}, \frac{1}{3}\text{Constant-95}, \frac{1}{3}\text{Guess-132}\right)$. The strategies are explained in detail in Section 4.3.2. Each of these three strategies has an equal probability of being selected as the opponent in each iteration and will be fixed during the entire learning episode. We chose this setup because these strategies follow distinct price paths, making it easier for our agent to learn the price patterns. Given that our learning agent observes a portion of previous prices played by the opponent, the different price sequences should help the agent differentiate among the opponents and play accordingly.

We set the memory of the opponent's price equal to 4, with a learning rate of 10^{-5}. Since the opponent is different in each new episode, in Figure 5.5 we plotted the scatter plot of returns instead of the previous line plot. The three lines that the returns have converged to represent the returns against each of the opponents.

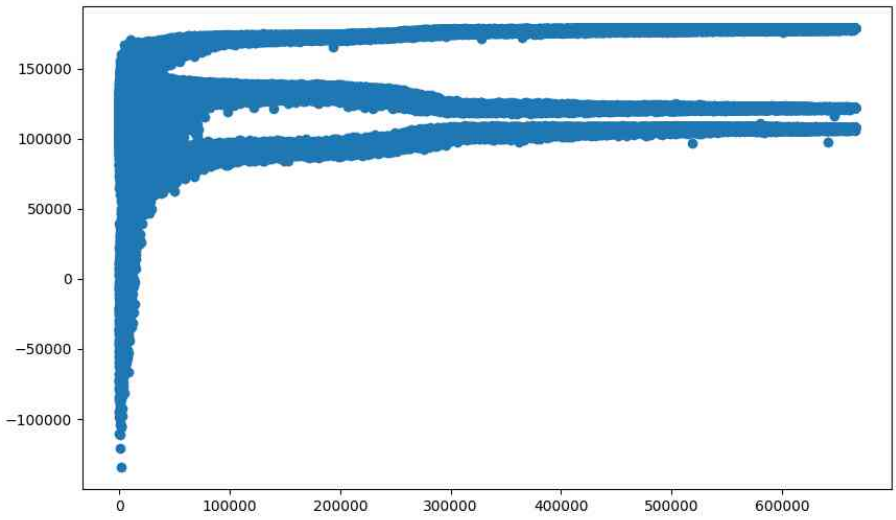


Figure 5.5.: Plot of returns of trained agent using myopic-baseline policy gradient method against the "myopic-constant-guess" mixed opponent, with a memory of size 4 and a learning rate of 10^{-5}.

To see if the agent can differentiate between the opponents, we tested our agent against each opponent 200 times and plotted the sequence of actions played by our agent in Figure 5.6. On the bottom, we can also see the sequence of prices played by the opponent to view the game from the agent's point of view. There are 600 sequences plotted, but as can be seen, they are all the same (except for a green deviation in one stage); hence, we can see only one line. This means that the agent does not differentiate between the opponents at all and plays exactly the same against all of them. However, a positive point is that in the final stage, the agent has learned to play nearer to the myopic price (choosing the action zero is equivalent to playing the myopic price), which is a positive result.

This result led us to look for more complex learning algorithms to help our agent learn better to play against mixed opponents. In the next section, we test and compare the actor-critic method, which is a more advanced algorithm.

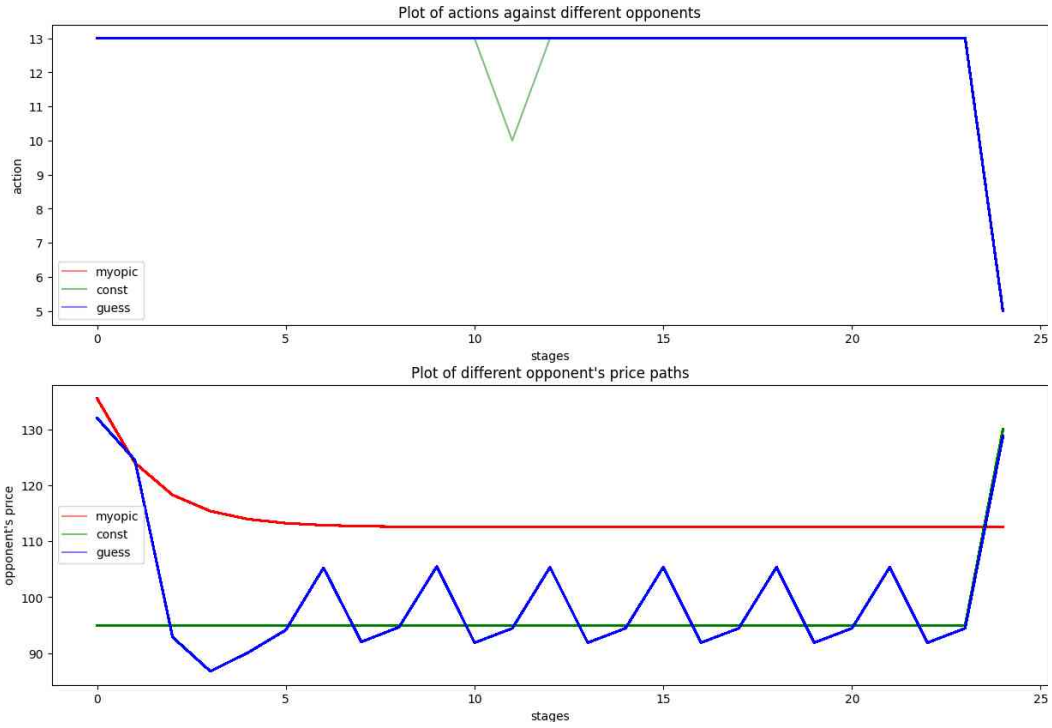


Figure 5.6.: Action sequence and opponent's price sequence plots for trained agent using the myopic-baseline policy gradient method, with a memory of size 4 and a learning rate of 10^{-5}.

Table 5.1.: Average return of the agent from 200 trials against each opponent. The agent is trained using the myopic-baseline policy gradient method against myopic-constant-guess mixed opponents, each with probability $\frac{1}{3}$.

	Myopic	Constant-95	Guess-132
trained against mixed	178,500	108,200	121,900
trained independently	179,100	116,600	130,400

5.3. Actor-Critic

The baseline that we introduced in the previous Section 5.2 was the simple version of the *actor-critic* method. As the name suggests, this method consists of two main parts, the *actor* and the *critic*. The actor represents the agent's policy: it observes the state and decides the action to play in the game. This part is what we had so far, implemented using a neural network. The new part is the critic, which replaces the baseline in the previous algorithm. The critic determines how much return is expected to be received for the given state. In the baseline algorithm, we used the average return, and later the myopic-strategy return as a baseline to compare the return received at each stage and determine if the policy is performing well. As we extend the model to a more stochastic setting, we need a baseline that can capture the effect of different opponents. Based on the opponent, the expected return should be different. Continuing with the myopic-strategy return would not necessarily be an optimal choice to compare with because, depending on the opponent, this strategy

may have an acceptable or unacceptable return. Considering the newly added complexities, this strategy might not be the best baseline to lead our agents.

Of course, this expectation of return should be learned itself from experiences as the game progresses, similar to the average return baseline; however, it should consider the state and identify the opponent from the price memory part of the state.

The actor and critic together help the agent learn better policies by guiding it towards policies with higher evaluations, as described in Algorithm 5. At each stage of an episode, the actor and critic both observe the state. The critic determines the expected return from this state and stores it. The actor independently selects an action, plays it, receives the reward, and stores it. When the episode is finished, the return is computed based on the rewards and compared with the evaluation from the critic. As before, the actor network will be updated according to the loss function based on the advantage (expected return minus evaluation). The critic network will be updated to minimise the error between the returns and the evaluations vector.

Algorithm 5 Actor-Critic Algorithm

Initialise Actor and Critic networks with parameters θ (actor) and θ' (critic)
for each episode **do**
    for each stage i in the episode **do**
        Receive state s_i from environment
        Critic observes s_i, estimates expected return $V(s_i;\theta')$ and stores it as V_i
        Actor observes state s_i and selects an action a_i
        Execute action a_i, receive reward r_i, and store it
    end for
    Compute discounted returns vector $G = \{G_i\}$ based on stored rewards, as defined in (5.1)
    Define the Advantage function: $A(s_i, a_i) = G_i - V(s_i;\theta')$
    Update Actor network parameters to minimise loss:
    $L_a = -\mathbb{E}\left[A(s_i, a_i) \cdot \log(\pi(a_i|s_i;\theta))\right]$
    Update Critic network parameters to minimise error:
    $L_c = \mathbb{E}[(G_i - V(s_i;\theta'))^2]$
end for

Neural network architecture

For the structure of the neural network in the actor-critic algorithm, we took inspiration from [75, p. 127]. Initially, there are two linear layers, similar to the structure used in the REINFORCE algorithm implementation, which are shared between the actor and critic. These layers process the input state, and by sharing them, we reduce the computational cost of having each network extract features separately. However, when updating the critic network, we do not want to impact the actor's parameters, because this can destabilise the actor's network since they have different optimisation objectives. To address this, the output of these shared layers is *detached* before being passed to the critic layers. This way, backpropagation for the critic only affects its own layers. Since the critic does not update the shared layers, we defined two additional linear layers specific to the critic. As shown in Figure 5.7 on the left side, these layers use the ELU (exponential linear unit) activation function instead of the more common ReLU (rectified linear unit).

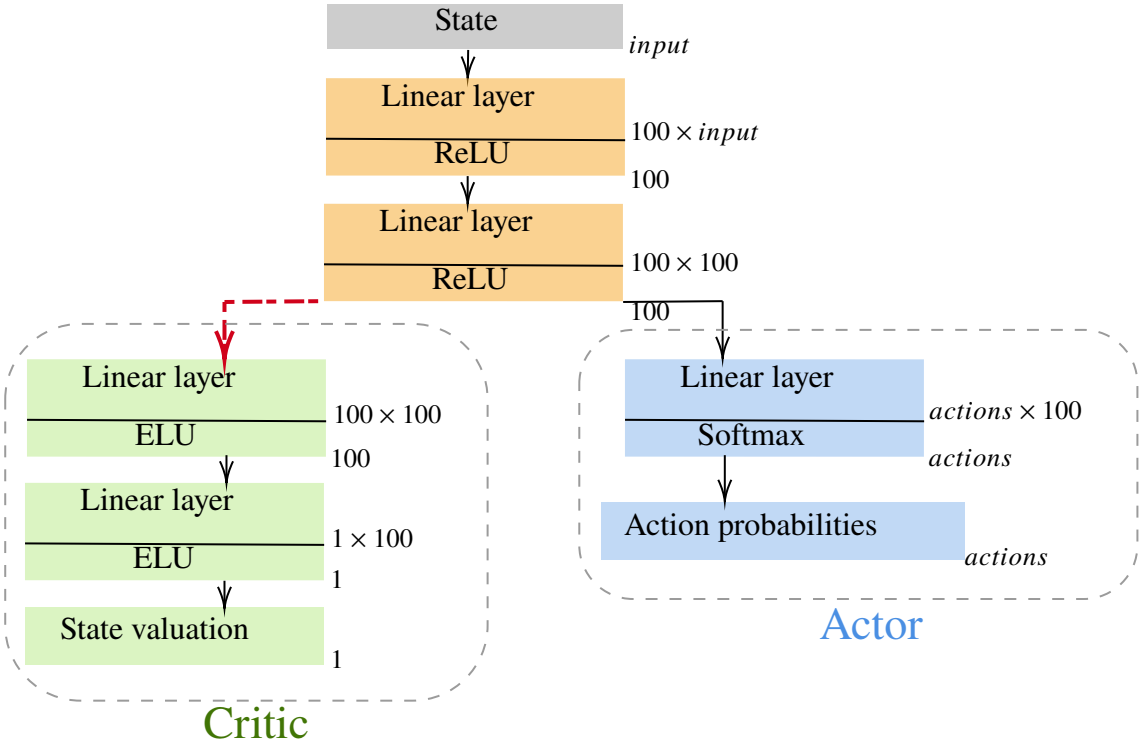


Figure 5.7.: The actor-critic model.

Typically, ReLU is a straightforward way to introduce non-linearity to the network. However, when using ReLU, we observed inefficient learning, as the network frequently faced zero gradients. ELU has smoother gradients, which improved learning stability, so we used this activation function instead. The output of the critic network is a single scalar representing the expected return from the given state.

On the other hand, on the right side in Figure 5.7, the actor receives the output of the shared layers, passes it through an additional linear layer specific to the actor, and then applies a Softmax activation function to output a probability distribution over the set of possible actions. As mentioned, only the actor updates the shared layers, so backpropagation from the actor's loss function adjusts the shared layers' parameters. At first, we tested the

new model against a fixed opponent, the myopic high cost strategy, to compare the results with the previous myopic baseline model.

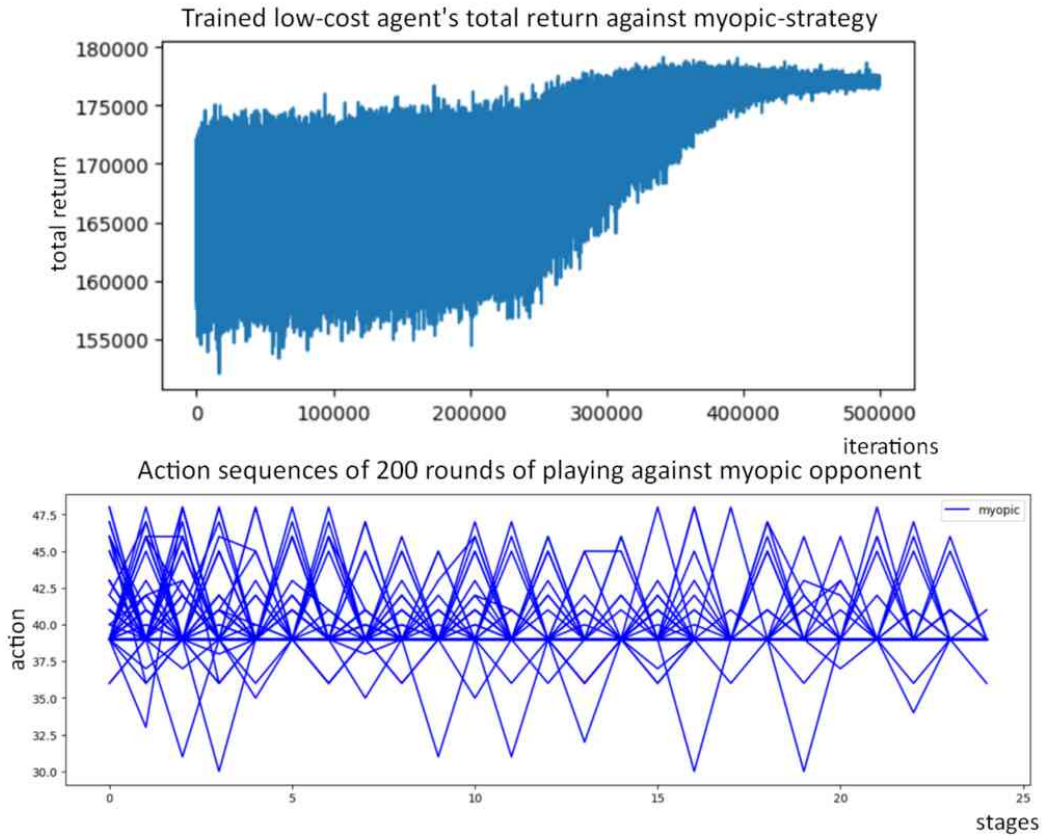


Figure 5.8.: Returns during learning and action sequence of 200 rounds of trained low-cost agent using the actor-critic method, against myopic opponent with a memory of size 0 and a learning rate of 10^{-5}.

We trained the agent using the actor-critic method against a myopic strategy.

The results were not as expected. In terms of learning, learning occurs, and the agent converges to policies with higher returns. The final returns were very close to the final returns of the myopic baseline model; however, they were slightly lower. This can be seen by comparing Tables 5.2 and 5.1 under the returns of independent models. Also, returns during learning are depicted in the top plot of Figure 5.8.

However, the bigger shock was the bottom plot of Figure 5.8, which depicts the action sequences of 200 rounds of playing against the myopic opponent and does not show any signs of learning the end-effect of the game. At the final stage, none of the action sequences show any signs of actions close to the myopic price (action=0). This was disappointing as we had hoped to improve upon the myopic baseline, which has reached this level. Nonetheless, we decided to proceed with evaluating the agent's performance against mixed opponents before completely dismissing this model.

To test whether our model can effectively adapt to different opponents, we defined the test similarly to the previous model.

We set up the mixed strategy as $\left(\frac{1}{3}\text{Myopic}, \frac{1}{3}\text{Constant-95}, \frac{1}{3}\text{Guess-132}\right)$. As mentioned earlier, in each iteration, an opponent is sampled from the distribution of opponents, and this sampled opponent remains fixed during the entire learning episode. In the next iteration, a new opponent is sampled.

We trained multiple agents with various learning rates, different memory lengths for opponent prices, and different numbers of episodes. We explain the results we obtained by comparing these experiments.

Similar to the previous section, in all the following cases, after training, the trained agent is played for 200 iterations against each opponent, and the actions' plot shows the action paths for each iteration. We have also plotted the opponent's price path to observe if the paths are distinguishable.

Number of episodes

It is common practice to set a convergence criterion for the number of episodes and continue learning until that criterion is met. We defined our convergence criterion such that learning would stop when the probability of all chosen actions in their distribution exceeds 99.9%. The plot in Figure 5.9 shows the results of this experiment with the following parameters:

Number of episodes: 1,923,538, learning rate: 10^{-5}, memory length: 4

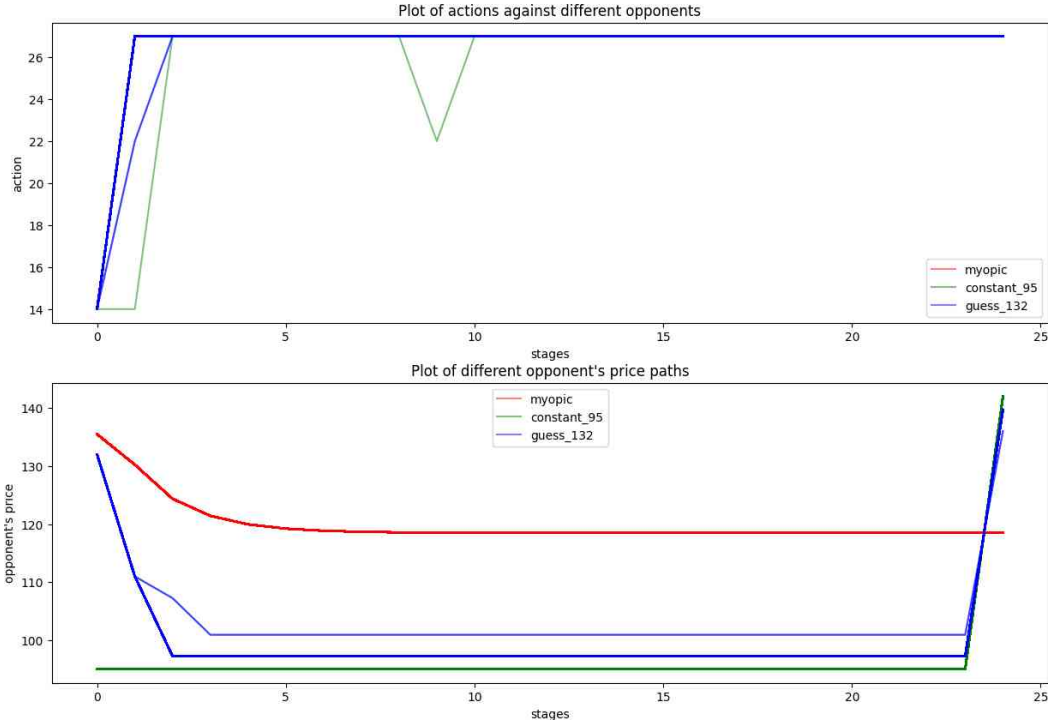


Figure 5.9.: Action sequence and opponent's price sequence plots for trained agent using the actor-critic method, trained for 1,923,538 episodes with a memory of size 4 and a learning rate of 10^{-5}.

As can be seen, although this is a plot of 200 trials against each opponent—meaning 600 paths in each plot—the plots appear quite empty because they coincide. This is an

indication of convergence, as the action distributions have become pure strategies with minimal oscillation. However, this is not the model we aimed for.

First of all, the action plot shows that the agent plays exactly the same against all opponents, whether they are Myopic, Constant-95, or Guess-132 strategies. This reveals that the agent cannot differentiate between opponents. Additionally, the action in the last stage is 27, meaning the agent plays 27 price units lower than the myopic price. Therefore, the action plot indicates that the agent does not learn the end-effect of the game, which we achieved with the less complicated baseline model.

As the opponent's price paths indicate, the price paths of opponents do not intersect and should be distinguishable for the agent, so we suspected that the problem might lie with the stopping criteria. We decided not to stop the learning based on the convergence criteria and instead set a fixed 5,000,000 episodes for learning, which is a relatively large number.

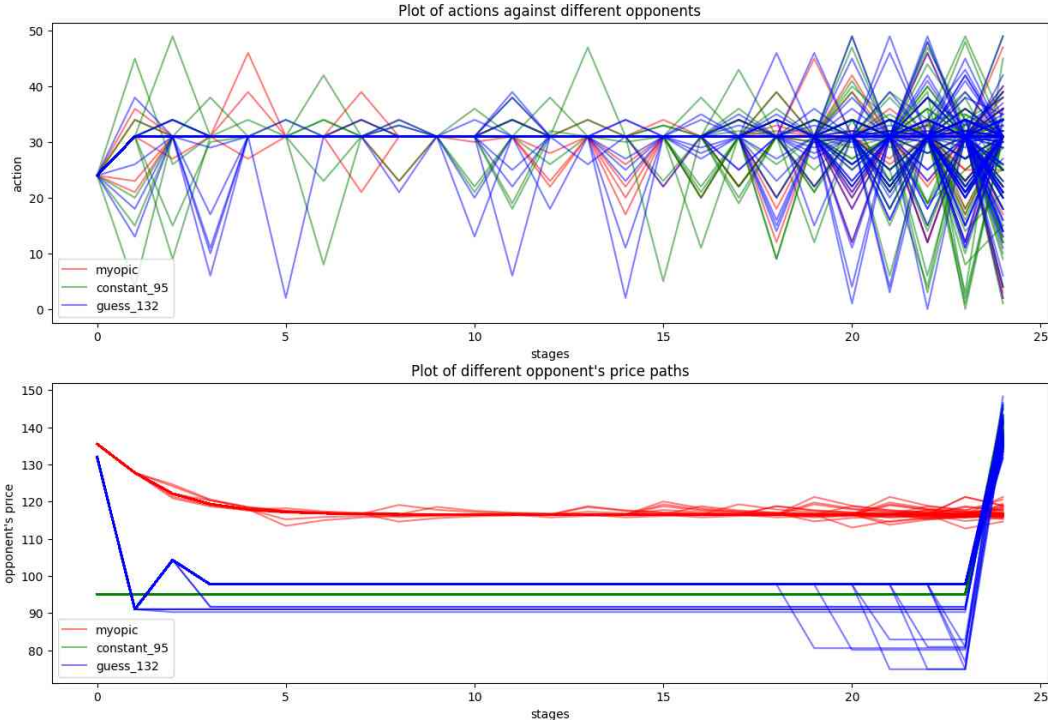


Figure 5.10.: Action sequence and opponent's price sequence plots for trained agent using the actor-critic method, trained for 5,000,000 episodes with a memory of size 4 and a learning rate of 10^{-5}.

The plots in Figure 5.10 show the action sequences and opponent's price paths for the agent trained with the same parameters, differing only in the number of episodes trained. The actions plot shows that the actions against different opponents vary in many cases, which is a positive outcome.

Observing the blue action sequences (against the Guess strategy), we see that the agent has not fully converged to a pure strategy against each opponent and follows multiple sequences. However, an encouraging observation is that the final action in many of these sequences has decreased to near zero. While we cannot fully count this as learning the end-effect, it is still an improvement over the previous experiment.

Learning rate

We experimented with different values of the learning rate, hoping that a higher learning rate might speed up the learning process. However, as is often the case, this led to quick convergence to playing the same action sequence against all opponents, without learning the end-effect. Increasing the number of episodes or changing the memory length did not improve this result.

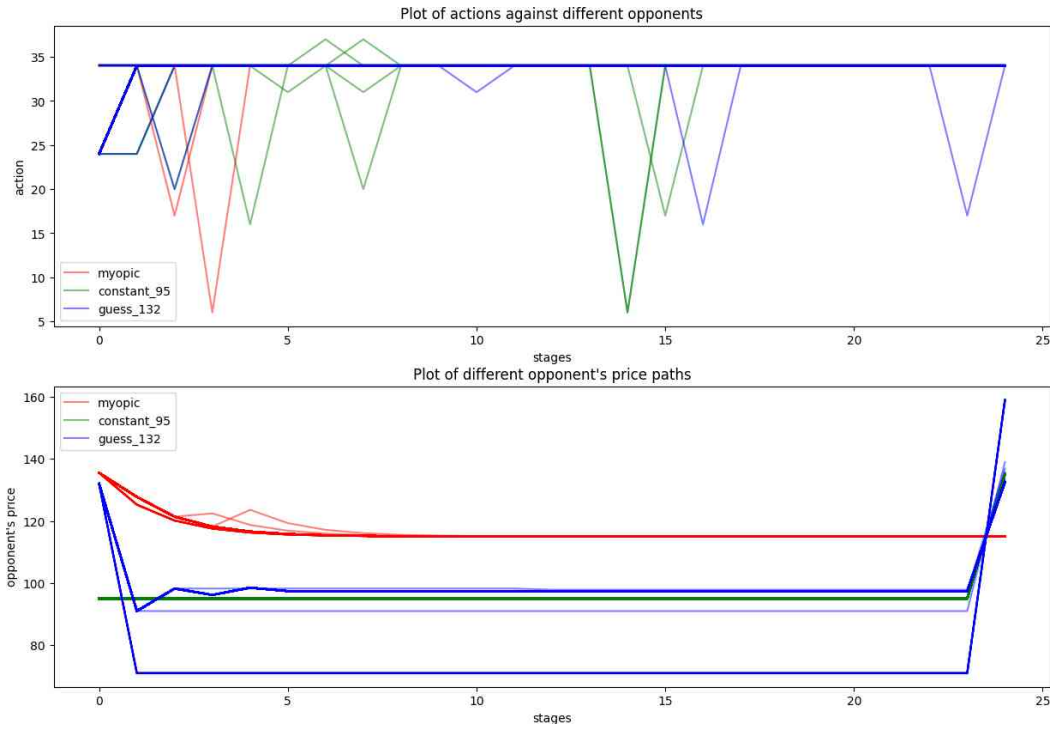


Figure 5.11.: Action sequence and opponent's price sequence plots for trained agent using the actor-critic method, trained for 5,000,000 episodes with a memory of size 2 and a learning rate of 10^{-4}.

Figure 5.11 shows the plot for an agent that we trained for 5,000,000 episodes with a higher learning rate of 10^{-4}. As shown in the action plot, this agent did not learn the end-effect and plays the same action sequence against all opponents.

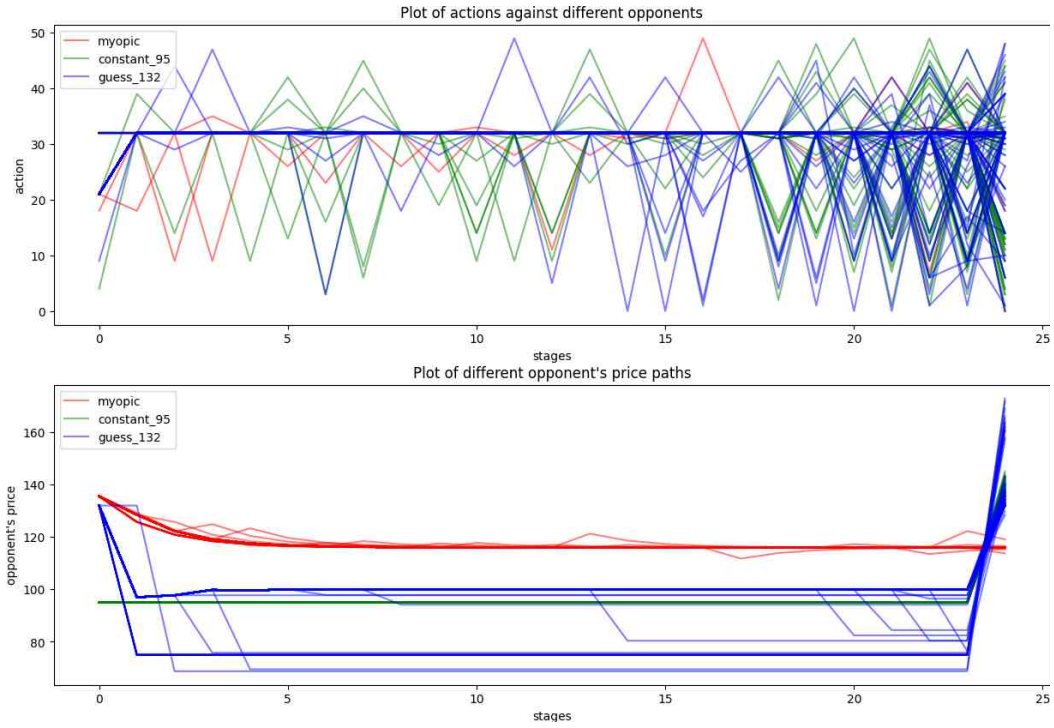


Figure 5.12.: Action sequence and opponent's price sequence plots for trained agent using the actor-critic method, trained for 5,000,000 episodes with a memory of size 2 and a learning rate of 10^{-5}.

Memory length for opponent prices

To examine the effect of memory size, we repeated the experiment plotted in Figure 5.10, this time with a shorter memory of size 2, as shown in Figure 5.12. The action variance in both experiments is high, and neither converged to a single pure strategy against each opponent.

Table 5.2 shows the average return of the agent from 200 trials against each opponent for these two experiments. In the last row, we included the returns of agents trained independently against each opponent (i.e., without mixed opponents) using the actor-critic method. Notably, these are not the highest possible returns; with a myopic baseline, the agent received a return exceeding 180,000 against the myopic opponent.

In Table 5.2, we observe that the experiment with a memory size of 2 performed better against all opponents, especially against the Guess strategy. A larger memory size increases the state representation length, potentially slowing down learning, which we suspect is happening here. Larger memory sizes add complexity, similar to adding more features in a model; this can improve performance but requires more training. For reference, we included the last row to show that, while the memory size of 2 has better performance, it still underperforms compared to agents trained independently, using the same algorithm and parameters.

After multiple experiments, adjusting each parameter, we observed a pattern: after 1–2 million episodes, the agent tends to select an action sequence with very high probability and low variance at each stage, which might be mistaken for convergence but actually results in

Table 5.2.: Average return of the agent from 200 trials against each opponent for different memory sizes

	myopic	constant-95	guess-132
memory = 2	174,500	96,600	111,300
memory = 4	174,100	95,700	107,900
trained independently	177,300	104,300	127,000

identical actions against all opponents. Extending the training to millions more episodes led to differentiated behaviour against opponents and signs of learning the end-effect.

From these comparisons, we concluded that the number of episodes needed to train the agent against mixed opponents using the actor-critic method is much larger than anticipated, making it highly time-consuming. This model is therefore too costly to be used for the meta-game. Increasing the learning rate did not help either, so to increase efficiency, we need to employ additional techniques to speed up learning.

If we compare the experimental results from the actor-critic method and the previous myopic-baseline REINFORCE algorithm, we observe that although the myopic-baseline algorithm is more simplistic by considering myopic as the baseline for all opponents, it still resulted in better learning of the end-effect of the game. Additionally, the returns in Tables 5.1 and 5.2 show that the myopic-baseline method achieved higher returns against each opponent, whether trained against mixed opponents or against each opponent independently. The only possible advantage of the actor-critic method in our experiments was a slight differentiation in behaviour against different opponents, though not to an acceptable level. Given the significantly higher returns of the myopic-baseline method and its success with the end-effect, the myopic-baseline method is clearly the better choice at this level of complexity of opponent strategies.

These observations led us to adopt the Stable-Baselines3 reinforcement learning framework, which provides efficient implementations of many advanced RL algorithms. Using this platform yielded more successful results, as discussed in the next chapter, which constitutes the main portion of our project.

6. PSRO Framework and Advanced RL Algorithms

In the previous chapter, we described relatively simple reinforcement learning algorithms, which we implemented ourselves in Python using the PyTorch library, along with several improvements. Although the results of the REINFORCE algorithm with a myopic baseline against fixed opponents were promising, we did not achieve our goal of learning differentiated behaviour against mixed opponents. Implementing these algorithms manually was time-consuming and increased the risk of potential errors. The results indicated that more advanced learning algorithms are needed to compute effective best-response strategies in the PSRO setting. In our implementation of basic RL algorithms, the mixed strategy of opponents seemed to "distract" the learning agent, leading to a slower learning process and ultimately preventing the agent from adapting to different opponents.

Therefore, at this point in the project, we decided to adopt advanced reinforcement learning algorithms to ensure a more stable training process, allowing us to move beyond the strategy training step and focus on analysing the meta-game in the PSRO setting. The efficiency of implementation and adaptability were the main requirements for the new RL framework, so that we could define our own setting and apply the learning algorithms to our training environment.

Before completely switching to the new RL framework, we decided to experiment with running the PSRO setting using our more successful self-implemented algorithm from the last chapter in order to compare the results later.

6.1. PSRO Framework Using REINFORCE with Myopic Baseline

We include this experiment as the final step in our self-implemented learning algorithms, which also serves as the foundation for the PSRO framework that we later refined and extended.

In the previous chapter, we obtained better results when training independent agents using the REINFORCE algorithm with a myopic baseline. Therefore, we adopt this algorithm as the learning method for training agents within the PSRO setting.

Following Algorithm 2, for initialising the meta-game, we decided to include the three strategies we previously tested in our experiments: *Myopic*, *Constant-132*, and *Guess-132*. These strategies expose the learning agent to varying levels of complexity. We modified the constant strategy to play at 132 instead of 95 (as used in earlier experiments) because we observed that the price path of *Constant-95* can resemble that of the *Guess-132* strategy (as seen in the price plot in Figure 5.12). In contrast, the price path of *Constant-132* is more distinct, making it easier for the agent to differentiate between opponents. Additionally, *Constant-132* facilitates higher payoffs for the learning agents, as it allows them to reduce the price more and increase their share of demand.

These three strategies are considered for both the low-cost and high-cost agents of the initial meta-game, forming a 3×3 game. Then, the equilibria of the meta-game are computed, and new low-cost and high-cost agents are trained against the chosen equilibrium. We used the tracing procedure, implemented by my co-author, Bernhard von Stengel, to compute the equilibria. We explain this algorithm in detail in Section 6.3.3. We used the tracing procedure to compute the equilibria, running 100 traces and selecting the equilibrium that appeared most frequently. This choice was based on the intuition that the most frequently occurring equilibrium is likely to be more dynamically stable than others. While we later explored alternative equilibrium selection methods, this was the setting used for the initial experiment.

We ran this meta-game over several days, as training each agent was time-consuming. The first equilibrium of the initial 3×3 game was (Guess-132, Myopic), where the low-cost (row) player played the Guess-132 strategy and the high-cost (column) player played the Myopic strategy. New agents are trained against the current equilibrium strategies, but only those achieving a higher expected payoff than the equilibrium payoff are added to the meta-game.

To compute the return of a trained agent against each strategy in the meta-game, the agents play against each other for 100 rounds. The payoff recorded in the meta-game is the average payoff across these rounds. This evaluation is crucial, as the trained agents follow stochastic strategies, and their returns may vary depending on whether they have converged to a pure strategy and the level of variance in their behaviour.

The game in Figure 6.1 was reached after six rounds, with five low-cost and two high-cost trained strategies added to the game. However, after this point, no additional strategies were added. We continued training agents for this game over 14 more training rounds for 14 low-cost and 14 high-cost agents, yet none of them entered the meta-game.

For future comparisons, we have plotted in Figure 6.2 the payoff pairs of the equilibria used to train new agents as the meta-game progresses. Additionally, the payoff of the subgame perfect equilibrium is included as a baseline reference.

As we gained more experience, we made several adjustments to the PSRO setting in later experiments. For instance, we considered multiple equilibria for training, introduced alternative equilibrium selection criteria, and employed multi-processing techniques to train agents more efficiently.

low-cost \ high-cost	myopic	constant-132	guess-132	H1	H2
myopic	93,895 139,591	94,732 138,555	94,732 138,555	117,874 98,440	118,409 97,159
constant-132	94,760 138,527	103,712 127,512	103,712 127,512	159,265 56,140	160,786 54,261
guess-132	79,666 154,998	68,729 168,709	79,091 141,639	70,510 124,168	69,823 121,632
L1	35,257 178,433	-40,901 294,260	32,920 91,786	44,166 122,729	43,625 120,188
L2	40,500 179,169	-21,026 272,487	44,007 104,497	51,182 125,383	52,000 123,073
L3	40,916 179,003	-20,436 271,526	46,605 107,660	52,011 124,716	52,516 122,747
L4	41,349 179,200	-19,699 271,077	47,435 108,604	52,067 125,874	53,243 123,156
L5	44,815 179,353	-11,059 261,888	53,380 116,425	56,485 126,715	56,184 124,336

Figure 6.1.: 8×5 meta-game that became stationary after training agents with REINFORCE and myopic baseline as the learning algorithm.

6.2. Advanced Learning Algorithms: New RL Framework

From this point in the project, we adopted the learning algorithms implemented in the Stable Baselines3 (SB3) framework to use within our PSRO setting.

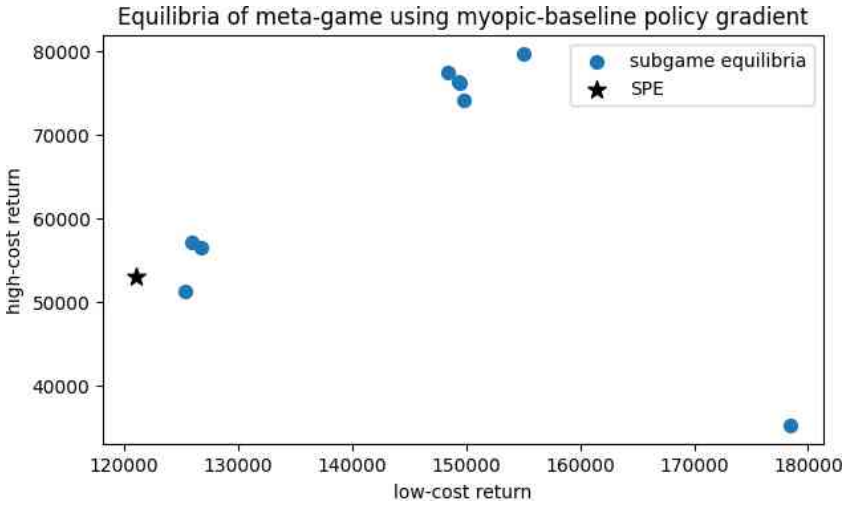


Figure 6.2.: Plot of equilibrium payoffs of subgames of the meta-game as the learning proceeds, using the REINFORCE algorithm with myopic baseline as the learning algorithm

The Stable Baselines3 (SB3) package, developed by Raffin, Hill, Gleave, Kanervisto, Ernestus and Dormann [54], provides reliable implementations of many reinforcement learning algorithms. These implementations are well-tested, efficient, and adaptable to different environments. Additionally, the extensive online documentation was very helpful in guiding us to make our code compatible with their structure.

We tested multiple learning algorithms from this framework on a shortened version of our pricing game, consisting of only 3 stages. In the end, two algorithms, *PPO* and *SAC*, proved to be the most successful in terms of achieving the highest returns against a fixed opponent. Based on these results, we decided to focus on these two algorithms for the remainder of the project. Each algorithm possesses important features, yet they differ substantially from one another.

6.2.1. Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) [60] is a policy gradient actor-critic method that improves upon the Trust Region Policy Optimisation (TRPO) algorithm [59]. TRPO is effective in optimising large nonlinear policies, such as neural networks, by maximising the advantage function subject to a Kullback–Leibler (KL) divergence constraint. As we mentioned earlier in Algorithm 5, the advantage function is the observed returns minus the estimated state value given by the critic network. The theoretical foundation of TRPO ensures monotonic improvement when the KL divergence penalty is incorporated into the objective function, as it limits the change between the old and new policies.

This penalty term is weighted by a fixed coefficient, denoted by ζ. However, in practice, the recommended value for this coefficient often leads to excessively small step sizes, and the optimal choice of ζ tends to vary across different problem settings. To address this issue, in practice, a bound on a heuristic approximation of the average KL divergence between the

old and new policies is considered as a constraint, called the trust region constraint:

$$\begin{aligned} \underset{\theta}{\text{maximise}} \quad & \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t - \zeta KL[\pi_{\theta_{\text{old}}}(.|s_t), \pi_\theta(.|s_t)] \right] \\ & \Downarrow \\ \underset{\theta}{\text{maximise}} \quad & \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t \right] \\ \text{subject to} \quad & \mathbb{E}_t \left[KL[\pi_{\theta_{\text{old}}}(.|s_t), \pi_\theta(.|s_t)] \right] \leq \delta \end{aligned} \tag{6.1, 6.2}$$

This transformation, although necessary for TRPO to achieve the best empirical results, leads to higher optimisation complexity. Proximal Policy Optimisation (PPO) addresses this by removing the constraint. However, without the KL constraint, optimisation can result in excessively large policy updates. To mitigate this, PPO limits the magnitude of policy updates by modifying the objective function.

$$\begin{aligned} \underset{\theta}{\text{maximise}} \quad & L^{\text{clip}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon) A_t \right) \right] \\ \text{where} \quad & r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \end{aligned} \tag{6.3}$$

In (6.3), r_t is referred to as the probability ratio, defined as the ratio of the probability of an action under the new policy to that under the old policy. If an action has a positive advantage, the probability ratio should be greater than one because we aim to increase the probability of selecting that action. Conversely, if the advantage is negative, the probability ratio should be less than one, indicating a decrease in the probability of selecting the action.

The second term in the objective function is the *clip* function, which controls the magnitude of the update by constraining the probability ratio within the interval $[1-\varepsilon, 1+\varepsilon]$. In our project, we utilise the Stable-Baselines3 framework, and we explain the implementation of the *clip* function as provided in this framework [51], as stated below. It is noteworthy that alternative implementations of the *clip* function are also feasible.

$$\begin{aligned} \underset{\theta}{\text{maximise}} \quad L^{\text{clip}}(\theta) &= \min \left(r_t(\theta) A_t, g(\varepsilon, A_t) \right) \\ \text{where} \quad g(\varepsilon, A) &= \begin{cases} (1+\varepsilon) A & (A \geq 0) \\ (1-\varepsilon) A & (A < 0). \end{cases} \end{aligned} \tag{6.4}$$

With this modification, PPO interpolates between the new and old policy by ensuring that the probability ratio $r_t(\theta)$ remains close to 1. In (6.4), the function g restricts (or *clips*) the magnitude of increase in the objective function, regardless of whether the advantage is positive or negative. This prevents excessive changes in the new policy.

Another approach tested in the same paper [60] involves using an adaptive coefficient for the KL penalty term in the objective function (6.1). However, this approach resulted in weaker performance compared to the clipping approach.

When using actor-critic methods where parameters between policy and value function are shared, the objective should include the critic's error term as well. Moreover, adding an entropy term can improve the exploration. Considering all these terms, the objective function for PPO becomes the following:

$$\begin{aligned} &\underset{\theta}{\text{maximise}} \quad \mathbb{E}_t \left[L^{\text{clip}}(\theta) - c_1 L_t^{VF}(\theta) + c_2 H(\pi_\theta(.|s_t) \right] \\ &\text{where} \\ &\qquad L_t^{VF}(\theta) \quad = (V_\theta(s_t) - V_t^{\text{targ}})^2, \\ &\qquad H(\pi(.|s_t) = -\textstyle\sum_{a \in A} \pi(a|s_t) \log(\pi(a|s)). \end{aligned} \tag{6.5}$$

in which L_t^{VF} is the squared-error loss and H is the entropy of the policy.

In the game studied in this project, the trajectory length is fixed at 25 stages. However, the advantage estimator used in PPO (Proximal Policy Optimisation) typically focuses on fixed-size trajectory segments to estimate the advantage, which is unnecessary in our case.

6.2.2. Soft Actor-Critic

Soft Actor-Critic (SAC) is an off-policy, Q-learning-based actor-critic algorithm that combines elements of both value-based and policy-based reinforcement learning methods. The definitions and algorithm explanations in this section are adapted from the paper by Haarnoja, Zhou, Abbeel and Levine [23].

As an actor-critic method, SAC employs separate networks for the policy and value functions, which do not share parameters. SAC is an extension of the Deep Deterministic Policy Gradient (DDPG) [43] algorithm to the stochastic setting. This algorithm considers stochastic policies, hence the term *soft*, meaning that at each step, it provides a probability distribution over all actions instead of one deterministic action. The main objective of SAC is to maximise the overall expected return as well as the entropy of the stochastic policy, which enhances exploration.

$$\underset{\theta}{\text{maximise}} \quad \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[\gamma^t (r_t(s_t,a_t) + \alpha H(\pi_\theta(\cdot|s_t))) \right]. \tag{6.6}$$

In (6.6), $H(\pi_\theta)$ is the entropy of the policy (similar to (6.5)) and α is the entropy coefficient. While this coefficient can be omitted in theoretical proofs by scaling, it significantly affects the exploration of the policy in practice and should be carefully tuned. The entropy term in the objective also helps the algorithm avoid local optima and allows it to assign equal probabilities to multiple attractive actions. Adding the entropy term has been shown to significantly improve results in various algorithms [23, 58]. Another strong point of SAC is that it reuses previously collected data in a replay buffer for future updates. The replay buffer increases the sample efficiency of the algorithm since experiences can be

used in multiple updates. Moreover, sampling randomly from the replay buffer stabilises the algorithm by not overfitting to recent experiences.

The soft Bellman equations for the soft state values and soft Q-value functions, according to the objective function (6.6), are:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}[V(s_{t+1})] \tag{6.7}$$

$$V(s_t) = \mathbb{E}_{a_t \sim \pi}[Q(s_t, a_t) - \log \pi(a_t|s_t)] \tag{6.8}$$

In theory, it has been proved that repeated application of soft policy evaluation and soft policy improvement will converge to the optimal policy. However, for the continuous domain, the soft policy iteration must be approximated using function approximation techniques. In this project, we use the implementation in Stable-Baseline3 [52] and describe it below.

SAC learns a policy (the actor) and two Q-functions (the critics) with separate parameters θ, ϕ_1, ϕ_2. The two Q-functions are trained independently to estimate the action-value function. During training, a minimum of two Q-values is used to compute the value function target and the policy gradient update (we explained the Q-function learning in Section 4.1.4).

The primary reason for using two Q-functions is to reduce overestimation bias in the Q-value estimates, which has been shown to improve the stability and speed of training [18], [27].

Soft Actor Critic (SAC) minimises the mean squared Bellman error (MSBE) loss for each Q-function. Initially, for both Q networks, the target Q-function parameters are set equal to the Q-function parameters, $\phi_{\text{targ},i} = \phi_i,\ i = 1, 2$. At each update step, a batch of experiences $\mathbf{B}$ is randomly selected from the replay buffer. Each experience is of the form (s, a, r, s', d), where d is the done flag and $d = 1$ if s' is a terminal state, otherwise $d = 0$. For each Q-network i, the network parameters ϕ_i are updated to minimise the following loss function:

$$L_{Q_i}(\phi_i, \mathbf{B}) = \mathbb{E}_{\mathbf{B}}\left[\left(Q_{\phi_i}(s, a) - g(r, s', d)\right)^2\right]$$

$$g(r, s', d) = r + \gamma(1 - d)\left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

where g is the target value used for training the Q-functions, which combines the current reward, expected future Q-values (from the target Q-functions), and the entropy term to increase exploration. The

For the policy update, a reparametrisation trick is used to reduce the variance of the estimator. The policy parameters are then updated to minimise the following loss function:

$$L_\pi(\theta, \mathbf{B}) = \mathbb{E}_{\mathbf{B}}\left[\alpha \log \pi_\theta(\tilde{a}_\theta(s)|s) - \min_{j=1,2} Q_{\phi_j}(s, \tilde{a}_\theta(s))\right]$$

where $\tilde{a}_\theta(s)$ is a differentiable sample from $\pi_\theta(\cdot|s)$ with respect to θ.

Both Q-functions and policy parameters are updated by one-step gradient descent according to the corresponding loss functions.

Finally, Q-value networks are updated using Polyak averaging, which is an affine combination of the parameters of the target Q-function and the current Q-function:

$$\phi_{\text{targ},i} \leftarrow (1-\tau)\phi_{\text{targ},i} + \tau\phi_i \ , \quad i = 1, 2. \tag{6.9}$$

This updating rule limits the changes to the parameters and stabilises the training. The value of τ should be close to zero to ensure the new value does not deviate significantly from the old one. This parameter should be evaluated when tuning the hyperparameters, as we did in Table 6.4.

6.2.3. Comparison of the Learning Algorithms

The two algorithms we study in this chapter, PPO and SAC, differ in many fundamental characteristics, which is why we chose them. Our aim was to determine which model works best for the game at hand. Both of these models are model-free reinforcement learning algorithms, which means the learning is only experience-based.

PPO (Proximal Policy Optimisation) is a policy gradient method where the policy is directly optimised to maximise the advantage function. The main feature of this algorithm is bounding the large updates and not allowing large changes at each iteration, which makes the learning more stable. On the other hand, SAC (Soft Actor Critic) is an advanced Q-learning-based algorithm that approximates the Q-values and derives a stochastic policy from them. The main feature of SAC is the entropy regularisation term in the objective function, which encourages more exploration.

Additionally, PPO is an on-policy learning algorithm, meaning the agent learns from actions chosen based on the current policy (target policy). In contrast, SAC is an off-policy algorithm, where the agent learns from experiences generated by the old policy (behaviour policy) while updating Q-values based on samples from the current policy (target policy).

PPO is less sample efficient than SAC because each experience is used only once (or a limited number of times in batch learning) during the learning process. Meanwhile, SAC can reuse experiences multiple times through the replay buffer, resulting in more efficient learning. However, in terms of time efficiency, SAC is more computationally expensive and considerably more time-consuming than PPO, as SAC trains multiple networks.

6.3. Incorporating Advanced RL Algorithms into the PSRO Setting

In this section, we describe our initial experiments with new learning algorithms, the execution of the PSRO framework, the improvements made to the setting, and the resulting meta-games.

To run the PSRO experiments using the selected learning algorithms, we used the same configuration for the action and state spaces in both. The implementations of both PPO and SAC algorithms in Stable-Baselines3 are compatible with continuous action spaces, which was our primary goal from the outset. Continuous action spaces allow for the representation of all possible strategies in the pricing game. Therefore, within this RL framework, we switched to a continuous setting and adopted an action space over the range $[0, 60]$, which aligns with the range used in our previous experiments.

The state space (observation space) is consequently also defined as continuous, as it includes the demand potential and prices, which are continuous variables.

In the first experiments with SAC and PPO, we continued with the same state representation as in the previous chapter, which includes the one-hot encoding of the stage, the current demand potential of the player, and the opponent's price history of fixed length, which we call memory.

Another modification to our training process, which was facilitated using Stable-Baselines3 (SB3), is multi-processing. In SB3, for training a single agent, it is possible to define multiple parallel environments, and the agent takes steps in all these environments simultaneously. The experiences from all these environments are recorded, and the model is then updated using the combined experiences. Each of these environments runs on a different CPU process, which significantly speeds up the training process (not exactly time divided by the number of processes, but close to it) and allows us to utilise multiple processor cores. This considerably decreases training time and allows us to focus more on the meta-games.

The way experiences from these environments update the policy depends on the algorithm, primarily because SAC is an off-policy algorithm, whereas PPO is on-policy.

In on-policy algorithms, a fixed number of experiences (`n_steps` = 2048 by default) are collected from all environments. These experiences are then divided into mini-batches, and the neural networks are updated using multiple epochs of gradient descent on these batches.

Conversely, in off-policy algorithms, all experiences from all environments are added to a shared replay buffer. At each step, a mini-batch of experiences is sampled from the replay buffer, and the neural networks are updated using gradient descent on the mini-batch.

In the first experiments, we started two separate meta-games using SAC and PPO, both starting from the 3×3 starting games of the Myopic, Constant-132, and Guess-132 strategies for both low-cost and high-cost players. We defined the following parameter values, while the rest of the parameters were set to their default values as defined by the model:

- **Learning rate (lr)** = 3×10^{-5}: We experimented with lower values initially and reached this value through experience. This value was later fine-tuned again along with other parameters during hyperparameter tuning, described in Section 6.3.2.

- **Discount factor** (γ) = 1: This value was chosen based on the results tested in the previous chapter.
- **Memory** = 3: We continued with the same value as in previous experiments; however, this parameter was also tuned later.
- **Number of episodes** = $3 \times 10^6 \times$ |support(opponent's strategy)|: A large number of episodes was initially considered to observe the agent's behaviour. The number of episodes increases as the number of strategies in the opponent's mixed equilibrium strategy increases because the learning task becomes more complex as the agent needs to distinguish between different opponents.
- **Target entropy for SAC** = 0: We set this target value to zero to encourage the strategies to become less stochastic as learning progresses; in our pricing game, randomly chosen prices are empirically less likely to lead to cooperation between the firms (see also Section 6.4.2).

In Figure 6.3, we have plotted the mean return throughout learning for all strategies trained using the two algorithms in separate meta-games. It is important to point out that these strategies are not trained against the same opponents. Although both meta-games started from the same initial game, from the second iteration of the meta-game, the opponents differ because the new equilibria are based on the newly trained agents. Therefore, while we have plotted the mean return, this is not the primary aspect we aim to compare. Additionally, the number of episodes the strategies are trained differs depending on the opponent's support or the early termination of training.

What we intended to highlight with the plot was the behaviour of these strategies during learning. As shown, the learning path of the SAC algorithm is very noisy, and the returns do not stabilise completely, even after millions of episodes. This is due to the entropy term in the SAC algorithm, which is one of its strengths as it encourages exploration. While setting the target entropy to zero reduces the variance in the final model, the resulting policy is still not a deterministic strategy.

The PPO-trained strategies are significantly less random compared to those trained by SAC, and the learning plot appears smoother because PPO employs bounded updates, which avoid sudden policy changes. The changes throughout the learning process for PPO are more gradual, resulting in smoother learning paths.

The policy trained using SAC updates more frequently, with the entropy term enhancing exploration and no external bound on the updates. We believe that the combination of these factors is the reason behind the sudden improvements observed in strategies trained using SAC.

There are 32 strategies plotted for PPO, 16 for each player, as we trained the PPO meta-game for 16 rounds, and 24 strategies, 12 for each player, trained using SAC as we trained SAC for 12 rounds. Among the PPO strategies, 3 low-cost and 11 high-cost strategies were successful against corresponding equilibria to be added to the meta-game. For SAC, 7 low-cost and 6 high-cost strategies were added to the meta-game.

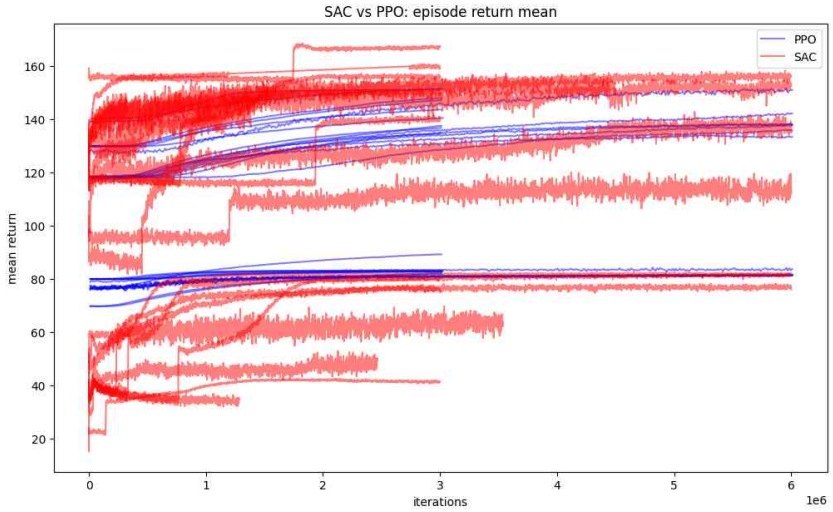


Figure 6.3.: Plot of mean returns of strategies in the two separate meta-games trained using SAC and PPO algorithms

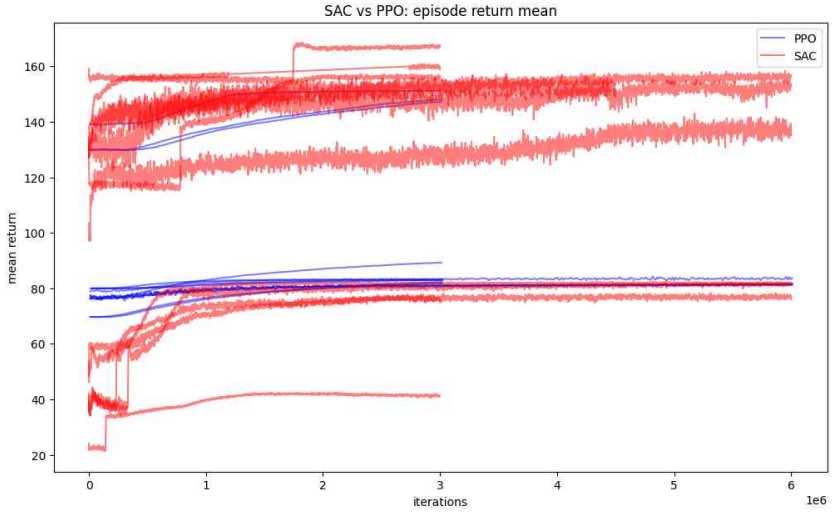


Figure 6.4.: Plot of mean returns of **successful** strategies in the two separate meta-games trained using SAC and PPO algorithms

In Figure 6.4, we have plotted only the successful strategies. We cannot compare the strategies directly because of the different opponents they have. However, what we could observe in these plots, which also appeared in other similar experiments we ran, was that the high-cost strategies trained using PPO in the PPO meta-game have higher returns than high-cost strategies trained in the SAC meta-game. In contrast, the higher return low-cost strategies were trained in the SAC meta-game. We still cannot conclude anything at this stage. There, in the next experiments, we put the two algorithms in the same meta-game and train strategies using both algorithms against the same opponent.

However, since the initial game for both experiments is the same 3×3 game, we can compare the strategies trained in the first iteration of the experiments. The only equilibrium of the initial game, as can be seen in the bimatrix game (6.10) in Section 6.3.4 below, is Guess-132 for the low-cost player and the Myopic strategy for the high-cost player. This

means in both experiments in the first iteration of the meta-game, a low-cost agent is trained against the high-cost Myopic strategy, and a high-cost agent is trained against the Guess-132 low-cost strategy.

We have plotted the prices, demand potential, and actions of these trained agents (one low-cost and one high-cost agent for each learning algorithm) over 200 trials of the pricing game in Figure 6.5. In the first plot, we have shown the returns of these agents against their respective opponents. For example, for the low-cost SAC agent, the opponent is the high-cost myopic strategy, and the agent's return is shown on the x-axis, with the myopic strategy's return on the y-axis. Similarly, for the high-cost SAC agent, the opponent is the low-cost Guess-132 strategy. The agent's return is plotted on the y-axis because the agent is the high-cost player in the game, and the x-axis shows the Guess-132 strategy's return. In each of the plots, we have highlighted the path following a greedy strategy (i.e., taking the action with the highest probability at each stage) with a bolder line.

Among the 4 trained agents plotted, the PPO low-cost strategy was not successful against the equilibrium strategy (myopic strategy) and was not added to the PPO meta-game. The remaining strategies were successful. As is very clear from the plots, the variance of the final probability distribution over actions learned by SAC is significantly higher than PPO, resulting in much noisier paths.

When we compare the low-cost strategies, we observe that while SAC is noisier, it consistently led to higher returns than PPO across all trials. It is important to point out that although the price paths of SAC fluctuate a lot, the greedy path does not show those fluctuations. The strong point of the SAC low-cost strategy can be seen in the demand potential plot. SAC-trained strategies reach considerably higher demand potential than PPO. This means SAC can better explore the advantage that low-cost strategies have, being able to price lower, compared to PPO. The PPO-trained strategy, in contrast, leads to lower returns than the Guess-132 low-cost strategy and is therefore not added to the game.

In contrast, the two algorithms show very similar behaviour for high-cost strategies. Although the PPO high-cost strategy has slightly higher returns, the price path, actions, and demand potential sequence of the two models are quite similar.

In terms of the end-effect, there is no visible sign of end-effect in PPO-trained strategies. However, SAC-trained strategies, in some trials, play closer to the myopic price in the last stage, though this behaviour is inconsistent.

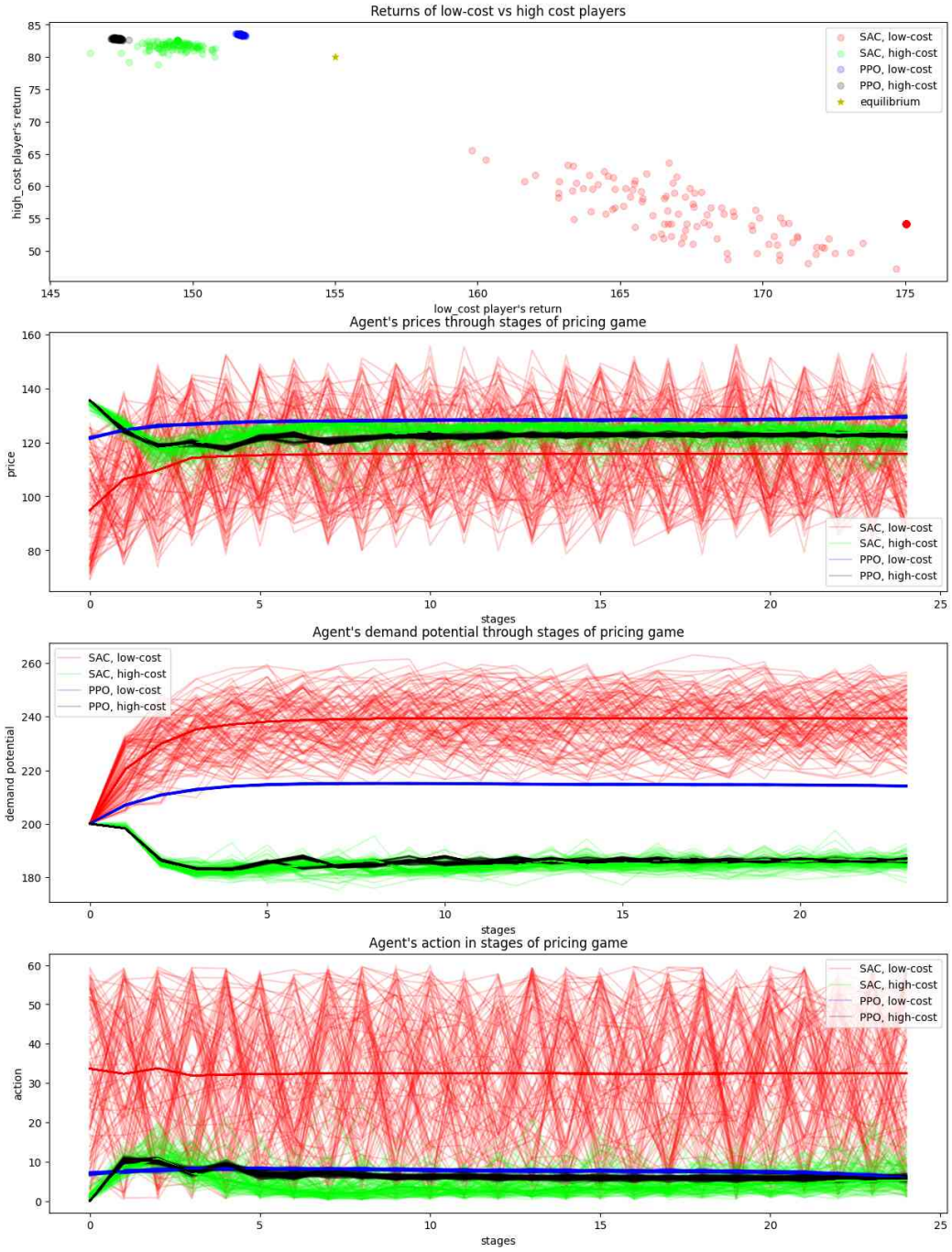


Figure 6.5.: Plot of mean returns of strategies in the two separate meta-games trained using SAC and PPO algorithms

6.3.1. Integrating PPO and SAC into a Unified Meta-Game

In the previous section, we suspected that SAC is more successful in training agents (particularly low-cost agents), and PPO shows better performance with high-cost agents. However, the first iteration of meta-games and the final games are insufficient to draw definitive conclusions. Therefore, we initiated an experiment to test the learning rate, memory length, and learning algorithms that we believed required further investigation. As

the meta-game progressed, we adjusted the parameters based on the results obtained and improved the model iteratively.

Since multiple parameters were tested and, for each combination, an agent was trained at every iteration of the meta-game, this process was time-consuming. To facilitate and accelerate our experiments, we used the Fabian high-performance computing facility at the London School of Economics [17]. Fabian provides access to multiple high-speed processors and a robust platform for running large-scale computational tasks, which was essential for managing the computational demands of our experiments.

MCG-FREQ

We refer to the first experiment as MCG-FREQ because it started from the 3×3 MCG game shown in (6.10), and the equilibrium selection method used was the most frequent equilibrium found via the tracing procedure.

We made the following modifications to this experiment compared to the previous one:

- **State representation:** In the previous representation of the state, we included the memory of opponent's prices. As we discussed, this memory is essential for the agent to distinguish between different opponents. However, in the earlier models, the agent could only recall its own most recent price and not any prior prices. At this stage of the project, we decided to include the same memory length for the agent's own prices as was used for the opponent's prices. This information was added to the state because it provides a clearer context to the agent regarding its position in the pricing game and the prices it has previously played, which could influence the opponent's actions.

one-hot encoding of stage	demand potential	agent's price memory	opponent's price memory

- **Base agent:** Training an agent from scratch with randomly initialised parameters takes a long time, and repeating this process for each agent separately is very time-consuming. However, the previously trained agent can be used to initialise the model parameters, and learning starts from there while adapting based on new experiences. This approach is known as *warm starting*. Intuitively, since the agent has already learned the approximate magnitude of returns and has an understanding of the observation space, it can focus more effectively on dealing with the specific opponent in that iteration. This reduces training time and improves learning efficiency. Hence, when training new agents in the meta-game, we search among previously trained agents that used the same learning algorithm, the same memory length in the observation space, and the same production cost. Among these agents, we prioritise those that have faced the same opponents during their training, which provides a head start for our new training process.
- **The test parameters:** We wanted to test which learning algorithm trains more successful agents in the meta-game to use in future experiments. Moreover, we considered different values for the learning rate and the length of memory in the

state representation. We hypothesised that longer memory could help agents learn better, so we tested the trade-off between the added complexity of a larger state representation and the efficiency of learning.

- **Database:** The changes mentioned to the models required a structured and efficient database. This database allowed agents to utilise previous training results to identify the best base agent for warm starting and enabled us to analyse the most effective tested parameters on learning performance, as all data related to agents, trials, and equilibria are recorded in detail.

We ran the MCG-FREQ experiment for 21 iterations. In each iteration of the meta-game, 16 agents were trained: 8 low-cost and 8 high-cost agents, against the equilibrium found most frequently using the tracing procedure. These agents were trained with different combinations of learning algorithms (PPO or SAC), memory lengths (3 or 12), and learning rates (0.0003 or 0.00016), resulting in 8 combinations.

In Table 6.1, the proportion of successful strategies is calculated for each combination. Low-cost strategies are specified in the rows with 'L' and high-cost strategies with 'H'. The columns represent the tuple of (memory, learning rate).

	(3, 0.0003)	(12, 0.0003)	(3, 0.00016)	(12, 0.00016)
SAC, L	$\frac{4}{21} = 19\%$	$\frac{10}{21} = 48\%$	$\frac{3}{21} = 14\%$	$\frac{9}{21} = 43\%$
SAC, H	$\frac{4}{21} = 19\%$	$\frac{11}{21} = 52\%$	$\frac{4}{21} = 19\%$	$\frac{9}{21} = 43\%$
PPO, L	$\frac{2}{21} = 9\%$	$\frac{0}{21} = 0\%$	$\frac{2}{21} = 9\%$	$\frac{0}{21} = 0\%$
PPO, H	$\frac{2}{21} = 9\%$	$\frac{1}{21} = 5\%$	$\frac{2}{21} = 9\%$	$\frac{1}{21} = 5\%$

Table 6.1.: Proportion of successful strategies in the MCG-FREQ experiment. The rows represent the algorithm followed by 'L' (low-cost) or 'H' (high-cost). The columns represent the tuple of (memory, learning rate).

As can be seen, SAC trains significantly more successful strategies compared to PPO, both for low-cost and high-cost strategies. As the meta-game progresses, it becomes increasingly harder for PPO to train successful agents against the equilibria. Although PPO appears to perform slightly better for training high-cost strategies, the difference is minimal. Among high-cost strategies, only $\frac{6}{28} = 21\%$ of successful high-cost strategies are trained using PPO. Similarly, the proportion of successful low-cost strategies trained using PPO is only $\frac{4}{26} = 15\%$.

Another important observation from these results is that strategies trained with longer memory are considerably more successful than those with shorter memory. Regarding the learning rate, the differences in performance are not significant enough to draw conclusions, although a learning rate of 0.0003 seems to be slightly more effective.

After reviewing these results, we decided to continue the experiment without PPO to save computational costs, as most of the agents trained using PPO in later iterations were not added to the game.

Furthermore, observing the advantage of longer memory motivated us to test an even longer memory length of 18 (replacing the shorter length of 3). It is important to note that increasing the memory length increases the state representation length by twice the increase, as it involves additional price memory for both the agent and the opponent. This not only increases computational costs but could also slow down learning, as the agent must analyse more features. Hence, we prefer shorter memory lengths when the results are comparable.

We continued the meta-game from the last stopping point for additional iterations, incorporating the new parameters. Table 6.2 shows the proportions of successful strategies using the updated parameters during these new iterations. It is worth noting that the number of trained strategies is not consistent across sets of test parameters because the game was paused multiple times, and some iterations were restarted. As a result, some trained agents were added to the meta-game while others did not have the opportunity to be evaluated within the same iteration.

	(18, 0.0003)	(12, 0.0003)	(18, 0.00016)	(12, 0.00016)
SAC, L	$\frac{9}{27} = 33\%$	$\frac{14}{32} = 44\%$	$\frac{10}{27} = 37\%$	$\frac{9}{27} = 33\%$
SAC, H	$\frac{12}{26} = 46\%$	$\frac{10}{27} = 37\%$	$\frac{14}{18} = 78\%$	$\frac{7}{19} = 37\%$

Table 6.2.: Proportion of successful strategies in the MCG-FREQ experiment with adjustments in parameters. The rows represent the algorithm followed by 'L' (low-cost) or 'H' (high-cost). The columns represent the pairs (memory, learning rate).

These results appeared to be inconclusive as they did not provide a clear indication of the optimal parameter set. While a memory length of 12 led to more successful strategies for the low-cost agent, the opposite was observed for the high-cost agent, where a memory length of 18 produced better outcomes. Given these conflicting findings, we could not draw conclusions from this experiment. Consequently, we decided to conduct a comprehensive hyperparameter tuning experiment, examining all relevant parameters for each algorithm, as detailed in the next section.

Before moving to hyperparameter tuning, we plotted the strategies trained in the first iteration of the meta-game with a learning rate of 0.0003 and memory length of 3 in Figure 6.6. This figure can be compared to the previous plot in Figure 6.5, where the memory length was also 3. The fact that agents were trained in separate meta-games during the previous experiment does not affect the results since we are only analysing the first iteration where agents face deterministic strategies.

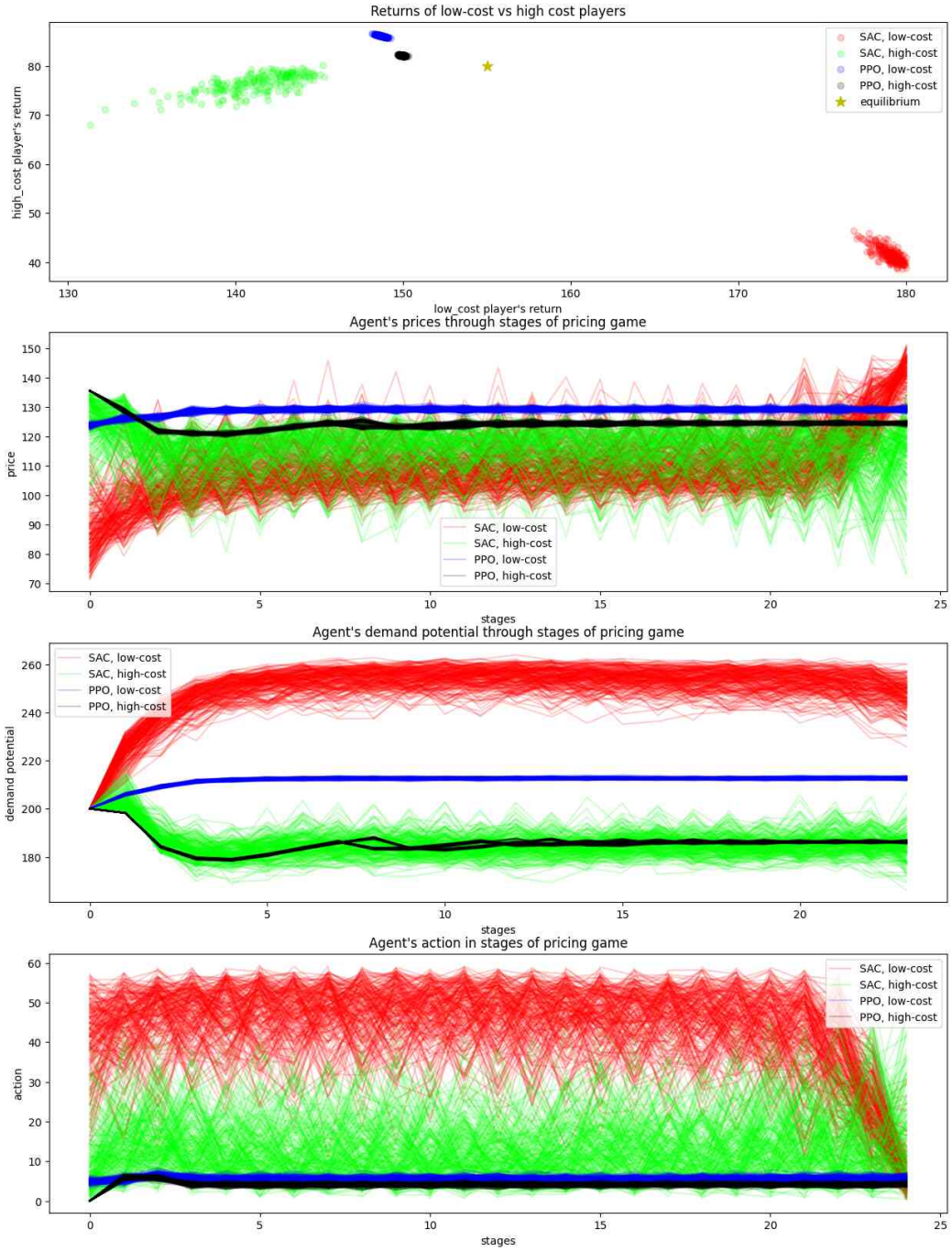


Figure 6.6.: Plot of mean returns, prices, demand potential, and actions of strategies trained in the first iteration of experiment MCG-FREQ, with lr=0.0003 and memory=3.

The plotted agents were trained for 2×10^6 episodes if they started learning from random network parameters, and for 8×10^5 episodes if initialised from previously trained agents.

By comparing the two plots, we observe that changes in the structure of the state, particularly the inclusion of memory of the agent's own previous prices, had the most significant effect on the low-cost SAC agent. This agent learned the end-effect of the game, as is evident from the action trending toward zero in the final stage. Furthermore, in the current experiment, the agents were trained for more episodes than in the previous

experiment, which helped stabilise the strategies. These changes were applied and improved as described in the next section.

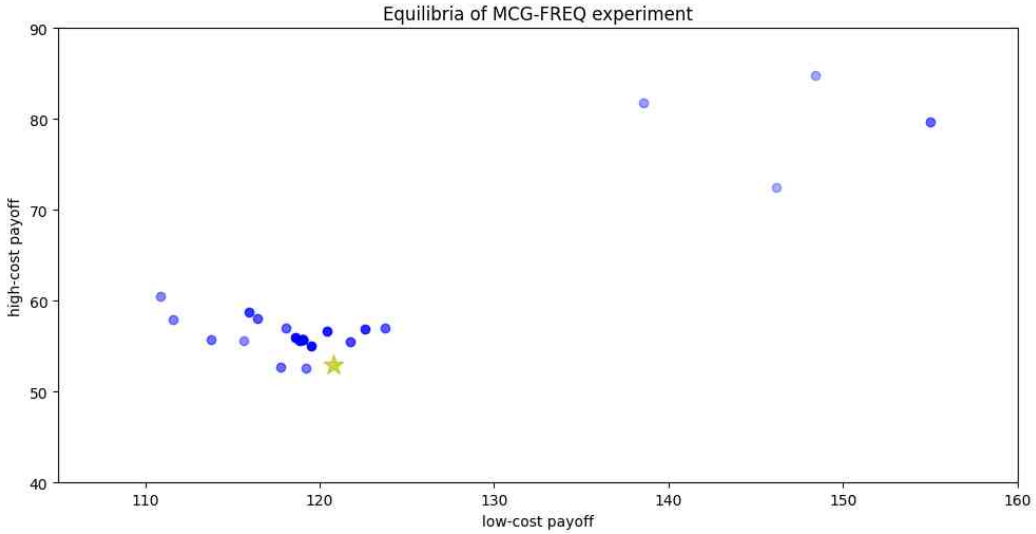


Figure 6.7.: Plot of payoff pairs of equilibria in the MCG-FREQ experiments. The yellow star is the payoff pair of the SPE.

Finally, we have plotted the equilibria of the meta-game in the early iterations of the MCG-FREQ experiment in Figure 6.7. This plot will later be compared with the results from our final experiments, which empirically identify a new mixed equilibrium.

6.3.2. Hyperparameter Tuning of the Learning Algorithms

Different choices for hyperparameter values can significantly impact the convergence rate, as well as the exploration and exploitation strategies of a learning algorithm, and can lead to either improvements or deteriorations in performance. In previous experiments, we tuned basic parameters such as the learning rate and discount factor. However, with the adoption of more advanced RL algorithms, we recognised the need to tune all the hyperparameters of these algorithms together to achieve the best performance for our specific environment setting.

To assess the impact of different parameters more carefully, we conducted multiple tests considering various factors, such as the average total reward for each strategy, the variance of prices over different iterations, and the running time.

In the first test, we investigated the structure of the state representation, specifically varying the length of memory provided in the state to include past prices of both the agent and the opponent. Memory provides important information in the learning process, especially since our agents compete against different opponents. For agents to effectively distinguish between different opponents and adapt their strategies accordingly, they need to retain information about previous interactions. The results presented in Table 6.1 highlighted the significant effect of this parameter on performance and motivated us to study it more carefully.

The optimal length of memory should strike a balance between providing sufficient information for the agent to learn the opponent's patterns and not excessively increasing the algorithm's complexity. To evaluate this, we trained three low-cost agents against the mixed strategy ($\frac{1}{2}$ Myopic, $\frac{1}{2}$ Guess-132) for each value of `memory` between 0 and 24. When `memory` = 0, agents lack any information about their own previous prices as well as those of their opponents. Conversely, when `memory` = 24, the agent is aware of all prices played by both itself and the opponent in all previous stages.

We compared the performance in terms of running time, standard deviation of returns, and return mean. Although memory values in the range 0 to 5 made a significant difference in the mean and standard deviation of returns, beyond that point, running time became the primary difference between the results. The values in the range 9 to 15 produced similar results across all criteria. Taking into account that the opponent's our agent face in future might be more complicated than the test opponent (higher support for opponent's strategy), we chose `memory` = 12 as the acceptable value for the length of memory in the state representation.

In the next test, we aimed to restrict the candidates for the learning rate to be considered in combination with other hyperparameters. We trained low-cost agents with different values of the learning rate (lr) against the mixed strategy of ($\frac{1}{2}$ Myopic, $\frac{1}{2}$ Guess-132). To assess the impact of the learning rate, we trained three agents for each of the 20 equally spaced values in the range $lr \in [0.00001, 0.00058]$, each over 4 million episodes.

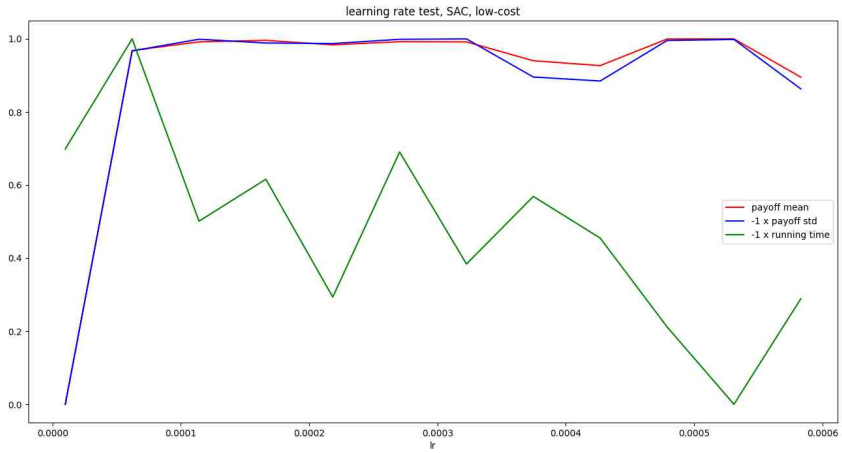


Figure 6.8.: Payoff, $-1\times$ standard deviation and $-1\times$ running time for different learning rates are plotted for low-cost agents trained by SAC.

We aim to find the values that maximise payoff while minimising variance and running time. In Figure 6.8, normalised values of the mean payoff, $-1\times$ payoff standard deviation, and $-1\times$ running time are plotted. All agents were trained using the SAC algorithm. By analysing this plot and examining the variance in the agents' prices and demand potentials over the episodes, we selected $lr = 0.000298 \approx 0.0003$ as the optimal learning rate parameter. The demand potential and price for this value of lr are plotted in Figure 6.9.

The next best candidate value to consider, in combination with other hyperparameters, was 0.00016.

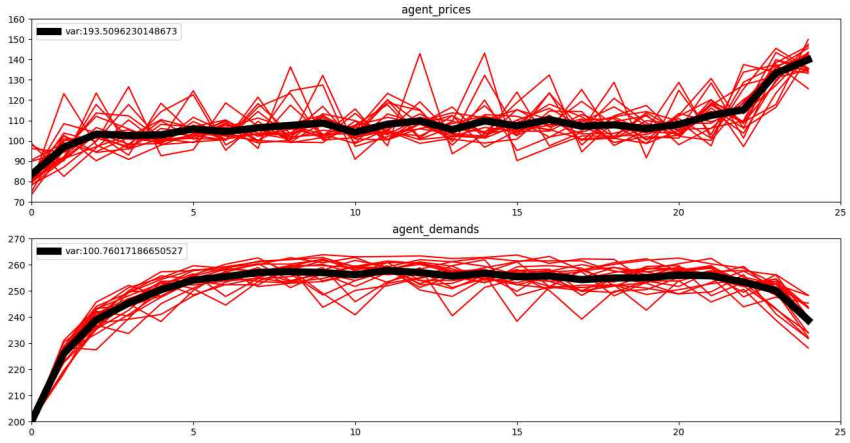


Figure 6.9.: Prices and demands of the trained agent through the 25-round game are plotted for different iterations. In this plot, $lr = 0.000298$.

The most important test, which had a crucial effect on our results in terms of convergence and optimality, was a comprehensive grid search across all hyperparameters of the two learning algorithms. Before conducting this test, adjusting basic hyperparameters that we assumed to be the most effective did not yield an acceptable variance in the final model trained using the SAC algorithm. Additionally, the PPO algorithm produced weaker results in terms of optimality compared to the SAC algorithm.

To address this, we performed the grid search on the respective parameters of both the PPO and SAC algorithms. Given the large number of parameters and the multiple values to test for each, using the main game for this evaluation was impractical due to time constraints. Instead, we decided to run the grid search on a smaller version of the game with 4 stages, while keeping the rest of the game details the same. Since the length of memory in the 25-stage main game was set to 12, we adjusted the memory length to 2 for the 4-stage game (half the length of the episode).

For each algorithm, we examined the parameters and values explained in Tables 6.3 and 6.4, testing each combination for three iterations. In all these tests, the discount factor is considered to be one.

To understand the parameters mentioned in Tables 6.3 and 6.4, we need to explain how they are used.

- **PPO:** Each step taken in the environment, in our case every stage of the pricing game, is called an *experience*. An *episode* is a sequence of experiences from the initial state to the final state; in our case, 25 stages form one episode. A large number of these experiences, from multiple episodes, are collected in each *batch*, which will be used to update the policy network. In each *epoch*, the batch is shuffled randomly, and then it will be divided into *mini batches* since the batch size is large. The division

parameter	**values**	**description**
lr	$\{0.0003, 0.00016\}$	learning rate
n_epochs	$\{10, 20\}$	number of epochs (passes through batch)
clip_range	$\{0.2, 0.1, 0.3\}$	range used in clip loss in objective, ε in 6.4
clip_range_vf	$\{\text{None}, 0.2\}$	clip range for value function. If None, the value function will not be clipped.
ent_coef	$\{0.0, 0.01, 0.001\}$	the coefficient of entropy loss in the objective, c_2 in (6.5)
vf_coef	$\{0.5, 0.4, 0.6\}$	the coefficient of value function loss in the objective, c_1 in (6.5)

Table 6.3.: Hyperparameters examined for PPO

parameter	**values**	**description**
lr	$\{0.0003, 0.00016\}$	the learning rate for Adam optimiser, used for all networks
target_entropy	{'auto', 0}	target entropy when learning entropy coefficient
ent_coef	{ 'auto', 'auto_0.1', 'auto_0.001'}	Entropy regularisation coefficient, α in (6.6), 'auto' for the value to be learned automatically, 'auto_0.1' means use 0.1 as initial value
tau	$\{0.005, 0.001, 0.01\}$	Polyak update coefficient, τ in (6.9)
train_freq	$\{1, 10, 20\}$	after how many steps should the model be updated
gradient_steps	$\{1, -1\}$	how many gradient steps to do after each experience. '-1' means as many steps as taken in the environment

Table 6.4.: Hyperparameters examined for SAC

helps use computational resources more efficiently and leads to stable learning. Each mini-batch is used once to update the policy network within each epoch.

- **SAC:** At the start, the agent interacts with the environment for *train_freq* steps using the current policy. These experiences are collected and added to the replay buffer. Then, a batch of experiences is sampled from the replay buffer. The Q-networks and policy network are updated *gradient_steps* times using the Adam optimiser with the *lr* learning rate. The coefficient of the entropy term in (6.6) is learned according to the *ent_coef* and *target_entropy* settings. At the end of each update cycle, the target values are updated using Polyak averaging with the parameter *tau*. This process gets repeated.

After training three agents with each combination of hyperparameters, we evaluated their performance using the following metrics over 100 trials:

- Mean of cumulative rewards, *return_mean*,
- Mean of the three agents' standard deviations, *return_std_mean*,
- Standard deviation of the three agents' standard deviations (to measure how different the behaviour of the three agents was), *return_std_std*,
- Running time, *time_mean*.

The top combinations with acceptable mean return and mean standard deviation of final models, for low-cost agents and high-cost agents, are plotted in Tables 6.5 and 6.6.

PPO, low-cost

lr	n_epochs	clip_range	clip_range_vf	ent_coef	vf_coef	return_mean	return_std_mean	return_std_std	time_mean
0.00016	10	0.3	None	0.010	0.5	22.950655	0.026702	0.012418	111.125
0.00016	10	0.3	None	0.010	0.5	23.019558	0.035856	0.014384	71.125
0.00016	20	0.3	None	0.010	0.5	22.863185	0.020626	0.007057	112.250
0.00016	20	0.3	None	0.001	0.5	22.809958	0.013811	0.005614	112.750
0.00016	20	0.3	None	0.001	0.4	22.821347	0.016700	0.007731	112.625

(a) Parameters with best performance for training low-cost agents using the PPO algorithm

PPO, high-cost

lr	n_epochs	clip_range	clip_range_vf	ent_coef	vf_coef	return_mean	return_std_mean	return_std_std	time_mean
0.00030	10	0.3	None	0.010	0.5	14.424052	0.007342	0.002926	71.750
0.00016	10	0.3	None	0.010	0.4	14.510688	0.005696	0.003224	71.500
0.00016	10	0.3	None	0.010	0.6	14.508981	0.005758	0.002920	71.250
0.00016	10	0.3	None	0.001	0.5	14.493045	0.007316	0.003973	71.625
0.00016	10	0.3	None	0.001	0.6	14.510556	0.004996	0.002593	71.250

(b) Parameters with best performance for training high-cost agents using the PPO algorithm

Table 6.5.: Results of hyperparameter tests for both low-cost and high-cost agents using PPO

Based on these results, for each of the algorithms, we chose a combination of parameters that performs well for training both low-cost and high-cost agents. The final selected hyperparameters for each algorithm are listed in Table 6.7. For the subsequent experiments, we used these parameters, which led to more stable and optimal results in the trained models.

After finalising the best choice of all hyperparameters, we turned our attention to two additional assumptions from the previous models: the equilibrium selection method in the meta-game and the initial meta-game configuration.

6.3.3. Equilibrium Selection in PSRO

One of the challenging steps in implementing the PSRO framework is selecting the equilibrium that new agents are trained against. At each step of the meta-game, one or more equilibria of the current game are computed, and then the new agents are trained against these equilibrium strategies. The new agents are added to the game if they achieve a higher

SAC, low-cost

lr	target_entropy	ent_coef	tau	train_freq	gradient_steps	return_mean	return_std_mean	return_std_std	time_mean
0.0003	auto	auto	0.010	1	1	23.517860	0.108013	0.012151	497.625
0.0003	auto	auto	0.010	1	-1	23.517860	0.108013	0.012151	496.000
0.0003	auto	auto_0.1	0.005	20	-1	23.476461	0.105596	0.028992	471.500
0.0003	auto	auto_0.1	0.010	1	1	23.452549	0.104834	0.021052	491.125
0.0003	auto	auto_0.1	0.010	1	-1	23.452549	0.104834	0.021052	491.250

(a) Parameters with best performance for training low-cost agents using the SAC algorithm

SAC, high-cost

lr	target_entropy	ent_coef	tau	train_freq	gradient_steps	return_mean	return_std_mean	return_std_std	time_mean
0.00030	auto	auto	0.001	10	-1	14.442437	0.068717	0.022465	497.750
0.00030	auto	auto	0.010	1	1	14.444639	0.064042	0.017620	498.000
0.00030	auto	auto	0.010	1	-1	14.444639	0.064042	0.017620	499.375
0.00030	auto	auto_0.1	0.005	20	-1	14.439821	0.063326	0.009176	474.875
0.00030	auto	auto_0.1	0.010	20	-1	14.446415	0.059221	0.014173	476.250
0.00016	auto	auto_0.1	0.010	10	-1	14.379490	0.108330	0.057499	495.125

(b) Parameters with best performance for training high-cost agents using the SAC algorithm

Table 6.6.: Results of hyperparameter tests for both low-cost and high-cost agents using SAC

PPO: parameter	**value**	**SAC: parameter**	**value**
lr	0.00016	lr	0.0003
n_epochs	10	target_entropy	'auto'
clip_range	0.3	ent_coef	'auto'
clip_range_vf	None	tau	0.01
ent_coef	0.01	train_freq	1
vf_coef	0.5	gradient_steps	1
		buffer_size	200,000

Table 6.7.: Selected hyperparameters for SAC and PPO

return than the expected payoff of the corresponding equilibrium. In our experiments, we observed that the choice of equilibrium can significantly influence our results.

The first step is to compute the equilibria of the game. The meta-game is a bimatrix game, and all equilibrium vertices of bimatrix games can be computed using the *lrsNash* algorithm by Avis, Rosenberg, Savani and von Stengel [3]. However, the time complexity of this algorithm is exponential, making it impractical for games larger than approximately 20×20. As we expected our meta-games to scale to higher dimensions—indeed, the size of the meta-games in our experiments exceeds 80×80—the use of *lrsNash* was not an option. In addition, a full (possibly exponentially long) list of all Nash equilibria would give no information on which equilibrium is more likely to be played by the agents.

Instead, we employed the *tracing procedure* first proposed by Harsanyi [26], usually called the "Harsanyi-Selten" tracing procedure, as it is the basis of the equilibrium selection

method in their book [25]. An algorithm to implement it was proposed by van den Elzen and Talman [65], and shown by von Stengel, van den Elzen and Talman [68] to be a special case of the algorithm by Lemke [41]. This version of the algorithm was implemented in Python by my co-author Bernhard von Stengel and used in our experiments.

The tracing procedure is a Bayesian approach that begins with a mixed strategy pair, called the *prior*, that represents a commonly known initial belief of the players of what both players may possibly play. For example, the prior may be the uniform probability over each player's pure strategies (as assumed in [25]), or a pair of *random* points in each mixed strategy simplex, as is used in our case. The tracing procedure then follows a path of equilibria in a parametrised game where the players best respond against a mixture of the prior and the actually played mixed strategy pair. The parameter is the weight of the prior, which changes from initially 1 to 0, with possibly intermittent increases. When the prior weight has reached zero, an equilibrium of the game is found.

The tracing procedure thus generates a piecewise linear path of mixed-strategy pairs. The corresponding payoffs interpolate between the prior-based payoffs and the actual game payoffs, starting with the prior payoffs and ending with those in the actual payoff matrices. Along this path, each player best responds to their belief about the other players' strategies. Their beliefs keep getting updated based on their prior and the current equilibrium strategy. This continues iteratively until the end of the path, where they reach an equilibrium of the game.

The tracing procedure finds only equilibria that, in generic games, have a positive index. This is a necessary condition for an equilibrium to be locally dynamically stable [30], meaning that dynamics that are compatible with best responses (such as the replicator dynamics) bring slight deviations from the equilibrium back to the equilibrium. Generic games have one more equilibrium of index +1 than of index -1. The latter are never dynamically stable [30], and it is therefore good that the equilibria found by the tracing procedure and considered for training our agents have always index +1.

In each iteration of the meta-game, we run the tracing procedure with 100 traces (random priors) to find the equilibria of the current game and consider the set of equilibrium points found by these traces as our candidates for training the new agents. Finding these 100 equilibria did not take longer than a few minutes of computation time, so the PPAD-hardness of finding one equilibrium in a bimatrix game was no issue at all. In comparison, the training of agents was the main time requirement. An additional storage requirement arises when the bimatrix meta-game is to be generated by playing the trained agents against each other, because loading all their parameters and neural networks requires high RAM (as future work, there are ways to optimise memory management here).

The first selection method we considered for choosing the equilibria of the meta-game was based on their higher empirical frequency in the tracing procedure, as they are found starting from random priors. This approach prioritises equilibria with more traces leading to them, making them more plausible as an equilibrium choice. We used this method in all previous models, which we specified as the FREQ equilibrium selection.

In later experiments, an interesting question arose: Could agents learn to cooperate and converge to equilibria with higher returns? To explore this, we introduced an (in general different) selection method that chooses the equilibrium with maximum social welfare, defined as the equilibrium (as found by the tracing procedure) with the highest sum of payoffs for both players. This approach encourages cooperation, which benefits both players. We specified these experiments as WELF. The experiments in the next section adopt this selection method.

6.3.4. Initial Meta-Game

In the next experiments, we investigate the effect of different initial games on the progression of the meta-game. The strategies in the initial game, by appearing in various equilibria and subsequently training agents against them, can introduce different levels of complexity to the agents we train. In our experiments, we considered four different initial games, each considering distinct sets of deterministic strategies to start. These strategies are different in their level of complexity and cooperation.

Myopic-Constant-Guess (MCG)

Our first idea for the initial game, as previously used in earlier experiments, was a 3×3 game with the Myopic, Constant-132, and Guess-132 strategies (as defined in Section 4.3.2) for both low-cost and high-cost agents.

The Myopic strategy is a shortsighted, highly cooperative strategy that only considers the current stage's payoff. The Constant strategy, with its clear price sequence, should be relatively easy for the agents to exploit and achieve a high payoff against. Finally, the Guess strategy allows for cooperation but may not be easy to play optimally against.

This selection represents a variety of behaviours that we hope will enable our agents to better understand and adapt to the game settings.

low-cost \ high-cost	myopic	constant-132	guess-132
myopic	93.895 139.591	94.732 138.555	94.732 138.555
constant-132	94.760 138.527	103.712 127.512	103.712 127.512
guess-132	79.666 154.998	68.729 168.709	79.091 141.639

(6.10)

Using this starting mode, the game in 6.10 serves as the initial meta-game. The equilibrium identified by the tracing procedure is (Guess-132, Myopic), so next, low-cost agents will be trained against the Myopic strategy, and high-cost agents will be trained against the Guess-132 strategy.

It is worth mentioning that Constant-132 is a dominated strategy for the low-cost player, but it is only weakly dominated for the high-cost player. We intentionally kept this strategy in the game to study its impact on future performance in the PSRO process.

Random (RND)

In the implementation of the PSRO method, a common approach is to start the game with a strategy that plays randomly. Starting from a random strategy allows our model to begin learning without any prior knowledge, exposing opponents to various behaviours and training them against a diverse set of strategies. However, while a random strategy may initially increase exploration of the game, it is inherently unstable. As a result, agents struggle to learn effectively from these experiences, as they are constantly being pulled in different directions, leading to limited meaningful progress. We implemented this starting method by initialising the learning model with random parameters for the neural networks.

Multiple Guess (GUESS)

Another initial game we considered in our latest experiments started from variations of the Guess strategy. As mentioned earlier, Guess is a smart strategy that is not easy to exploit. Therefore, we decided to study the game that begins with the 3×3 bimatrix game, using the strategies Guess, Guess2, and Guess3 as both low-cost and high-cost strategies. These strategies are defined in Section 4.3.2.

This initial bimatrix game is shown in (6.11). The equilibrium (Guess3, Guess3) is the only one found by the tracing procedure for this game, so the low-cost and high-cost strategies will be trained against Guess3 in the next step.

low-cost \ high-cost	guess	guess2	guess3
guess	79.091 141.639	77.183 138.955	79.652 142.781
guess2	76.650 140.063	74.996 136.919	76.891 140.939
guess3	80.489 142.293	78.360 139.466	80.958 143.336

(6.11)

All and SPE (ASPE)

The last initial game mode we considered involved adding all our predefined strategies to the initial game, as well as the subgame perfect strategy, to study how the game proceeds.

We defined the initial game with the strategies SPE, Myopic, Constant-132, Imitation-132, Guess, Guess2, and Guess3 for both high-cost and low-cost agents.

low-cost \ high-cost	spe	myopic	const-132	imit-132	guess	guess2	guess3
spe	53 121	39 181	10 233	52 115	50 112	50 111	50 112
myopic	121 87	94 140	95 139	101 130	95 139	95 139	95 139
constant-132	184 23	95 139	104 128	104 128	104 128	104 128	101 131
imitation-132	97 104	102 130	104 128	104 128	104 128	104 128	103 128
guess	66 116	80 155	69 169	72 130	79 142	77 139	80 143
guess2	65 117	74 161	55 186	71 130	77 140	75 137	77 141
guess3	66 116	80 154	73 163	72 130	80 142	78 139	81 143

(6.12)

The bimatrix game (6.12) represents the initial game (with rounded payoffs for presentation purposes). The tracing procedure finds two equilibria for this game. Clearly (SPE, SPE) is one of these equilibria (where SPE stands for each player's strategy in the subgame-perfect equilibrium). The other equilibrium, which is essential for the extension of the meta-game, is (Guess3, Guess3). We explain next that no strategies could be added to the game when trained against (SPE, SPE), so in the first round, (Guess3, Guess3) is chosen to expand the meta-game.

Why not start from only SPE?

The idea of starting the game from the subgame perfect equilibrium (SPE) strategies is interesting to study. However, this is not feasible because SPE is the optimal strategy against itself. By definition, using backward induction, at each stage players maximise their payoffs, considering that the opponent does the same. Therefore, when both players use this strategy, neither can improve their payoff in any subgame. If we start the game with low-cost and high-cost strategies playing SPE, new high-cost and low-cost strategies will be trained against SPE, but none of them will be added to the game because they cannot achieve a higher payoff than the equilibrium, preventing the game from growing. The same argument applies to any initial game where the tracing procedure only finds the SPE as the equilibrium. While it is clear that SPE is an equilibrium of the game, for our meta-game to expand, the PSRO method requires at least one additional equilibrium at each round.

6.4. Final Experiments: Exploring Initial Meta-Games with Updated Equilibrium Selection

With all hyperparameters set and the WELF equilibrium selection method implemented, we proceeded to experiment with different initial meta-games.

First, to ensure that our implementation was correctly directed towards finding resource-bounded Nash equilibria, we tested an initial meta-game containing only the myopic strategy, assigned to both the low-cost and high-cost players. The payoff of the strategy profile (*myopic*, *myopic*), shown in Figure 6.12, lies on the Pareto frontier of the players' returns. This is because the myopic strategy is highly cooperative. However, (*myopic*, *myopic*) is not a Nash equilibrium of the pricing game: playing lower prices than myopic in the early rounds leads to higher potential demand for the players and, consequently, higher returns.

Our goal in this test was to determine whether, if the PSRO process started from a high-return profile that is not an equilibrium, it could move beyond it. This was quickly confirmed in the first iteration of PSRO, where newly trained strategies were added to the meta-game and the (*myopic*, *myopic*) profile no longer appeared as an equilibrium.

Having confirmed this point, we proceeded with experiments using different sets of strategies in the initial meta-games. Before describing these experiments, we outline the adjustments made to the framework at this stage.

These experiments employed a modified approach to multi-processing. In earlier experiments, separate processes were used to train a single agent while running multiple environments within each process. In this approach, each process independently trains an agent. Previously, only one low-cost and one high-cost agent were trained using each learning algorithm against the equilibrium. Now, we train seven new agents (equal to the number of processes) against the equilibrium, incorporating a mix of low-cost and high-cost agents using the PPO and SAC learning algorithms.

Additionally, in earlier setups, learning was based solely on the top equilibrium, either the one most frequently identified or the one that maximised social welfare. At this point,

we extended the approach to consider the top two equilibria (when available) identified by the tracing procedure that yield the highest social welfare. Within the PSRO framework, after computing the equilibria of the current meta-game, we select the top two and train best response pricing strategies separately against each of them. Any strategy that exceeds the expected payoff of its corresponding equilibrium is retained. We then add the union of all such successful strategies from both equilibria to the meta-game. This adjustment was motivated by the inclusion of the SPE strategy in some initial games, which encouraged exploration beyond SPE. As a result, at each meta-game iteration, seven agents are trained for each of the two selected equilibria.

We ran seven experiments, testing four different initial meta-games as the starting point, with WELF equilibrium selection applied to all of them.

- **MCG-WELF-1** and **MCG-WELF-2**: These two experiments both start from the MCG initial game. They were run for 33 and 50 iterations, respectively, resulting in final meta-game dimensions of 79×78 and 92×95.
- **RND-WELF-1** and **RND-WELF-2**: These experiments start from strategies initialised with random network parameters. They were run for 26 and 45 iterations, respectively, and the final meta-game dimensions are 74×71 and 121×169.
- **GUESS-WELF-1** and **GUESS-WELF-2**: These experiments begin from the GUESS initial game. They were run for 48 and 50 iterations, respectively, resulting in final meta-game dimensions of 94×117 and 125×120.
- **ASPE-WELF**: The meta-game in this experiment starts from ASPE, which includes all deterministic strategies as well as the SPE for both agents. This meta-game was run for 41 iterations, resulting in final meta-game dimensions of 104×74.

One question that might arise is why the number of iterations in these experiments was different. This is due to issues encountered while using an external server to run these experiments. It is possible to continue these games for additional iterations; however, since experiments with the same initial games led to similar results in terms of the equilibrium they converged to, we did not extend the iterations further.

In these experiments, agents using the PPO learning algorithm are trained for 400,000 episodes if they start learning from scratch with random network parameters. If they initialise their network parameters using a previously trained agent, they are trained for 200,000 episodes. For agents using the SAC algorithm, these numbers are multiplied by a factor of 3. This adjustment is based on observations that PPO quickly converges to a policy, whereas SAC requires more time to converge effectively. Our experiments showed that a factor of 3 provides SAC with sufficient time for training.

However, this decision has a downside. PPO is significantly faster than SAC when trained for an equal number of episodes, taking approximately one-third of the time required by SAC. Extending SAC training to more episodes intensifies the time difference. Since agents are trained in parallel processes, the entire iteration must wait for all SAC agents to

finish training, even though the PPO agents finish much earlier. This increases the overall runtime of the experiments.

As the first step in analysing the results, we plotted the details of the seven agents trained in the first iteration of the MCG-WELF-1 experiment to compare them with the previous MCG-FREQ experiment, which was conducted before the extensive hyperparameter tuning. The plots are shown in Figure 6.6. The initial games and equilibria are the same since the MCG initial game has only one equilibrium. Figure 6.10 shows the results of all seven agents in the first iteration to observe the end effect.

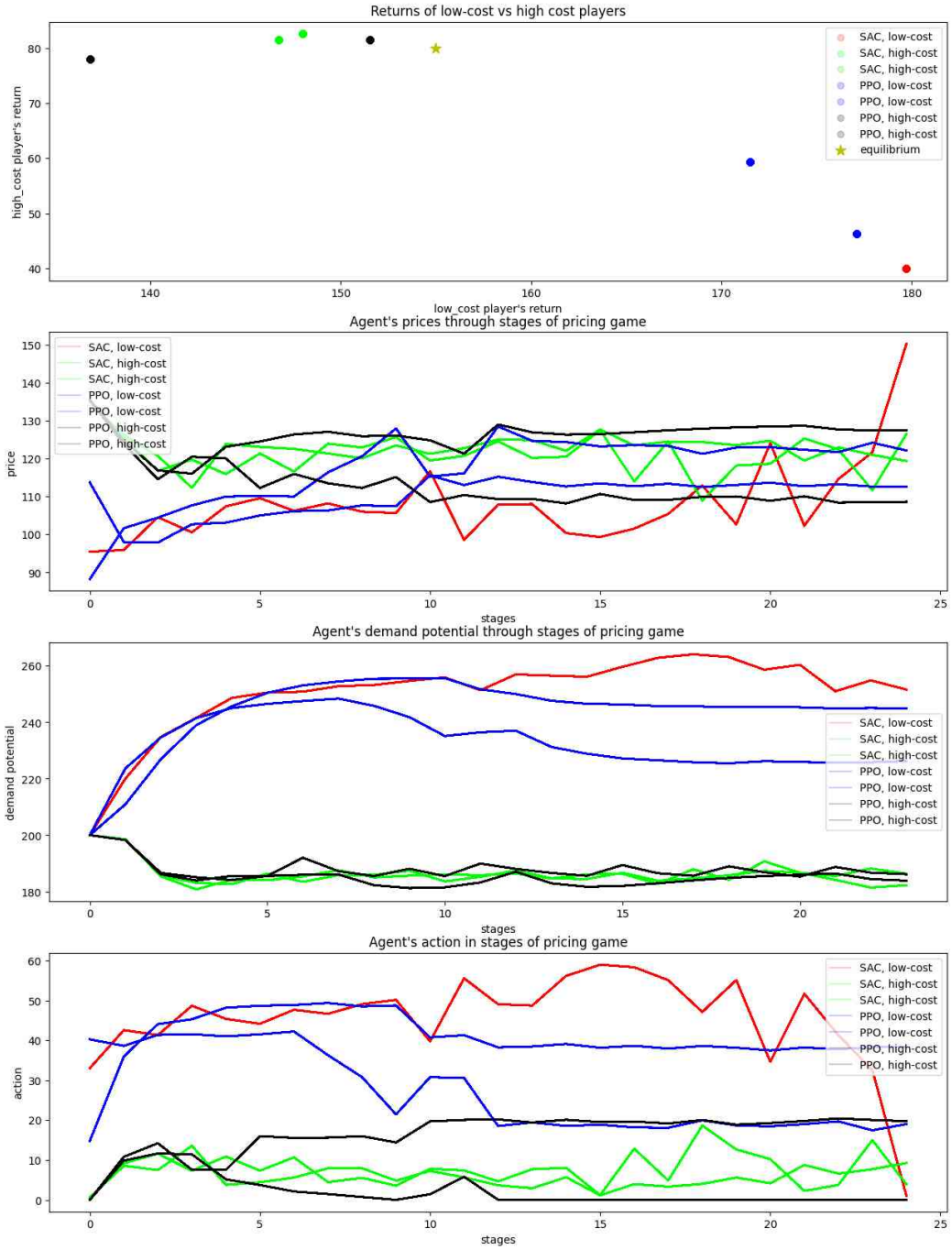


Figure 6.10.: Plot of mean returns, prices, demand potential, and actions of strategies trained in the first iteration of experiment MCG-WELF-1.

The first notable result of the extensive hyperparameter tuning is the stability of the learned policies. Although these agents were trained for half the number of episodes compared to the MCG-FREQ experiment, the standard deviation of the final policies is considerably lower, which indicates convergence to pure strategies for both learning algorithms and both low-cost and high-cost agents.

The SAC low-cost agent maintained its good performance, effectively learning the end-effect and achieving high returns. The SAC high-cost agent's final return improved and is now successful against the equilibrium; however, it has not consistently learned the end-effect.

For PPO, the low-cost agents showed substantial improvement. In the MCG-FREQ experiment, PPO low-cost agents had significantly lower payoff returns compared to SAC low-cost agents. Now, their payoffs have improved by 15% and 19%, respectively, and both agents are added to the meta-game since their payoffs exceed the equilibrium. However, none of them show any sign of learning the end-effect.

	MCG-1	MCG-2	RND-1	RND-2	GUESS-1	GUESS-2	ASPE
SAC, L	$\frac{63}{71}$	$\frac{72}{101}$	$\frac{73}{83}$	$\frac{115}{126}$	$\frac{84}{126}$	$\frac{96}{141}$	$\frac{75}{142}$
SAC, H	$\frac{59}{76}$	$\frac{75}{111}$	$\frac{68}{69}$	$\frac{140}{142}$	$\frac{91}{130}$	$\frac{92}{145}$	$\frac{51}{149}$
PPO, L	$\frac{13}{77}$	$\frac{17}{100}$	$\frac{0}{85}$	$\frac{5}{150}$	$\frac{7}{130}$	$\frac{26}{142}$	$\frac{22}{148}$
PPO, H	$\frac{16}{79}$	$\frac{17}{115}$	$\frac{2}{73}$	$\frac{31}{142}$	$\frac{23}{142}$	$\frac{25}{146}$	$\frac{16}{131}$

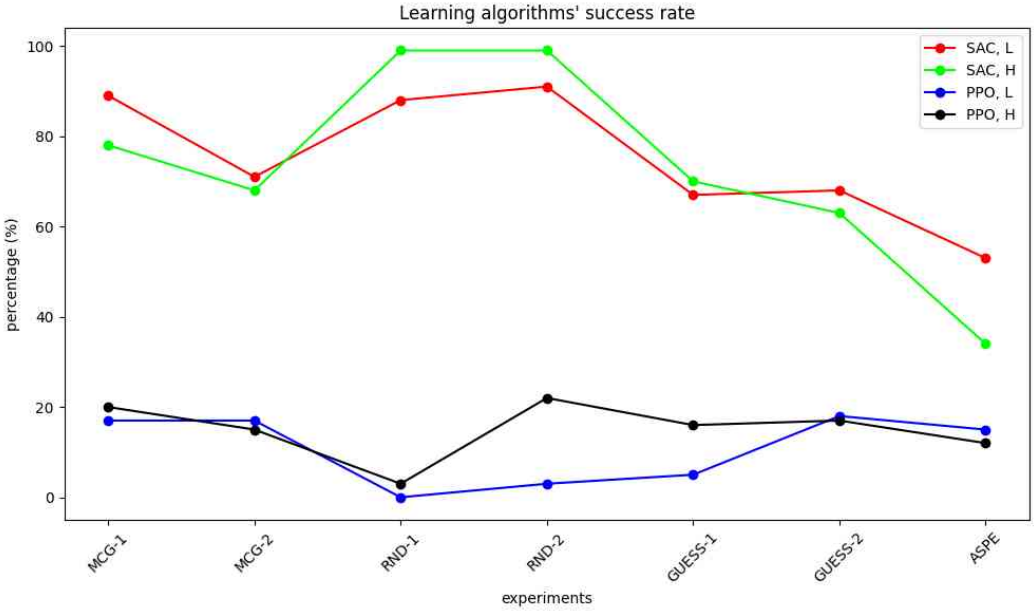


Table 6.8.: Proportion of successful strategies in the final experiments. The rows represent the learning algorithm followed by 'L' (low-cost) or 'H' (high-cost). The columns show the experiments, where the equilibrium selection method for all of them is WELF.

Similar to the previous section, we demonstrate in Table 6.8 the number of successful strategies trained by each learning algorithm, for low-cost and high-cost agents, to evaluate the effect of hyperparameter adjustments on their success rate (in the sense of how often they are being added to the meta-game). To compare this with Table 6.1, recall that the initial game in the MCG-FREQ experiment was MCG, so we compare the success rates only to MCG-WELF-1 and MCG-WELF-2.

The success rate of the trained strategies using both learning algorithms has improved compared to any column of Table 6.1, where the best success rates for SAC-L and SAC-H were 52% and 48% in the third column, whereas now they are higher at 65% for both. Similarly, PPO-L and PPO-H previously had their highest rates in the first column at 9%, whereas now the success rate for both is around 18%.

However, the consistent observation remains that PPO trains considerably fewer successful strategies. To understand the impact of PPO-trained strategies on the meta-game, we need to analyse how often PPO-trained strategies are included in the equilibria of subsequent iterations, as we will investigate in Section 6.4.3.

6.4.1. Equilibria of the Meta-Game

The central question of this research was whether there exist equilibria in the pricing game that yield higher returns for both players than the subgame perfect equilibrium (SPE). Furthermore, if such equilibria exist, we asked whether the strategies trained within our PSRO framework converge to them. Here, the notion of equilibrium is understood in the sense of empirical game theory [72], that is, no better strategy can be found by a *trained* agent.

In the first step, we investigate whether such an equilibrium can exist by analysing the Pareto frontier of the returns for the low-cost and high-cost players. For that purpose, we have to find the maximally achievable payoffs for the players. Similar to Keser [37, pp. 11–14] (with a discount factor of 1), with a parameter λ between 0 and 1 that determines how much profit should be allocated to the low-cost player, this can be found by maximising the weighted sum of the players' returns, working backwards from the last stage, and accepting weights that lead to positive returns for both firms, as assumed in our experiments. The Pareto frontier is plotted in Figure 6.12.

The most cooperative strategy we have defined for the pricing game is the myopic strategy, where the player maximises the payoff at the current stage without considering the demand in the subsequent stages. It is cooperative because if both agents play myopically, they do not compete over the demand potential. Instead, they quickly reach a demand potential split of $(207, 193)$, and the price that maximises the payoffs equals 132, as plotted in Figure 6.11.

However, this strategy is not an equilibrium strategy because agents can gain by deviating at the penultimate stage.

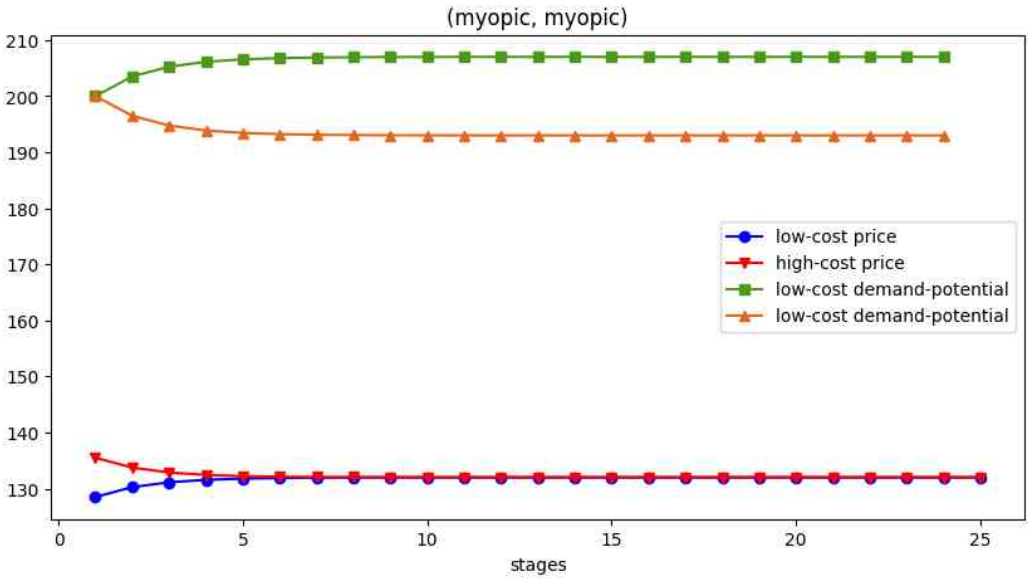


Figure 6.11.: Demand potential and prices of (myopic, myopic) strategies.

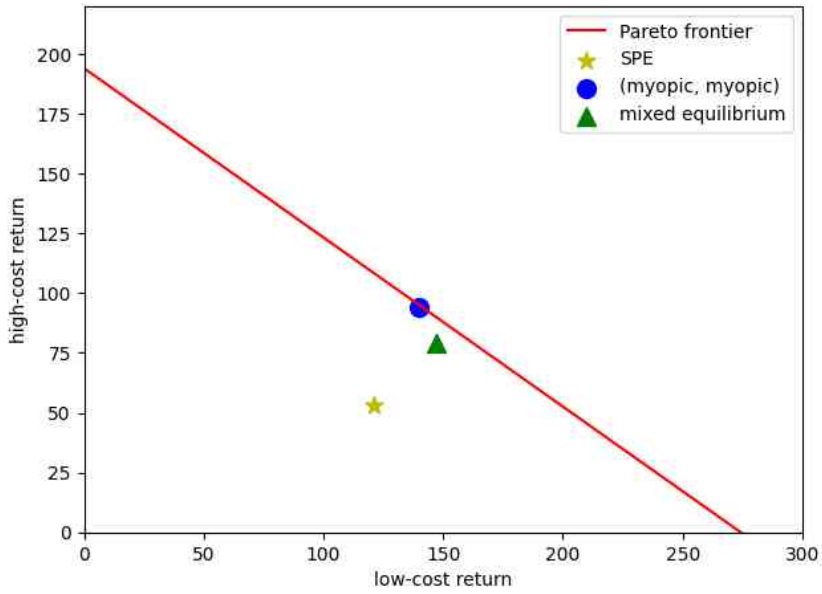


Figure 6.12.: Pareto frontier of returns in the pricing game.

In Figure 6.12, we observe that the return of (Myopic, Myopic) is located on the Pareto frontier, meaning there are no other strategies that allow higher returns for both players. Additionally, the SPE payoff pair lies some distance from the Pareto frontier. We use the returns from SPE and (Myopic, Myopic) as benchmarks for competitive and cooperative strategies.

The third point plotted is an apparent limit of the mixed equilibrium payoffs in the growing meta-game. We discuss this next.

In Figures 6.13, 6.14, and 6.15, we have plotted the returns of all the equilibria of the meta-game found by the tracing procedure as the game proceeds. The equilibria of the early meta-games are shown in a more transparent colour, and the latest ones are bolder.

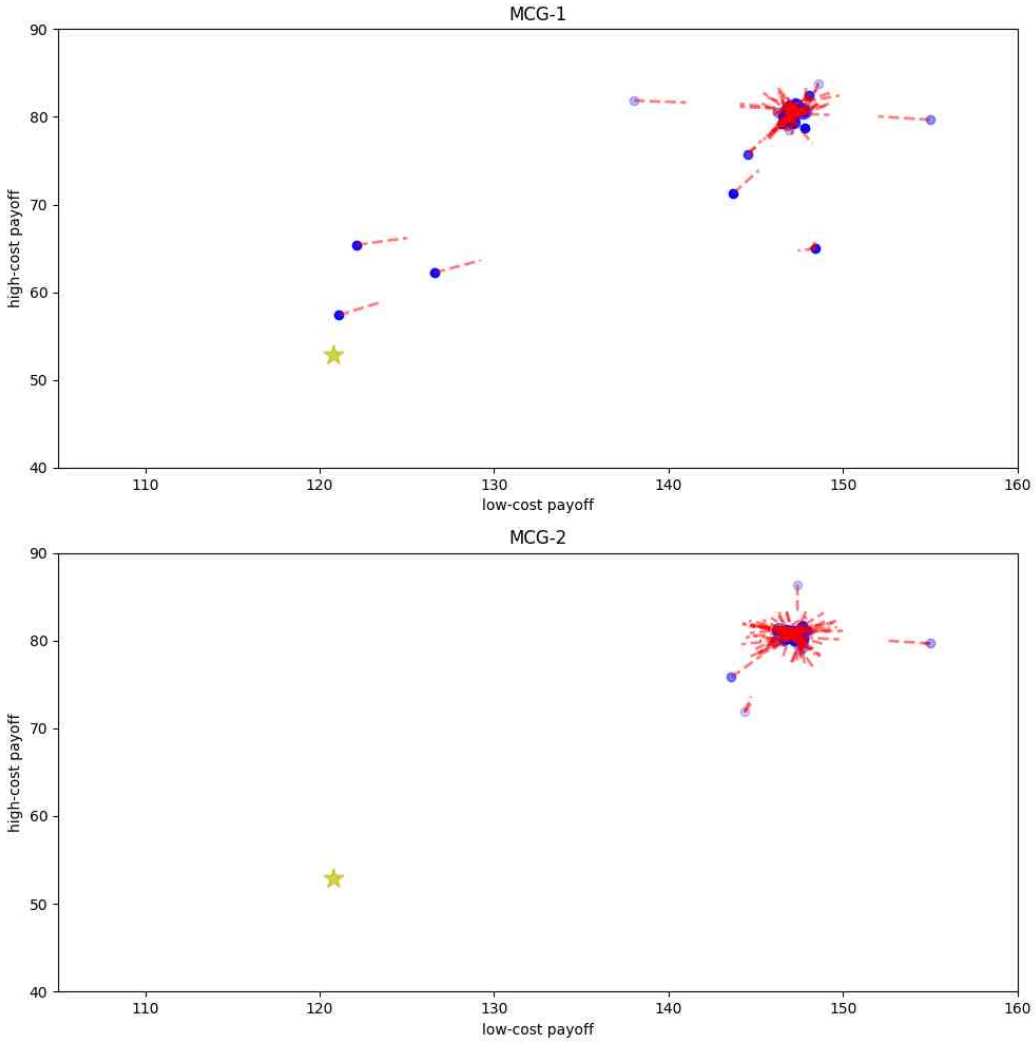


Figure 6.13.: Payoff of equilibria of the meta-game for the MCG experiments.

The dashed lines from each equilibrium point toward the equilibria of the next iteration, where the length of the line is proportional to the frequency of the equilibrium in the tracing procedure.

For the RND-1 and RND-2 experiments, we have not plotted the equilibria from the first iteration because it is an outlier due to the random initialisation of the starting strategies' network parameters. As done throughout, the SPE payoff is marked with a yellow star.

We start with RND-1 and RND-2. In these experiments, the agents are not exposed to any predefined strategies. Training starts from an initial game with a random strategy, similar to a non-rational player, and low-cost and high-cost agents are trained against this strategy. Learning against the random opponent in the first iteration is too noisy, as expected, and since the agents would get different signals from similar actions, they would be dragged in different directions. One advantage, however, is that the initial random strategy playing against itself has such a low return, $(69, 21)$ for RND-1 and $(66, 20)$ for RND-2, that it is very simple for the agents to get above that payoff threshold to be added to the game. Thereafter, the initial random strategy never comes up again in any of the equilibria of the meta-game in subsequent iterations.

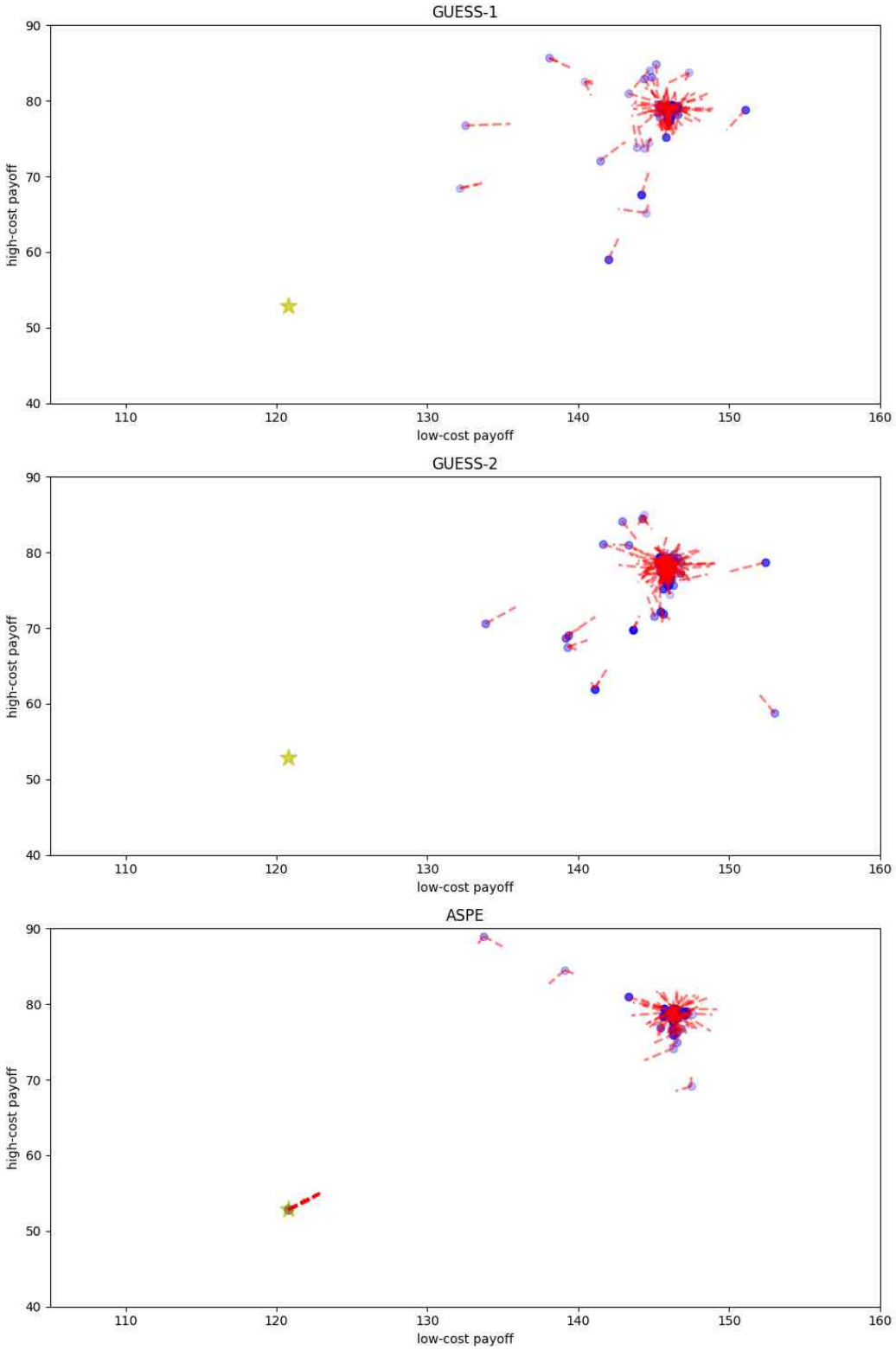


Figure 6.14.: Payoff of equilibria of the meta-game for the GUESS and ASPE experiments.

Our plots for both RND experiments show a wide cluster of equilibria (more dense for the experiment with more trained strategies). This cluster includes the payoff from the SPE strategy. Although the equilibria have not converged to the SPE, the behaviour of some strategies against opponents is similar to the SPE. For example, for the high-cost trained agent

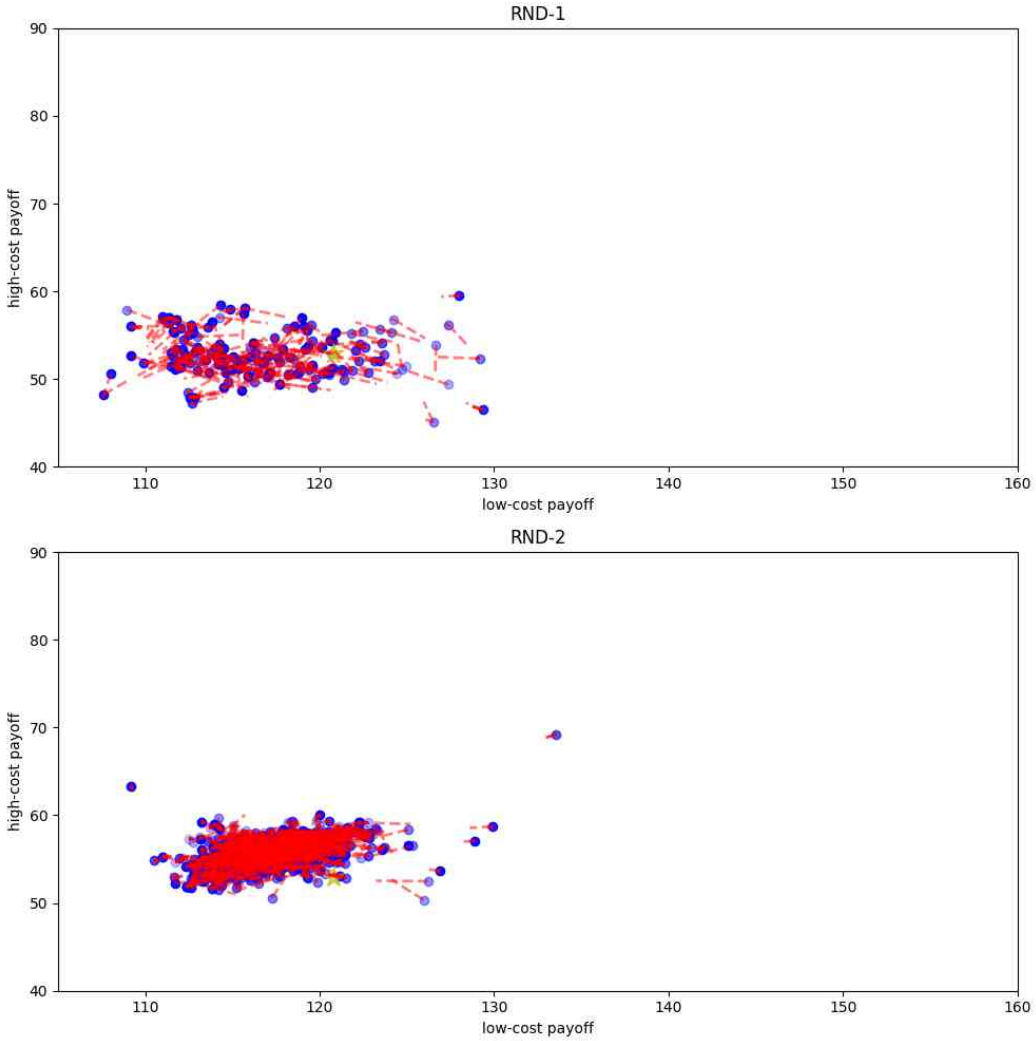


Figure 6.15.: Payoff of equilibria of the meta-game for the RND experiments.

RND-2_H_SAC_167 (see Figure 7.16) and low-cost trained agent RND-2_L_SAC_119 (see Figure 7.7), their path and level of demand potential and prices against the red opponent are similar to the SPE (4.4). These two strategies are among the last agents trained using SAC that are added to the RND-2 meta-game.

These observations suggest that, if the meta-game were continued, the equilibria might converge closer to the subgame perfect equilibrium (SPE). We did not continue this process due to computational cost and RAM capacity limitations. However, it is possible to remove the strategies that do not appear in the equilibria, reduce the size of the meta-game, and continue from there. One approach to identifying these strategies is by studying the average probabilities of the strategies in the equilibria, as explained in Section 6.4.3 below. Nonetheless, we did not proceed with continuing this experiment, as there were no indications of an emergence of other equilibria with higher potential returns.

An intriguing result was observed in other experiments. In experiments where some smart strategies were introduced into the initial meta-game, the equilibrium payoffs consistently approached a neighbourhood around $(147, 78)$. This neighbourhood includes most equilibria of the meta-game as the game progresses, regardless of whether the

experiments started from the MCG, GUESS, or ASPE initial games. However, starting from the RND game, none of the equilibria were near this point.

This suggests the existence of an equilibrium with the mentioned returns, which is closer to the Pareto frontier than the SPE and results in higher returns for both players. In all the experiments that approached this equilibrium, the equilibria were mixed. Therefore, we refer to this empirical equilibrium as the mixed equilibrium, with the payoff pair marked in Figure 6.12 with a green triangle.

Another interesting observation, from Figures 6.13 and 6.14, was that in the MCG and GUESS experiments, the equilibria did not approach the SPE at all, in contrast to the RND experiments. This observation inspired the idea behind the ASPE initial game, as we became curious about what would happen if we introduced the SPE to the game ourselves.

As can be seen in the plots, although the SPE is one of the equilibria found in all iterations of the ASPE experiment, there is still a similar cluster of equilibria to one in the GUESS and MCG experiments, at the top right. In the ASPE experiment, other than for a small number of outliers, in all iterations of the meta-game, one equilibrium is the SPE, and the other equilibria are in the discussed neighbourhood.

An important point in the ASPE experiment is that the SPE is not the only equilibrium of the initial game; otherwise, the meta-game would not grow at all. This is because the SPE strategy of the low-cost player is the unique best response to the SPE strategy of the high-cost player and vice versa. By definition of an SPE, there is no other strategy that can achieve a higher payoff than the payoff of that equilibrium when trained against these strategies. In the ASPE experiment, the equilibrium that helps the meta-game to progress is the (Guess3, Guess3) equilibrium. None of the strategies trained against the SPE is added to the game because they cannot reach the payoff threshold. However, they are used in early iterations as base agents to set the initial parameters of the next training agents.

6.4.2. Price Instability

As we studied the price path of trained agents, we noticed significant fluctuations in prices. These fluctuations can make it harder for opponents to anticipate future prices and, therefore, can decrease the chance of cooperation.

In this section, we study the fluctuations in prices for learning algorithms and experiments. We need a measure to study the fluctuations in the prices set by the agents throughout the pricing game. We define the price instability of the price vector P (prices played over the stages of the pricing game by the agent) as follows:

$$\text{price instability } (P) = \frac{\sum_{t=2}^{25} |P_t - P_{t-1}|}{24} \tag{6.13}$$

Keser in [37, p. 32] divides cooperative strategies based on the price instability of the strategies, where those with lower price instability are considered strongly cooperative, whereas higher instability leads to weakly cooperative strategies. It is important to note that Keser defined the price instability as the sum of the squared price differences (L_2 norm).

However, here we define in (6.13) the price instability using the L_1 norm to account directly for the average fluctuation in price per stage.

For reference, the price instability for the prices of the SPE strategies is $(2.01, 1.76)$; for the $(\text{Myopic}, \text{Myopic})$ strategy pair, it is $(0.14, 0.14)$; and for the $(\text{Guess2}, \text{Guess2})$ strategies, it is $(7.84, 4.23)$ for the low-cost and high-cost agents, respectively.

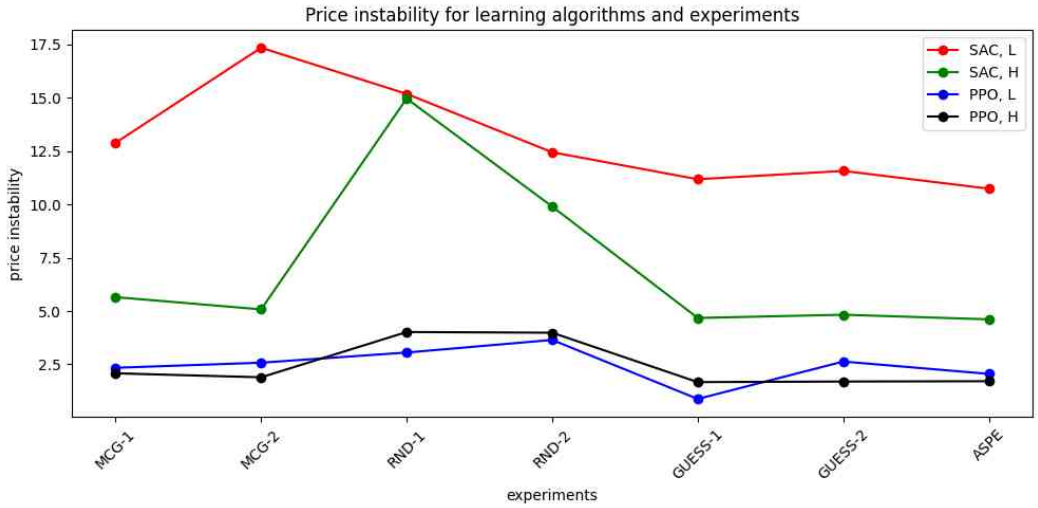


Figure 6.16.: Plot of price instability for each experiment, averaged over the last 20 trained strategies. Each strategy was run for 10 iterations against the opponent with the highest probability. The plot shows price instability for low-cost and high-cost agents trained by each learning algorithm.

In Figure 6.16, we can see the price instability of the last 20 trained agents using each learning algorithm, separated for high-cost and low-cost agents. PPO clearly exhibits lower price instability in training both types of agents, which is consistent with our previous observations. As we saw in Figure 6.10, the PPO-trained agents did not capture the end-effect of the game very well. We suspect that, despite the one-hot encoding of the stage of the pricing game in the state, PPO does not differentiate much between different stages of the pricing game. This can lead to playing a consistent price over the stages. While this is advantageous in terms of cooperation, not understanding the end-effect is suboptimal. This is apparent in Figures 7.18 and 7.17, which show two of the final PPO-trained strategies in our experiments. The fluctuations in prices are very low compared to the SAC-trained agents, and interestingly, in the latter figure, we can see that the agent plays very similarly to the Myopic strategy, as the action values are very low.

Between PPO-trained strategies, the difference in the instability of prices between low-cost and high-cost agents is not significant. However, for SAC, low-cost strategies show considerably more unstable prices compared to high-cost ones. Additionally, this slightly explains why PPO is more successful in training high-cost strategies, as SAC and PPO exhibit more similar behaviours in training high-cost agents.

Excluding RND experiments (because of their high price instability), we observe that low-cost trained strategies have considerably lower price instability for the SAC algorithm. As SAC is successful in training both types of agents, we interpret the low-cost strategies

as weakly cooperative compared to the high-cost ones trained by SAC, which are more cooperative.

As for the experiments, considering the sum of instability for low-cost and high-cost agents, RND experiments exhibit the highest instability in prices for both SAC and PPO. It is evident that the random starting point is highly unstable. However, it is interesting to note that even after many iterations of the meta-game and each strategy being trained for millions of episodes, the price instability remains higher than in other experiments.

6.4.3. Average Strategy Probabilities in Equilibria

In each iteration of the meta-game, only newly trained strategies that have higher returns than the equilibrium payoff are added to the game. However, this does not mean they will necessarily be effective in the subsequent iterations of the meta-game. If these strategies have positive probabilities in the equilibria we choose, they will be used to train the next agents. However, if they do not appear in the top two equilibria as found by the tracing procedure, their only effect will be in setting the initial neural network parameters for the next agents. These are the strategies that we propose to be omitted if we need to reduce the size of the meta-game to continue the process due to the large storage requirements for a large game. We did not implement this reduction because the experiments that were most important to us already answered our questions regarding convergence to an equilibrium with higher social welfare.

In this section, we aim to study the strategies that were most effective in the progression of the meta-game. We analyse the effect of learning algorithms on training each type of agent. Additionally, we examine the impact of the deterministic strategies introduced to the initial game, as this is an interesting aspect to address.

In each experiment, we consider the set of all equilibria that were used to train agents at different iterations of the meta-game and compute the average probability of each strategy in these equilibria. Clearly, the earlier strategies have a higher chance of appearing in the equilibria, whereas the last added strategies did not have the opportunity to be included in the equilibria since they were not part of the meta-game before. Therefore, for future work, we suggest introducing a threshold on the minimum number of iterations a strategy must be present in the game before omitting it due to low average probability as part of reducing the size of the meta-game.

In the following Figures 6.17, 6.18, 6.19 and 6.20 for each experiment, the left plot indicates the average probabilities for each strategy with positive average probabilities. The names on the x-axis are displayed for any strategy with an average probability higher than 0.03. The deterministic strategies are identified by their names. To avoid overcrowding, trained strategies are represented with the starting letter ‘S’ for those trained with SAC and ‘P’ for those trained with PPO, followed by their index in the meta-game.

In the right plot, we show the sum of probabilities for trained agents using each training algorithm and deterministic strategies (specified as `statics` in the plots) to illustrate their overall effect.

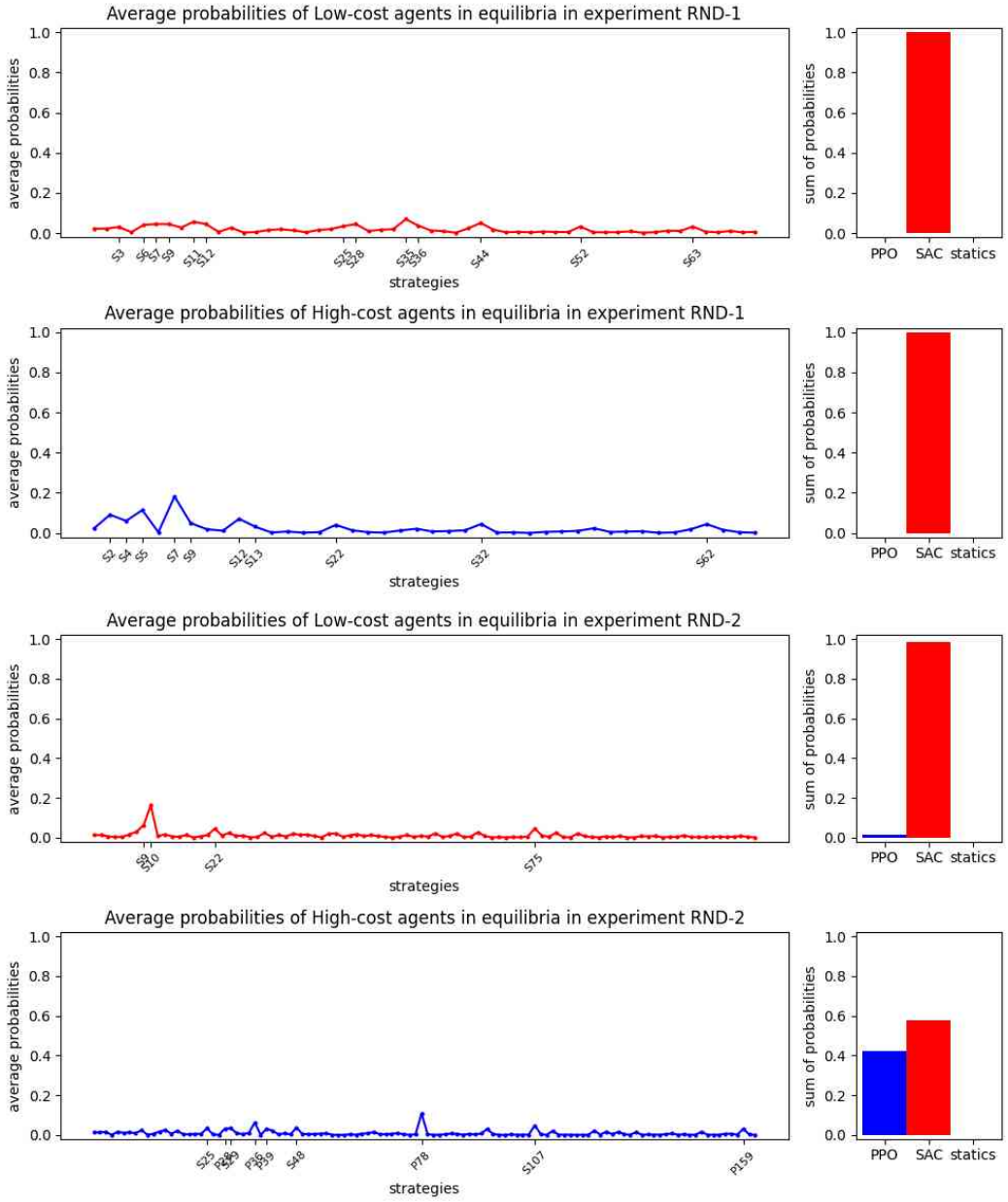


Figure 6.17.: Plot of average probabilities of strategies in the top 2 equilibria for experiments RND-1 and RND-2.

The RND experiments start with strategies initialised with random parameter networks, so no deterministic strategies were introduced. In Figure 6.17, we can see the average probabilities of strategies for experiments RND-1 and RND-2. In both experiments, among low-cost and high-cost agents, there is no specific agent that appears to act considerably more effectively than others in equilibria, as observed in other experiments. The maximum average probabilities among these strategies are about 0.2.

For low-cost agents, almost all added strategies were trained using the SAC algorithm. However, for high-cost agents, the PPO algorithm had a better rate, though still less successful than SAC. This trend is observed in other experiments as well: PPO performs better in training high-cost agents but is not as effective for low-cost ones.

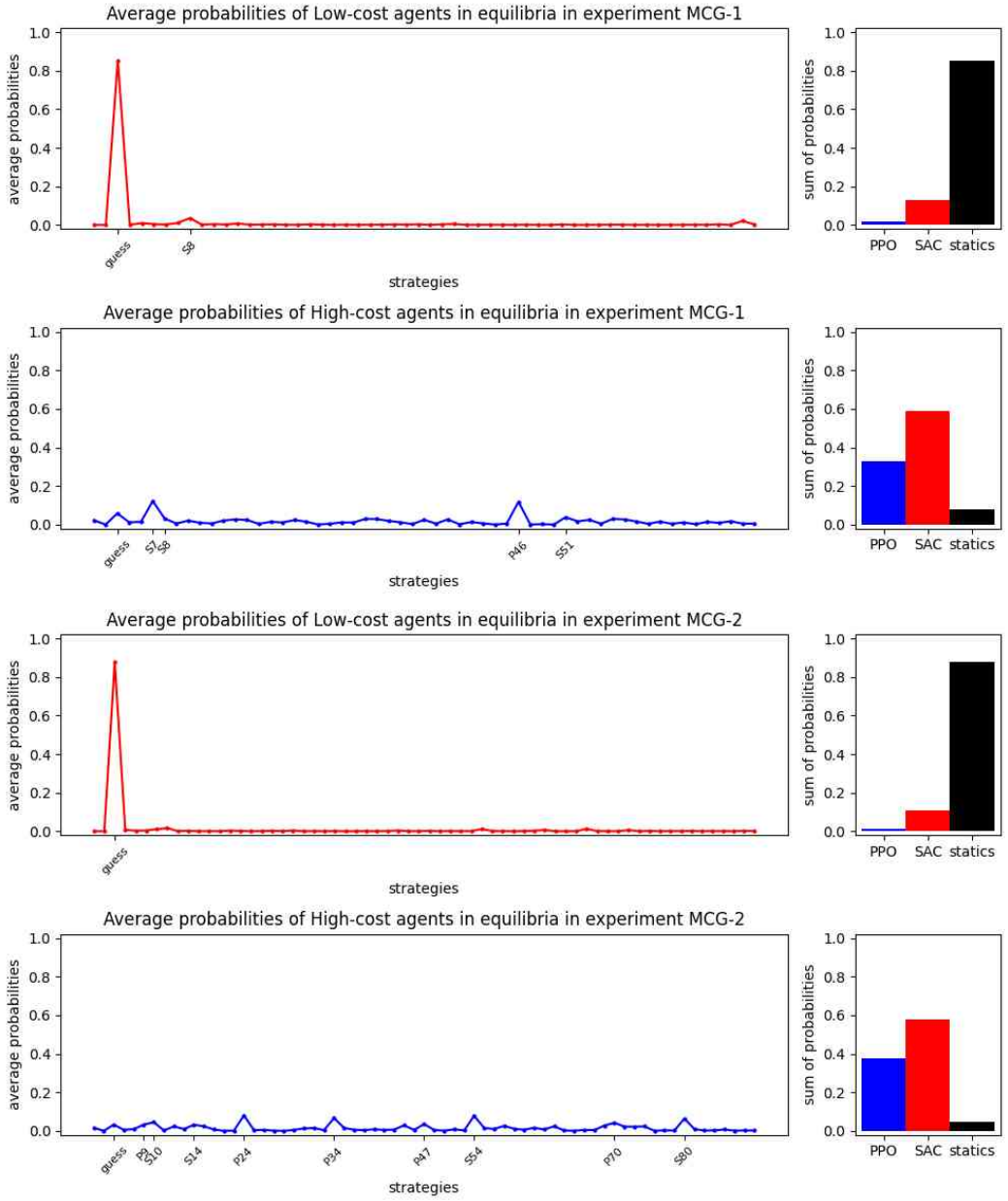


Figure 6.18.: Plot of average probabilities of strategies in the top 2 equilibria for experiments MCG-1 and MCG-2.

In Figure 6.18, we observe the plots for experiments MCG-1 and MCG-2. The Guess-132 deterministic strategy is undoubtedly the most effective strategy among low-cost agents, with an average probability nearing 0.9, appearing very frequently in the low-cost strategies of equilibria with a high probability. This strategy is competitive but allows a level of cooperation, which is advantageous for low-cost agents as they can compete more aggressively if cooperation breaks down.

For high-cost agents, however, we see the emergence of SAC and PPO-trained strategies that appear more frequently in the high-cost strategies of equilibria than the Guess-132 strategy. These trained strategies yield higher returns against the low-cost Guess-132 strategy (which they most frequently face, as indicated by the high average probability of Guess-132 in low-cost strategies) than the high-cost Guess-132 strategy does. This

outcome suggests greater cooperation by these trained strategies—they achieve the desired demand split by the Guess-132 strategy but play slightly higher prices at that demand level, increasing the returns for both players.

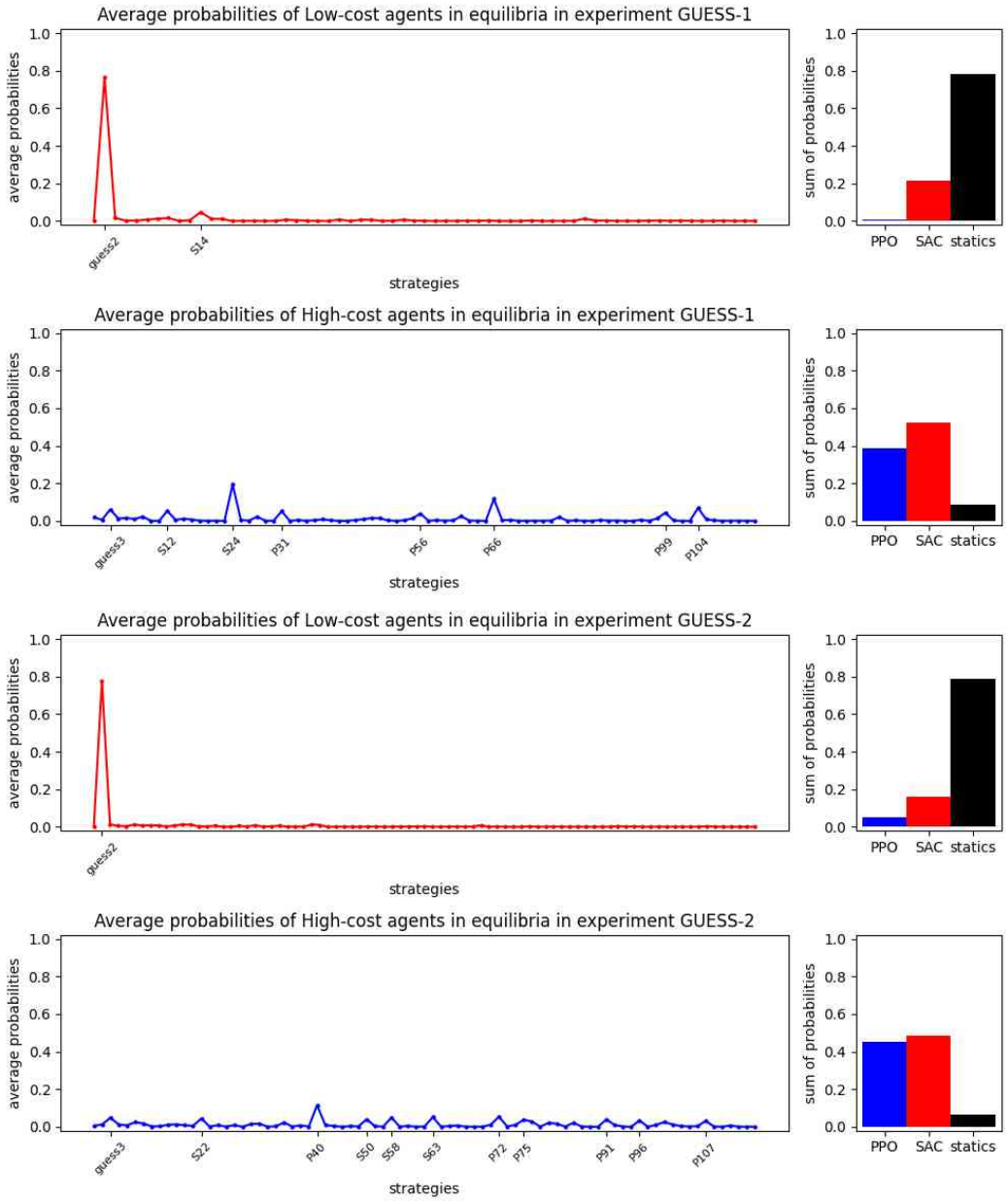


Figure 6.19.: Plot of average probabilities of strategies in the top 2 equilibria for experiments GUESS-1 and GUESS-2.

The results from the MCG experiments encouraged us to set up the GUESS experiments to observe the agents' behaviour when introduced to more variations of Guess-132. To recap, we consider Guess2-132 as more competitive and Guess3-132 as more cooperative compared to Guess-132, based on the returns they achieve when playing against themselves.

The plots in Figure 6.19 show the average probabilities of strategies in equilibria for experiments GUESS-1 and GUESS-2.

Similar to the MCG experiments, among the low-cost agents, the deterministic strategies are the most effective. However, in the GUESS experiments, the Guess2 strategy has replaced Guess as the most effective strategy observed in the MCG experiments. The Guess2 low-cost strategy has an average probability of 0.8, whereas the other Guess strategies have probabilities near zero. This indicates a more competitive behaviour from the low-cost agents compared to the MCG experiments.

For high-cost agents, similar to the MCG experiments, some trained agents using SAC and PPO achieve higher average probabilities in equilibria than the Guess strategies. However, an interesting observation is that among the Guess variations, Guess3, which is more cooperative, has higher effectiveness in the meta-game.

Compared to the MCG experiments, the GUESS experiments suggest that more competitive agents are more effective among low-cost strategies, whereas cooperative agents are more effective among high-cost strategies. This indicates the direction of behaviour that the equilibria of the meta-game encourage.

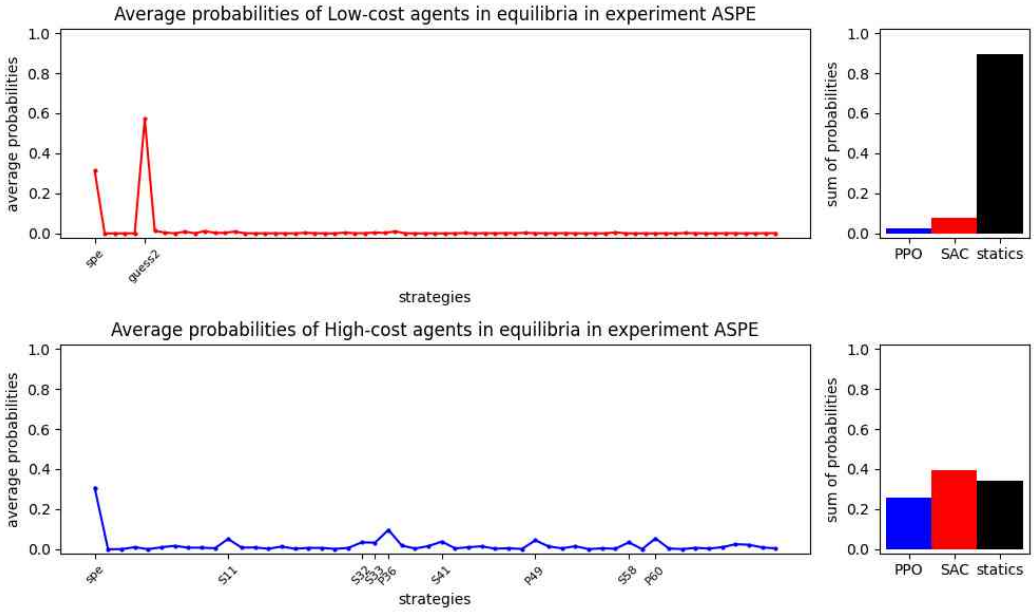


Figure 6.20.: Plot of average probabilities of strategies for the ASPE experiment.

Finally, Figure 6.20 shows the average probabilities in the ASPE experiment, where we included the SPE strategy as well as all other deterministic strategies we defined in Section 4.3.2.

The first point to note about this plot is that, since SPE is the equilibrium found in all iterations, the average probability of SPE in both low-cost and high-cost strategies is high. However, the average probabilities of SPE in equilibria involving trained agents that are added to the game are very close to zero, slightly higher for low-cost agents, but still near zero. Hence, we do not draw conclusions based on the SPE average probabilities.

Among low-cost agents, Guess2 appears more frequently in equilibria, similar to the GUESS experiments. However, with the existence of the SPE strategy in the game, which is

highly competitive, we believe the reason that Guess2 has a higher average probability is that it allows cooperation to some level.

For low-cost agents, none of the deterministic strategies seem to gain a high average probability among the equilibria that include trained agents added to the game (other than SPE). Instead, among the high-cost strategies, the ones with the highest average probability are those trained with SAC and PPO.

The most effective high-cost strategy is P36, and in Figure 6.21, we have plotted the prices and demand potential of this strategy playing against the low-cost Guess2 strategy, comparing it with the Guess2 strategy playing against itself.

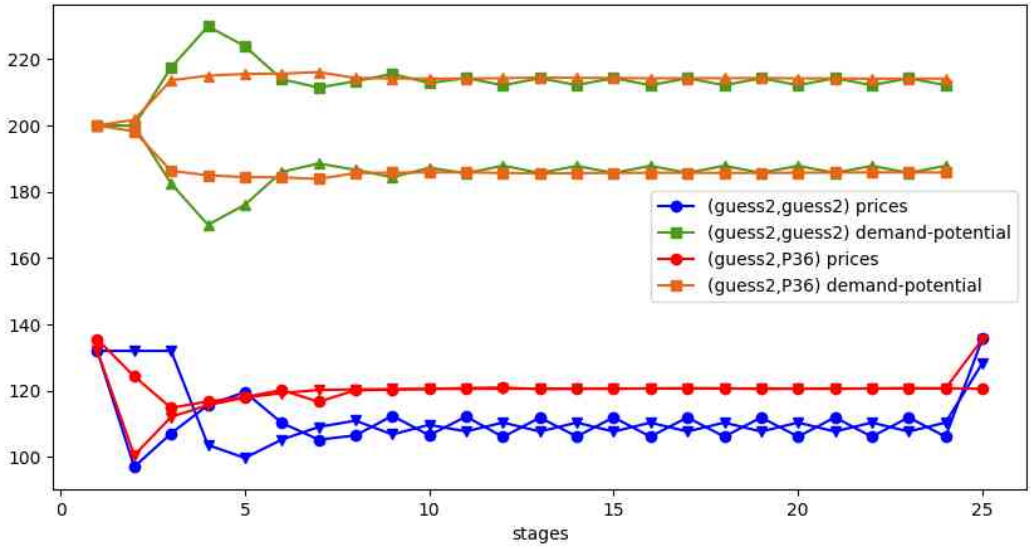


Figure 6.21.: Plot of prices and demand potential of players, comparing (Guess2, Guess2) to (Guess2, PPO-36) as low-cost and high-cost players for the ASPE experiment.

Similar to high-cost strategies in the GUESS experiments, we can see that P36 cooperates with Guess2 to reach the desired demand potential level observed in (Guess2, Guess2). This cooperation leads to higher prices for both the low-cost Guess2 and high-cost P36 strategies compared to prices in (Guess2, Guess2). This results in a return of 146 for Guess2 and 82 for P36, compared to 137 and 75, respectively, when Guess2 played against itself.

To conclude, through these experiments, we observe that the most frequent strategies in equilibria among low-cost agents are the most competitive ones that allow cooperation to some level. The Guess2 strategy has been the most effective low-cost agent in these experiments, and none of the trained strategies using PPO and SAC could reach its average probability in equilibria. However, for high-cost agents, the learning algorithms were able to develop cooperative strategies that appeared more frequently in equilibria than all our defined high-cost deterministic strategies. Both SAC and PPO train many effective strategies in the equilibria, with SAC achieving a slightly higher rate.

6.4.4. Behaviour of Final Pricing Strategies

In this section, we summarise our findings from the previous sections and analyse the behaviour of the final trained strategies to better understand our empirical mixed equilibrium. In Section 6.4.1, we observed that, in the payoff plots of equilibria across all non-RND experiments, there is a payoff region around $(146, 80)$ to which all our equilibria converge. We consider this point to be an empirical mixed equilibrium of the pricing game corresponding to these payoffs. However, the strategies that converge to this point differ, and even within each experiment, different equilibria of the meta-game involve a mix of various agents. In this section, we examine the behaviour of the final agents trained in each experiment to better understand the strategies' behaviour at this empirical mixed equilibrium.

We now concentrate on the MCG, GUESS, and ASPE experiments (non-RND experiments), as the RND experiments do not relate to the mixed equilibrium discussed here. Starting with the low-cost trained strategies, we observed in Section 6.4.3 that, in all non-RND experiments, the strategy with the highest probability in equilibria is a variation of the Guess strategy. When Guess2 is included in the initial game, it is the most effective; otherwise, the Guess strategy dominates.

Figures 7.4, 7.5, and 7.2 show the strategies added to the meta-games in the final iteration of the GUESS-2, ASPE, and MCG-2 experiments, respectively, all trained with SAC. Comparing the demand potential and price paths of these agents to those of the low-cost Guess2 strategy against the PPO-trained high-cost strategy P36 (see Figure 6.21) provides a strong indication of why Guess2 is more successful among low-cost strategies in the meta-games.

First, in these trained strategies, a similar trend of seeking up to 260 units of demand potential can be observed. This level of demand potential often results from setting very low prices at certain stages. Additionally, there are significant fluctuations in price, consistent with the observed price instability in SAC-trained strategies. These competitive behaviours, coupled with price instability, make these strategies less cooperative compared to the Guess2 strategy.

When Guess2 played against the cooperative high-cost P36 strategy, it achieved a demand potential of approximately 215 (top orange line), but at a high price of around 120 (top red line). This resulted in higher returns for both Guess2 and its opponent compared to SAC-trained strategies, whose excessive competitiveness reduced their profitability. Furthermore, the low price instability of Guess2 made it more cooperative with its opponent.

As for the high-cost strategies, Figures 7.11, 7.14, and 7.9 show the strategies added to the meta-games in the final iteration of the GUESS-2, ASPE, and MCG-2 experiments, respectively, all trained with SAC. The main opponent of these strategies during training (the red opponent in the plots) is the Guess2 strategy in the GUESS-2 and ASPE experiments and the Guess strategy in the MCG experiment.

Comparing their demand potential (bottom green line) and price path (bottom blue line) to those of Guess2 against Guess2 as shown in Figure 6.21, we observe that the demand

potential they seek is similar to that of high-cost Guess2, around 180. However, these strategies achieve this demand potential at a higher price of approximately 120, compared to about 110 for Guess2. Additionally, their price instability is lower than that of the low-cost trained strategies discussed previously.

This indicates a more cooperative behaviour, which is closer to that of the Myopic strategy (can be seen from actions that are generally lower than 4, meaning they do not deviate more than 4 units from the myopic price). Overall, these observations suggest highly cooperative behaviour in these trained high-cost strategies.

Comparing the above agents with their equivalents in RND strategies, as shown in Figures 7.7 and 7.8 for low-cost agents, and Figures 7.15 and 7.16 for high-cost agents, reveals notable differences. The high-cost RND strategies reach a similar demand potential level in the range of 180 to 200, but at a lower price of 90 to 100. For low-cost RND strategies, they compete at most to achieve a demand potential of 240 (compared to 260 for non-RND strategies); however, their price paths are similar to those of the non-RND strategies.

To summarise these comparisons, our results show that, at the empirical equilibrium, low-cost strategies are weakly cooperative, while high-cost strategies are strongly cooperative. The SAC-trained low-cost agents become too competitive; therefore, our introduced deterministic strategy, Guess2, which is a weakly cooperative strategy, proves more successful. However, for high-cost players, both SAC and PPO successfully train strongly cooperative strategies that are applied at the empirical mixed equilibrium.

6.4.5. Replicator Dynamics

In the last part of the experiment by Keser [37, p. 97], all the strategies entered an evolutionary game, and she studied the evolution of strategies using the replicator dynamics.

A variation of replicator dynamics known as *projected replicator dynamics* was also suggested [40] as another meta-strategy solver, instead of computing the Nash equilibrium in the meta-game, in order to prevent overfitting to the Nash equilibrium in PSRO meta-games.

Replicator dynamics, inspired by natural selection, considers a portion of a population playing each strategy (known as the weight of that strategy). Then, the fitness of each strategy is defined as its expected payoff against the opponents' strategies with respect to their weights. After assessing the fitness of each strategy, the successful strategies are identified by comparing their fitness to the average payoff of the population. Then, replicas of successful strategies are added to the population, while the defeated strategies are removed at some rate. This means that the weight of successful strategies increases and the weight of the less successful strategies decreases. In this iterative process, the distribution of the population keeps changing until convergence or a stopping criterion is met.

As we mentioned in Section 6.3.3, we used the tracing procedure to compute the equilibrium in our PSRO settings. This was part of our new approach, and as we saw, the

choice of the selected equilibrium turned out to be highly important in the final equilibrium reached in the iterative process of training agents to be added to the meta-game.

The similarity between the concepts of updating beliefs based on players' strategies and best responding to them in the tracing procedure, and increasing the population of more successful strategies based on their return against other players in replicator dynamics, incentivised us to conduct the final analysis in this chapter and run the replicator dynamics on our strategies in the final meta-games, using the same parameters as Keser's experiment.

To run the replicator dynamics, we start with a uniform distribution over all strategies, meaning that all strategies have equal weight at the start. Additionally, since we want the strategies not to die off quickly and continue to be involved in the evolutionary process, we add a probability of 10^{-6} to all strategies at the start of each iteration and then normalise the distribution. Moreover, we set the learning rate (lr) to 10^{-6}, meaning that the population is updated using this learning rate as the step size toward the new population distribution.

In Figure 6.22, we have plotted the returns of strategies for RND experiments in replicator dynamics, with each payoff point indicating the direction of the payoff point for the next iteration's strategies.

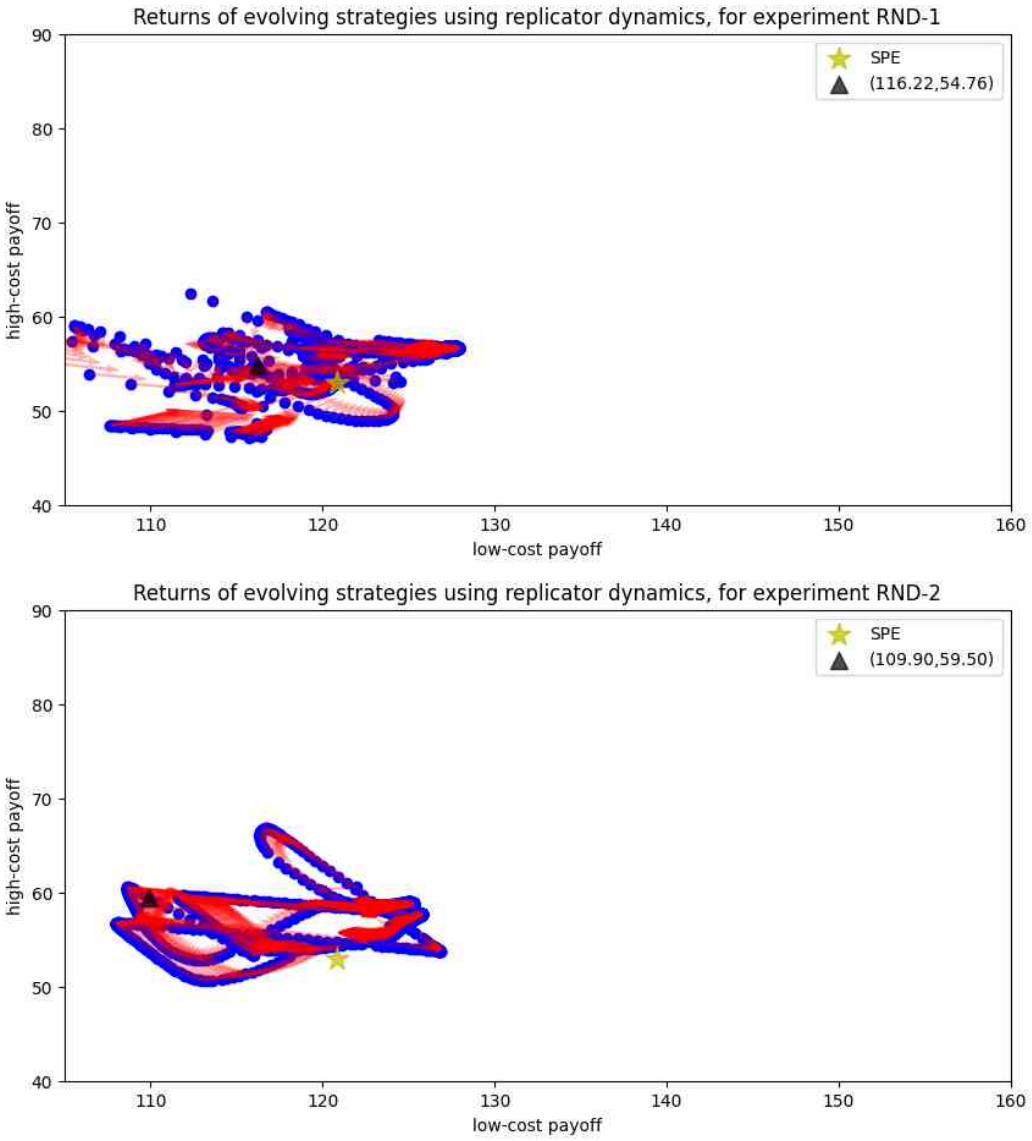


Figure 6.22.: Plot of returns for evolving strategies using replicator dynamics, for RND experiments.

We ran the replicator dynamics for 2×10^6 iterations, and as can be seen in the plot, they did not become stable. The returns oscillate, approaching many equilibria found in the meta-game, but do not stabilise by converging to the SPE or any specific equilibrium of the meta-game.

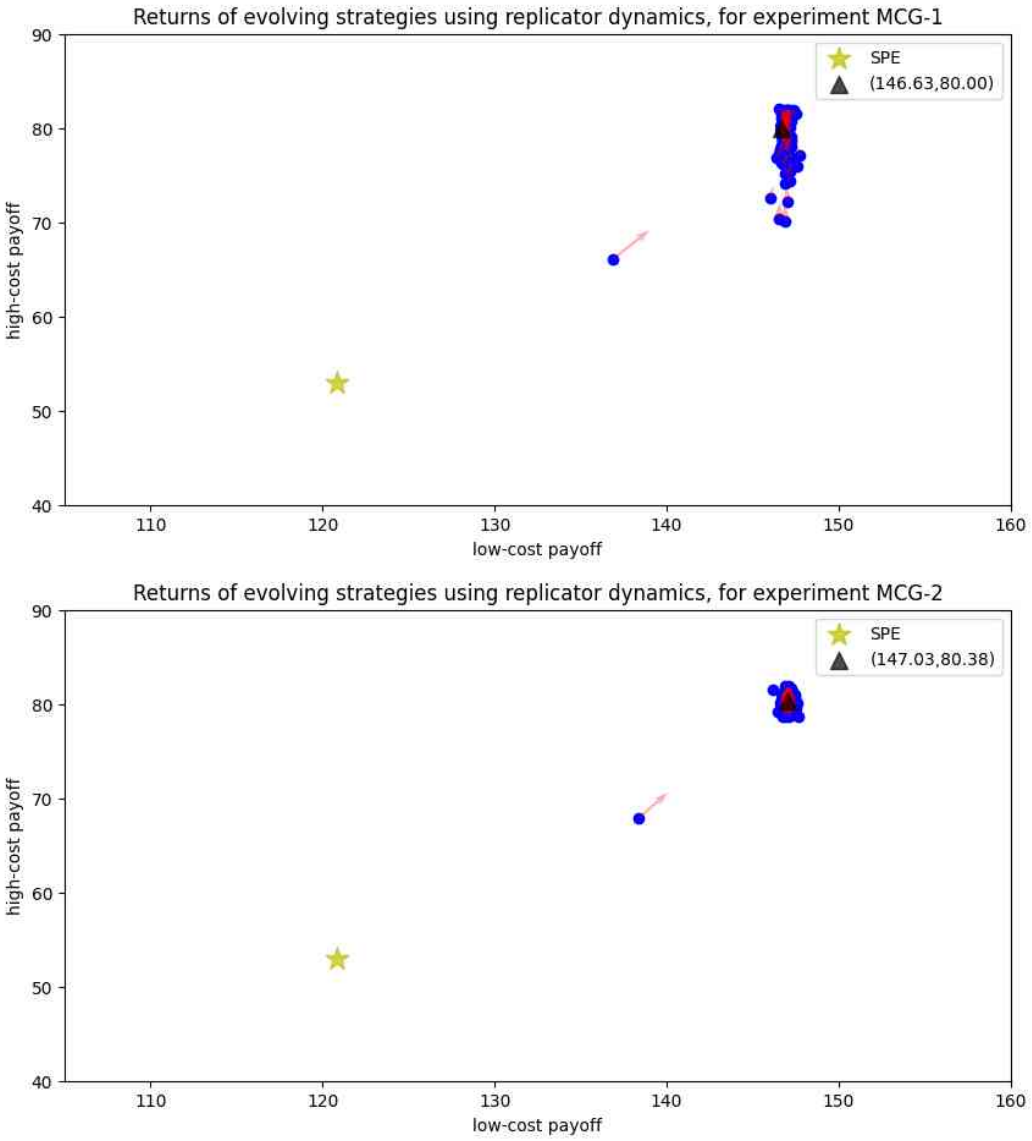


Figure 6.23.: Plot of returns for evolving strategies using replicator dynamics, for the MCG experiments.

In Figure 6.23, we observe the returns of strategies in the MCG experiments. In these experiments, we observed convergence to the equilibrium of the final meta-game, which has the highest social welfare, in fewer than 400,000 iterations. In these equilibria, Guess2 has the highest probability among low-cost strategies, whereas for high-cost strategies, many trained strategies with higher average probabilities, as shown in Figure 6.18, have significant probabilities.

We observed similar behaviour in the replicator dynamics for the GUESS and ASPE experiments (Figures 6.24 and 6.25), albeit with more oscillations. The final strategies are within a very small neighbourhood of the final meta-game's equilibria. In GUESS experiments, the final strategy is close to the equilibrium with high, though not the highest, social welfare. Since all the equilibria of the final meta-game are very close in terms of both returns and strategies, we consider this a positive result.

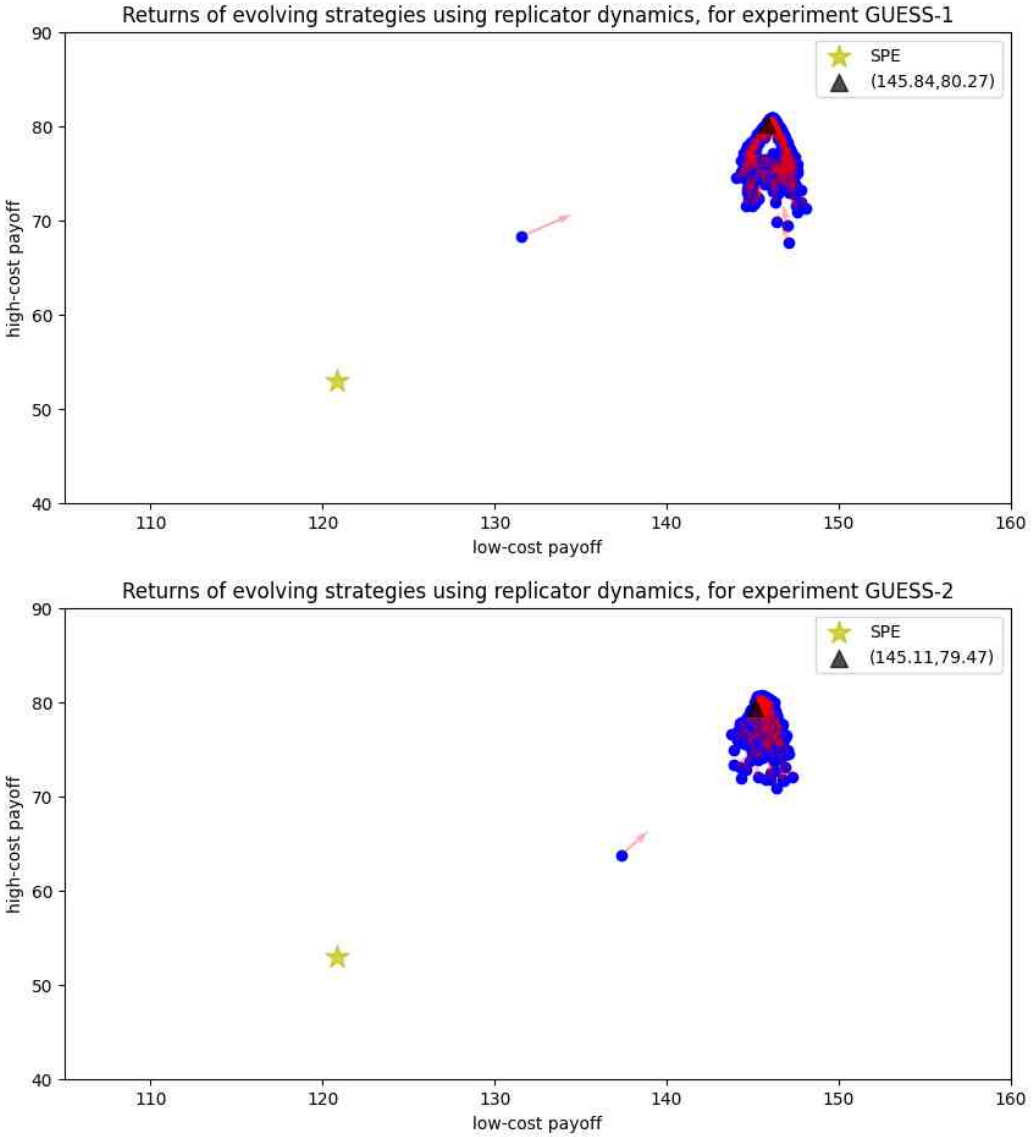


Figure 6.24.: Plot of returns for evolving strategies using replicator dynamics, for the GUESS experiments.

The replicator dynamics results for all experiments were consistent with the equilibria found using the tracing procedure for the meta-games, as explained in Section 6.4.1. The cluster of equilibrium payoff points observed in the non-RND experiments was also reached using replicator dynamics. Additionally, the oscillations and lack of convergence to a specific equilibrium were observed in the RND experiments.

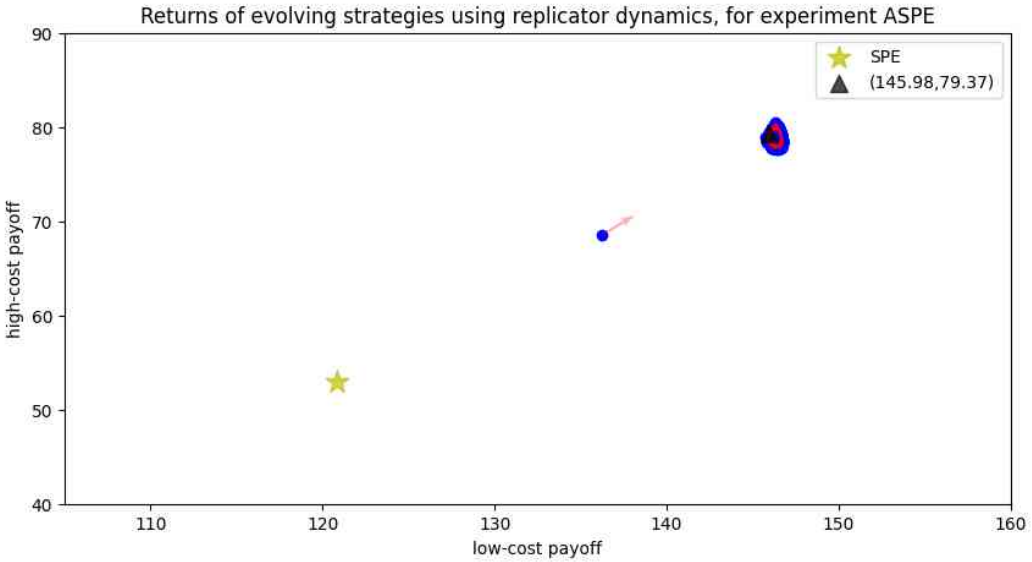


Figure 6.25.: Plot of returns for evolving strategies using replicator dynamics, for the ASPE experiment.

Conclusions to Part II

In this research, we studied the space of pricing strategies in a 25-round, asymmetric duopoly pricing game. The subgame perfect equilibrium (SPE) of this game was computed by Selten [62]; however, this equilibrium is too complex for general players to learn and represents highly competitive behaviour. The multi-round structure and infinite strategy space of this game make it intractable to analyse further using traditional game-theoretic methods. Observing the highly competitive strategies in the SPE led us to investigate the possible existence of other Nash equilibria that might be preferred by the players.

Our main objective was to use machine learning methods to approximate the equilibria of the pricing game that might be preferred by both players over the SPE, as they could yield higher payoffs for both. Seeing the potential for players to increase their payoffs through more cooperative behaviour in this pricing game, we examined whether our learning methods would lead newly trained strategies toward collusion in order to approach these preferred equilibria, and what behaviours these equilibria would encourage.

Therefore, we presented an empirical game-theoretic model to approximate the strategy space of the pricing game. We used the Policy-Space Response Oracles (PSRO) framework to iteratively improve this approximation, referred to as the "meta-game", by learning new best responses to the Nash equilibria. We used single-agent reinforcement learning algorithms to train new pricing strategies as best responses to the equilibrium strategies.

We began by focusing on the training process of best responses to prepare for the later implementation of the PSRO framework. Starting with simple policy-gradient RL algorithms, we examined different modelling specifications, algorithms, and techniques to improve performance. Among the basic RL algorithms examined, the REINFORCE algorithm with the myopic strategy as the baseline proved most effective in computing best responses against a pure deterministic opponent's strategy. However, we showed that these models did not capture the complexity of playing against mixed strategies of deterministic opponents.

As the learning agents face mixed strategies of opponents within the PSRO framework, we adopted more advanced RL algorithms: Soft Actor-Critic (SAC) and Proximal Policy Optimisation (PPO). After adjusting the modelling specification for these learning methods, we reached the goal we could not achieve with the previous, simpler learning algorithms. These trained pricing strategies captured patterns in opponents' behaviours and responded accordingly when facing a mixed strategy of deterministic pricing strategies. Thus, the settings for computing best responses were ready for the PSRO implementation.

We incorporated both these RL algorithms into the same meta-game in our implementation of the PSRO framework. This meant iteratively expanding the meta-game by adding the best responses trained separately by each learning algorithm in parallel.

Our first observation was the differences between strategies trained by these RL algorithms when they faced mixed strategies of stochastic opponents (strategies trained in previous iterations) within PSRO. Pricing strategies trained with SAC adapted their play to different opponents, but this was not observed in PPO-trained strategies, which played similar action paths against all opponents. Moreover, although SAC required longer training and exhibited a highly oscillatory learning path, it achieved strategies with higher returns due to its high exploration. SAC also produced noisier pricing behaviour, which is not conducive to collusion. PPO, in contrast, demonstrated a smoother learning path and faster training. The pricing behaviour was also smoother, which created more cooperative behaviour and helped collusion. This was expected due to the bound on the updates in the PPO objective function; however, it trained strategies with lower returns, as it either could not explore as much or the stochasticity of the opponents distracted the learning agent.

The more important observation concerned the equilibria of the evolving meta-game: the equilibrium payoffs did not get much higher than the SPE payoffs, and the payoffs did not converge to any point but kept oscillating within a cluster close to the SPE payoff. We did not observe signs of approaching an equilibrium with higher social welfare than the SPE, nor the emergence of cooperative behaviour between trained pricing strategies in the equilibria.

Since the learning agents did not naturally learn to collude, we experimented with the factors that might lead the agents toward more cooperative strategies in the meta-game.

Two factors proved effective for the emergence of cooperative behaviour between trained strategies: (1) introducing cooperative behaviour by including predefined strategies with varying levels of cooperation in the initial meta-game, and (2) guiding the PSRO process by selecting the Nash equilibrium of the meta-game with the highest social welfare.

Initially, we tested each of these factors separately: introducing cooperative strategies in the initial meta-game but using the Nash equilibrium most frequently found during the tracing procedure (not necessarily the one with the highest social welfare), and starting from random networks but guiding the PSRO process with the equilibrium that had the highest social welfare. In neither case did the equilibria converge to a better payoff point than the SPE payoff.

In our final set of experiments, we included both factors simultaneously. We set the equilibrium selection to be the one found by the tracing procedure with the highest social welfare. We then experimented with different sets of predefined strategies as the initial meta-game. These strategies varied in level of cooperation, from the highly cooperative myopic strategy to the highly competitive strategies of the SPE.

In all experiments starting with predefined strategies, the equilibria of the evolving meta-game converged, in terms of payoff, to a point considerably closer to the Pareto frontier of payoffs than the SPE.

To check the dynamic stability of these equilibria, we tested projected replicator dynamics on the meta-game, starting from a uniform distribution. In all experiments, the dynamics converged within this higher-payoff set of equilibria of the meta-game. This contrasts with experiments with a random network strategy as the initial meta-game, in which the dynamics never stabilised.

We further studied the strategies involved in these equilibria. Among low-cost strategies, the most frequent strategy in the equilibria was the predefined Guess2 strategy, which is weakly cooperative: it competes strongly for demand potential but allows cooperation once the desired demand share is reached. SAC- and PPO-trained strategies failed to reach this level of cooperation; SAC-trained strategies became too competitive, while PPO-trained agents were too cooperative, resulting in lower expected payoffs for both compared to the Guess2 strategy.

In contrast, among high-cost strategies, trained strategies had higher average probabilities than our predefined strategies. Notably, cooperative high-cost strategies were trained that enabled competitive low-cost agents to secure their desired demand while maintaining higher prices for mutual benefit. Both SAC and PPO were successful in training these high-cost strategies.

Furthermore, SAC mostly trained competitive strategies that adapted their play to the opponents, sought high demand, and kept prices unpredictable for the opponents. PPO, however, tended to train mostly cooperative strategies with stable prices and similar play against all opponents. Consequently, SAC-trained agents were more successful in training both low-cost and high-cost agents added to the meta-game, whereas PPO was only successful in training cooperative high-cost agents, as the low-cost ones failed to reach the payoff threshold of equilibria in the PSRO setting. This difference in low-cost strategies stems from the low-cost player's advantage in the pricing game, which allows them to compete aggressively to leverage their lower production cost to secure a higher share of demand potential.

The importance of the initial meta-game structure and equilibrium selection was evident in achieving these results. Omitting either factor prevented reaching higher-payoff equilibria. Thus, we conclude that exposing learning agents to strategies with varying levels of cooperation, combined with prioritising higher-social-welfare equilibria during the iterative approximation, was crucial to guiding the meta-game toward cooperation.

These experiments also raised some further questions. For instance, while our learning models successfully developed new, highly effective high-cost pricing strategies, the most frequent low-cost strategy in the equilibria remained the predefined Guess2 strategy. Despite employing advanced learning algorithms and techniques, the trained agents did not achieve a more sophisticated level of cooperation than this strategy. This raises the question of what additional techniques might help break down the complexity and make the development of smarter cooperative strategies more straightforward for the learning agents.

This research has numerous possible extensions, including varying the specifics of the learning process, modelling, or the pricing game itself. Our final experiments also revealed

improvements that could be made to our early models. The framework we developed can make examining these cases in future research more straightforward, as it automates the tracking of different aspects of experiments for further analysis. We explain these ideas under the directions for future research.

Directions for Future Work

A natural extension to our framework is to include a broader set of collusive strategies in the initial meta-game. As the variations of the Guess strategy emerged most frequently in the equilibria of the final experiments, applying Behavioural Cloning [20] to these strategies could provide the learning agents with a larger pool of strategies exhibiting similar levels of cooperation.

Another extension involves addressing the growth in meta-game size. As the PSRO process iteratively expands the meta-game, computing its Nash equilibria becomes increasingly complex. Removing trained pricing strategies that do not appear in the equilibria could reduce the meta-game size, thereby lowering computation costs and allowing the empirical process to continue toward better approximations of the hyper-game.

There are also many aspects of our research that invite further experimentation. Beginning with the pricing game itself, this project proved more complex than initially anticipated. Using machine learning to study a pricing game that is analytically challenging is both appealing and demanding. Defining step-by-step benchmarks to evaluate performance and gradually increasing complexity could simplify the process, especially when starting from simpler games, such as single-round repeated games, where more analytical results are available. Applying our experiments first to these simpler settings and then gradually increasing the complexity may yield valuable insights. Indeed, in some early experiments, we found studying a reduced form of the pricing game highly effective in monitoring the learning agents' progress.

Another interesting variation would be to remove the fixed number of rounds. Instead, the game could terminate probabilistically at each stage, with a fixed probability of termination. For comparability with our current results, this probability could be chosen so that the expected game length matches 25 rounds. Given that game length would then follow a geometric distribution, a termination probability of 0.04 would achieve this average. In this setup, the distribution of game lengths would implicitly discount later-round rewards, and the end-effect present in our fixed-length game would disappear. As a result, PPO might perform better in training strategies compared to our current setting. This change could also facilitate the emergence of cooperative strategies with punishment for deviation, as observed in repeated-game theory.

We also see potential in revisiting the early, simpler learning algorithms. Our empirical game model evolved alongside the project as we gained insights into engineering techniques that enhance learning. Several modelling improvements from the final experiments, such as the refined state representation, remain untested in these simpler algorithms. Given that the REINFORCE model with a myopic baseline performed well against fixed

deterministic opponents despite its simplicity, we suspect that incorporating the final modelling specifications could improve its performance. Running PSRO with these enhanced simpler algorithms might yield comparable results at lower computational cost.

In our implementation of the PSRO framework, we used Nash equilibria as the meta-strategy solver, as we were also interested in the hyper-game's Nash equilibria. However, experimenting with alternative meta-strategy solvers could lead the training in different and potentially beneficial directions. For example, Bighashdel, Wang, McAleer, Savani and Oliehoek [5] suggests that "projected replicator dynamics" within PSRO outperforms the Nash-based approach in preventing overfitting, while "minimum regret constraint profiles" are more effective in minimising regret.

Finally, we have made our framework implementation available [34] to facilitate experimentation with these ideas in future research. Our framework automates the pricing game model, data management, logging, and learning processes, and is compatible with advanced RL tools. It supports tracking the meta-game, analysing the behaviour of trained strategies, and evaluating learning performance.

7.

Appendix to Part II

7.1. Plots of the Final Trained Agents

In the following, we present plots for the price path, demand potential, actions, and rewards of the last two low-cost and two high-cost trained agents from our final experiments with different initial game setups. Each plot illustrates the behaviour of the learning agent against the two top opponents they were trained to compete with, with results shown for 10 trials against each opponent. The trials against each opponent are represented in different colours.

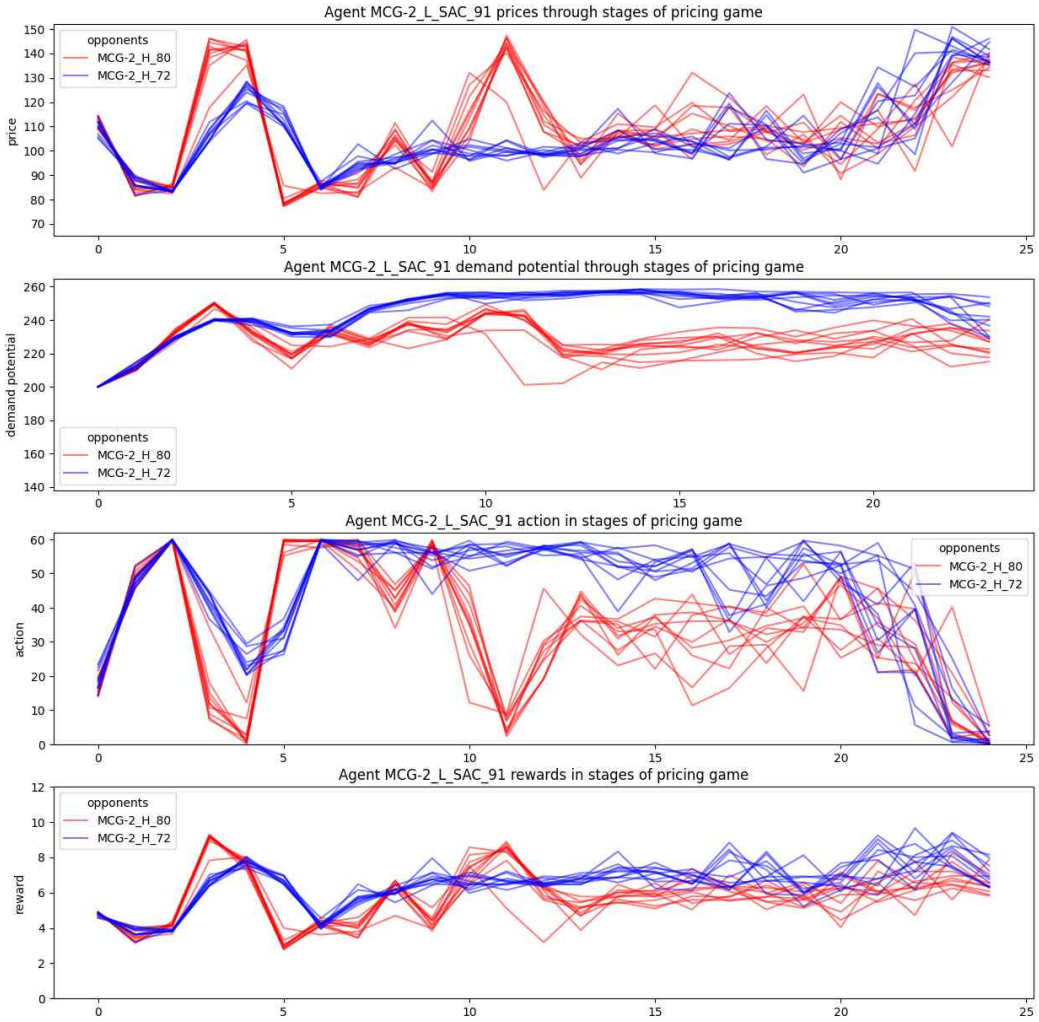


Figure 7.1.: Performance of the low-cost Player 91 in the pricing game, trained using the SAC algorithm in the MCG-2 experiment.

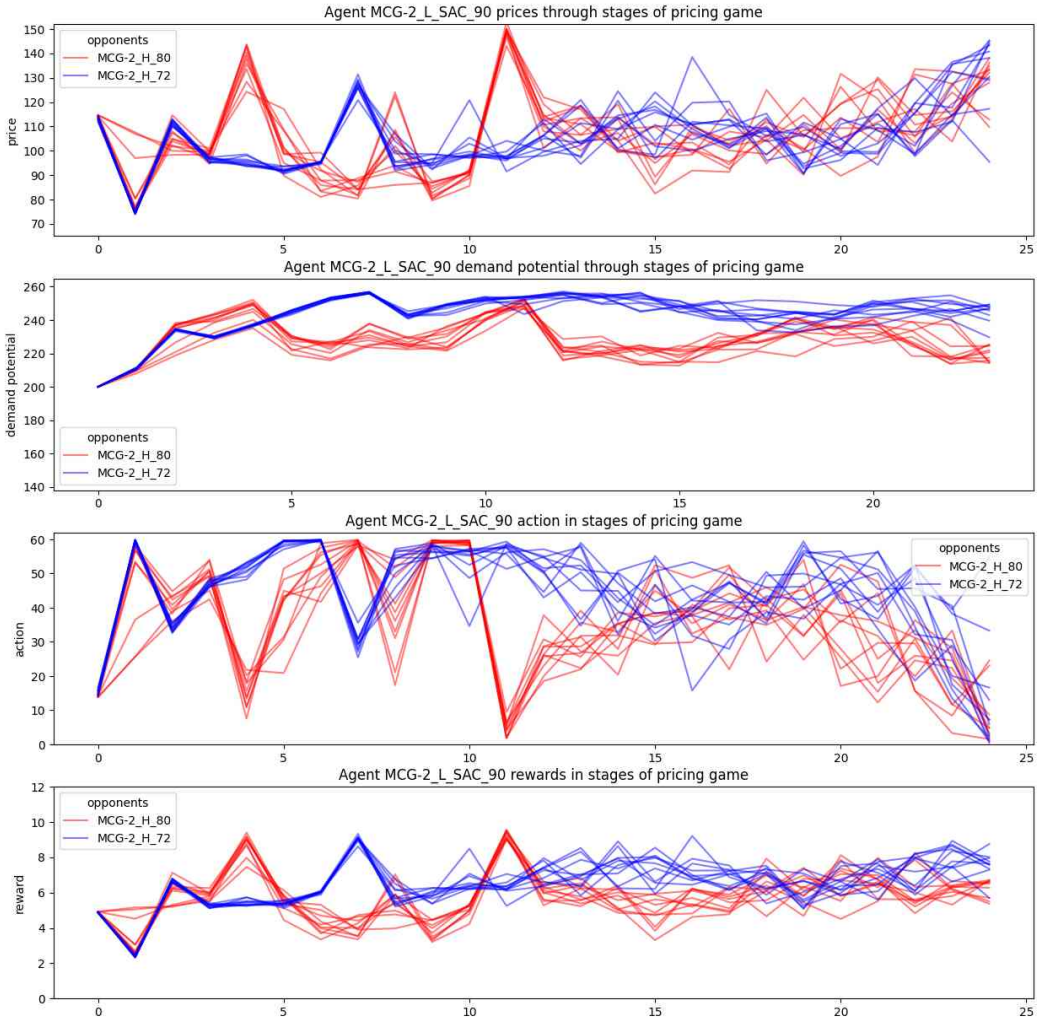


Figure 7.2.: Performance of the low-cost Player 90 in the pricing game, trained using the SAC algorithm in the MCG-2 experiment.

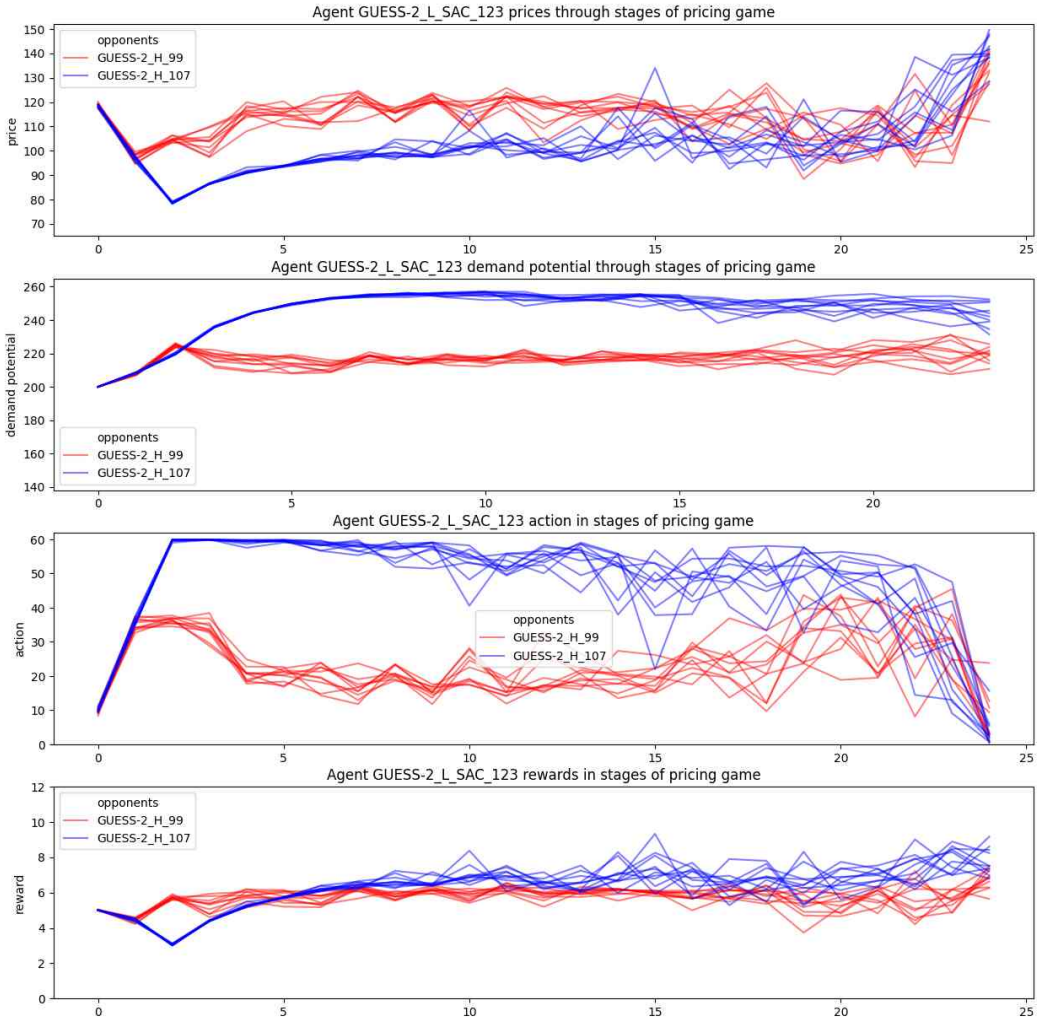


Figure 7.3.: Performance of the low-cost Player 123 in the pricing game, trained using the SAC algorithm in the GUESS-2 experiment.

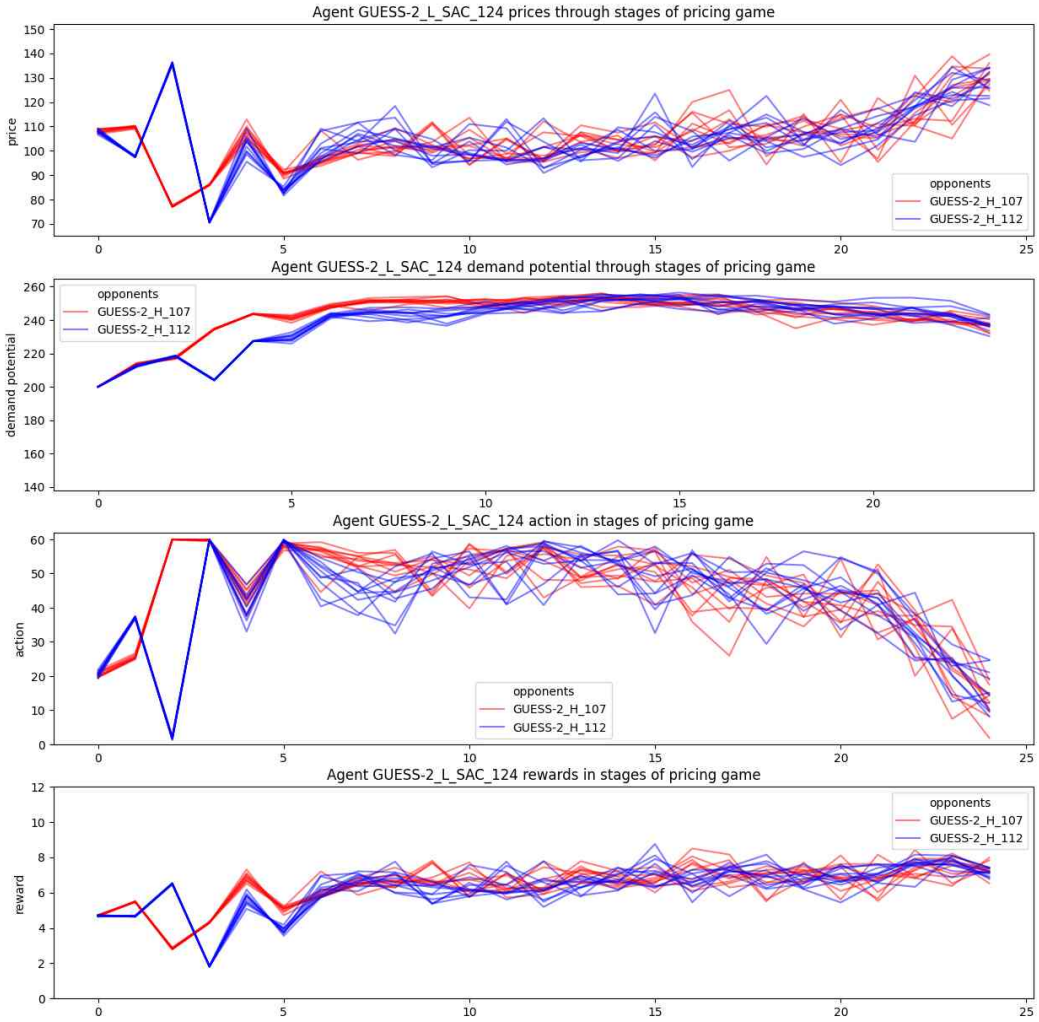


Figure 7.4.: Performance of the low-cost Player 124 in the pricing game, trained using the SAC algorithm in the GUESS-2 experiment.

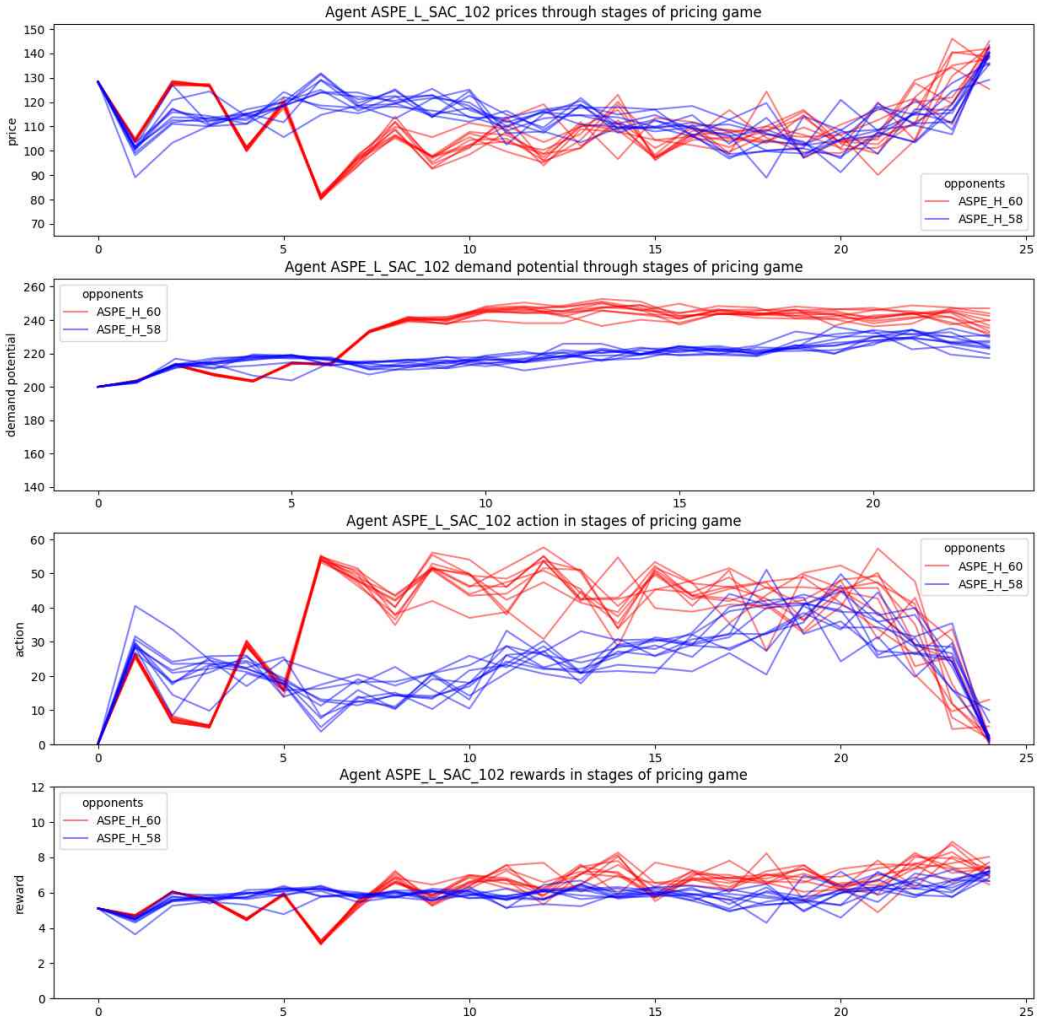


Figure 7.5.: Performance of the low-cost Player 102 in the pricing game, trained using the SAC algorithm in the ASPE experiment.

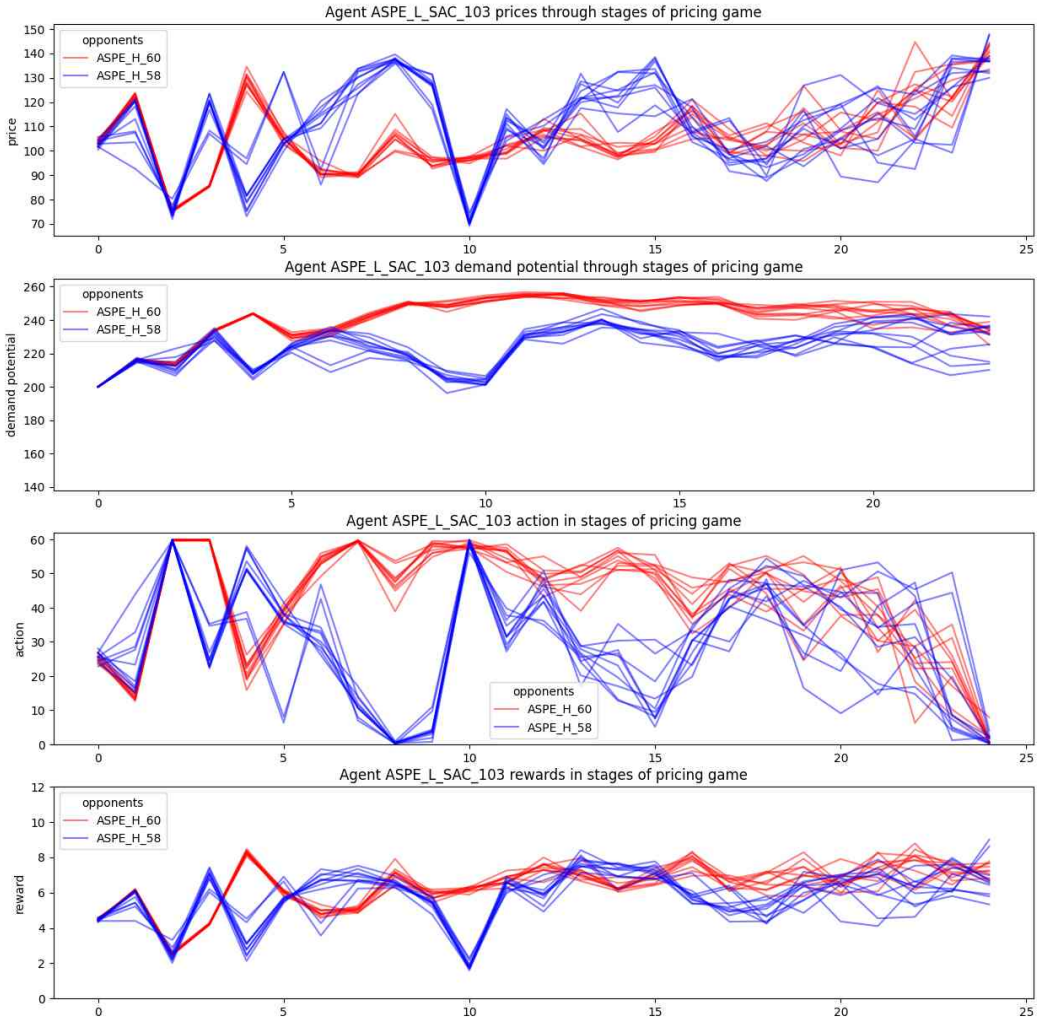


Figure 7.6.: Performance of the low-cost Player 103 in the pricing game, trained using the SAC algorithm in the ASPE experiment.

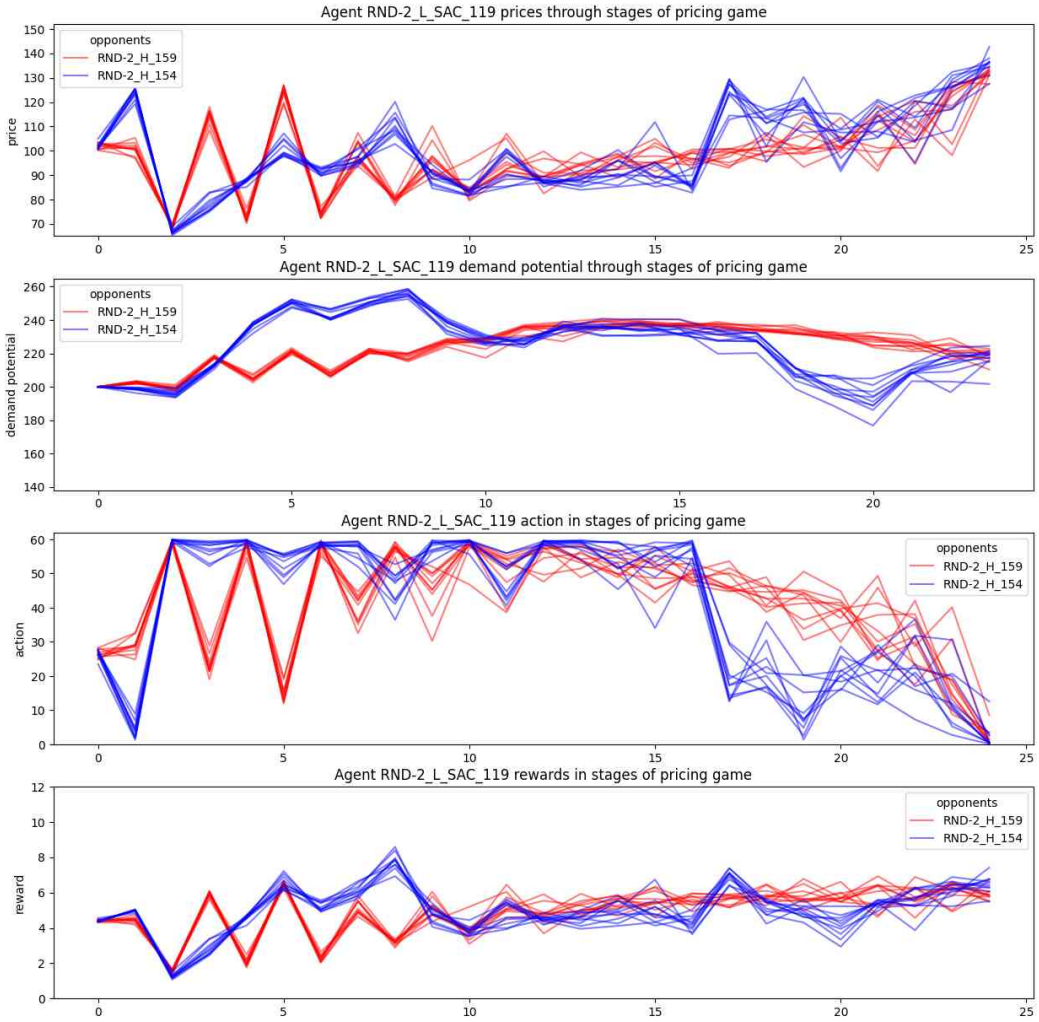


Figure 7.7.: Performance of the low-cost Player 119 in the pricing game, trained using the SAC algorithm in the RND-2 experiment.

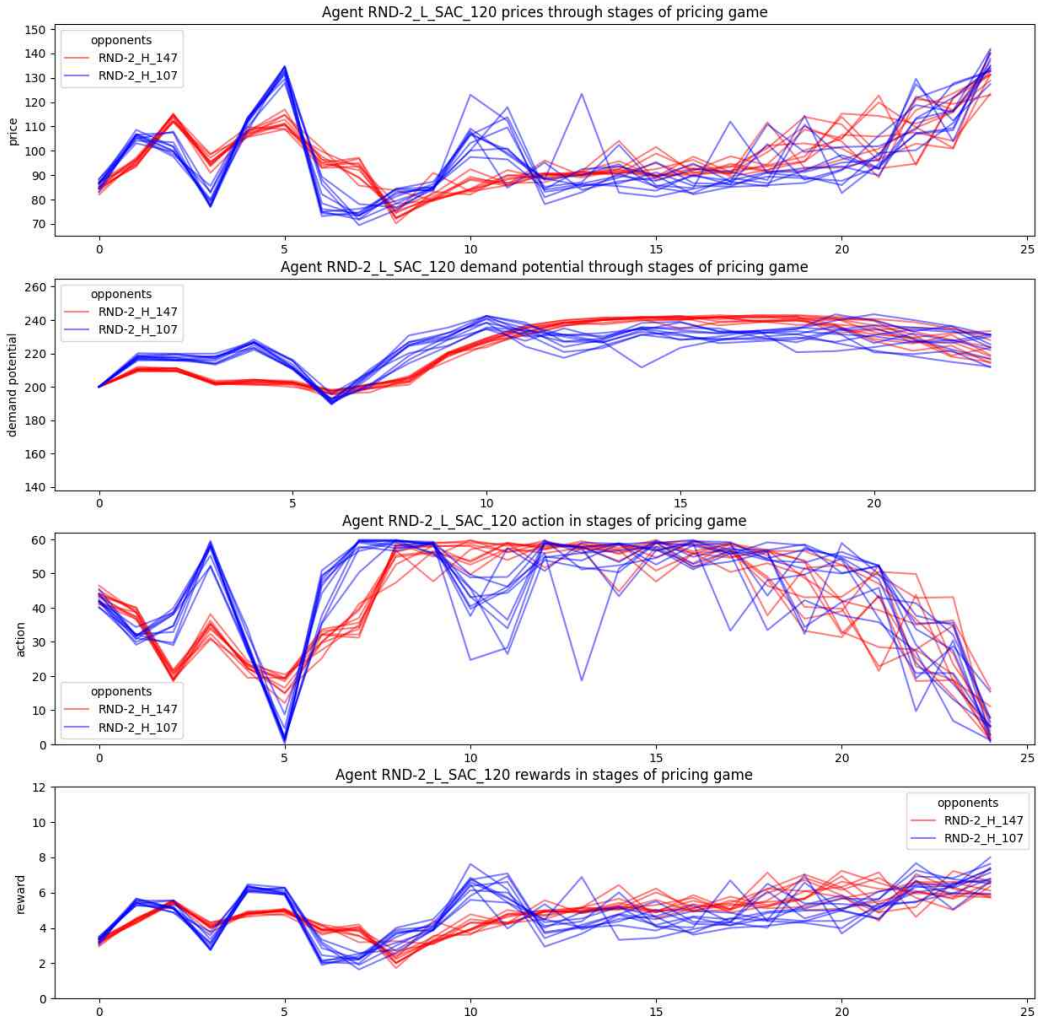


Figure 7.8.: Performance of the low-cost Player 120 in the pricing game, trained using the SAC algorithm in the RND-2 experiment.

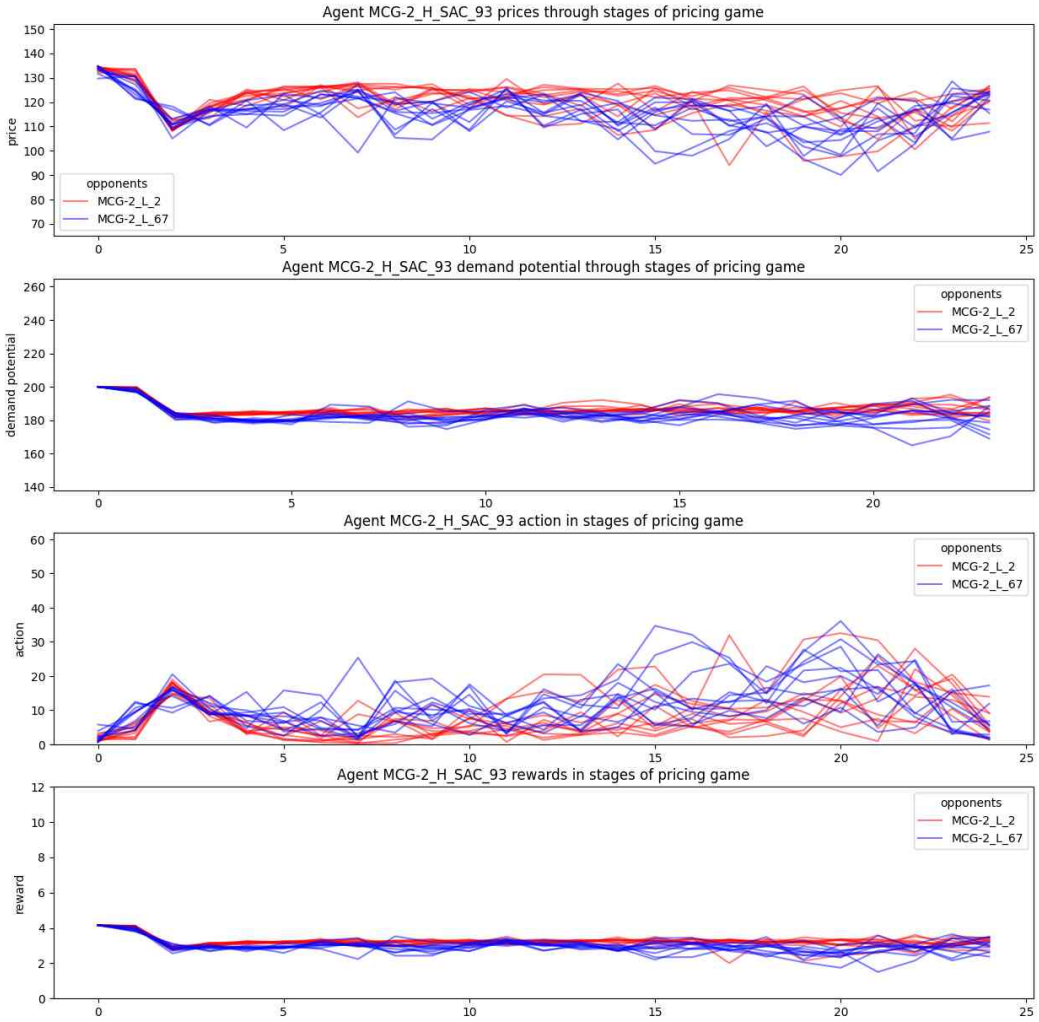


Figure 7.9.: Performance of the high-cost Player 93 in the pricing game, trained using the SAC algorithm in the MCG-2 experiment.

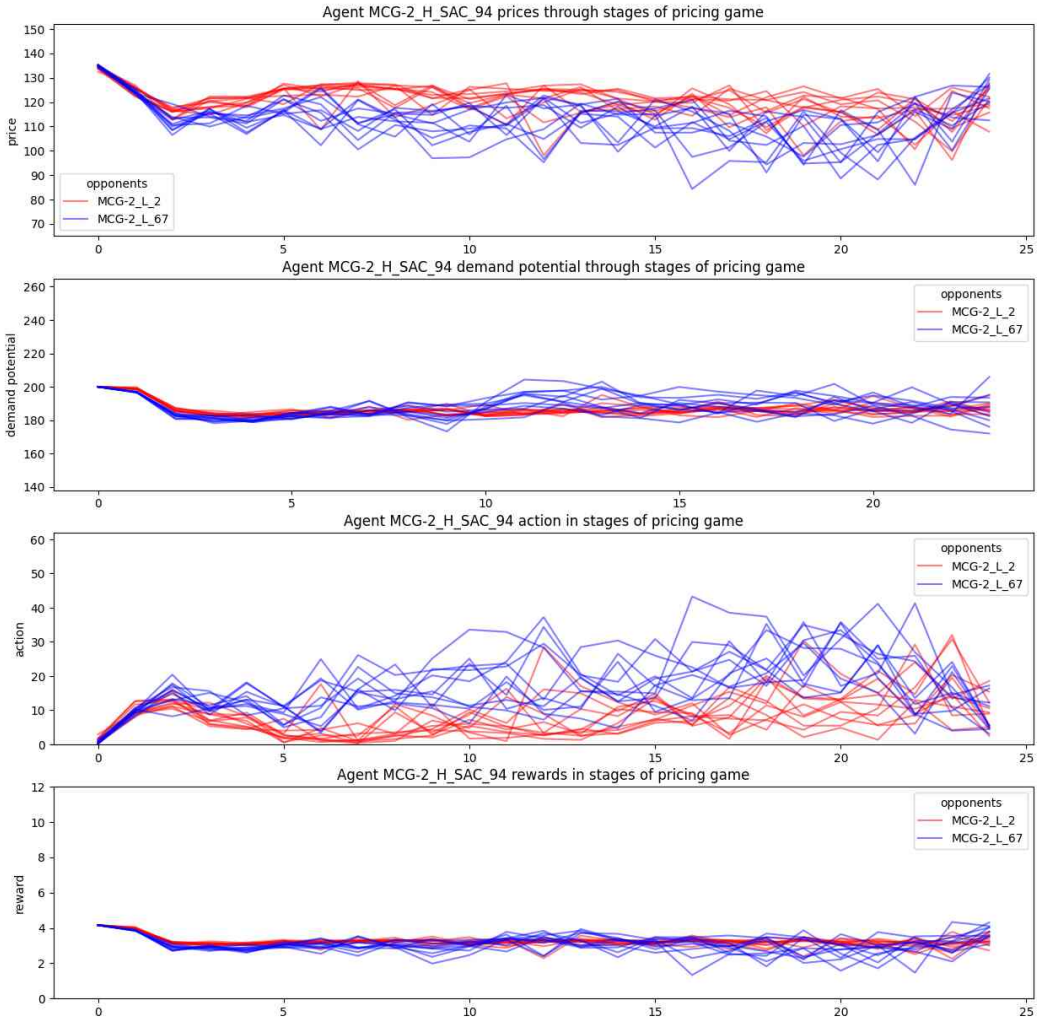


Figure 7.10.: Performance of the high-cost Player 94 in the pricing game, trained using the SAC algorithm in the MCG-2 experiment.

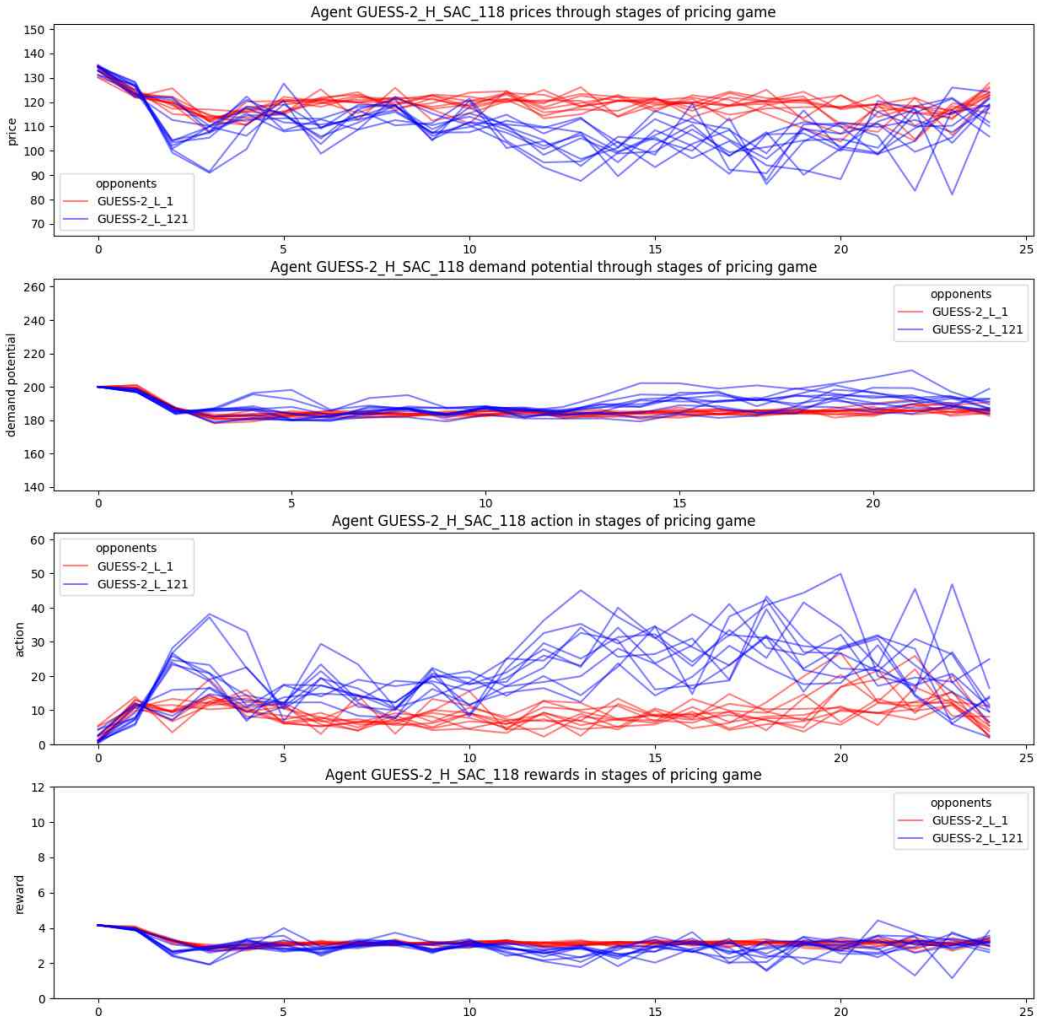


Figure 7.11.: Performance of the high-cost Player 118 in the pricing game, trained using the SAC algorithm in the GUESS-2 experiment.

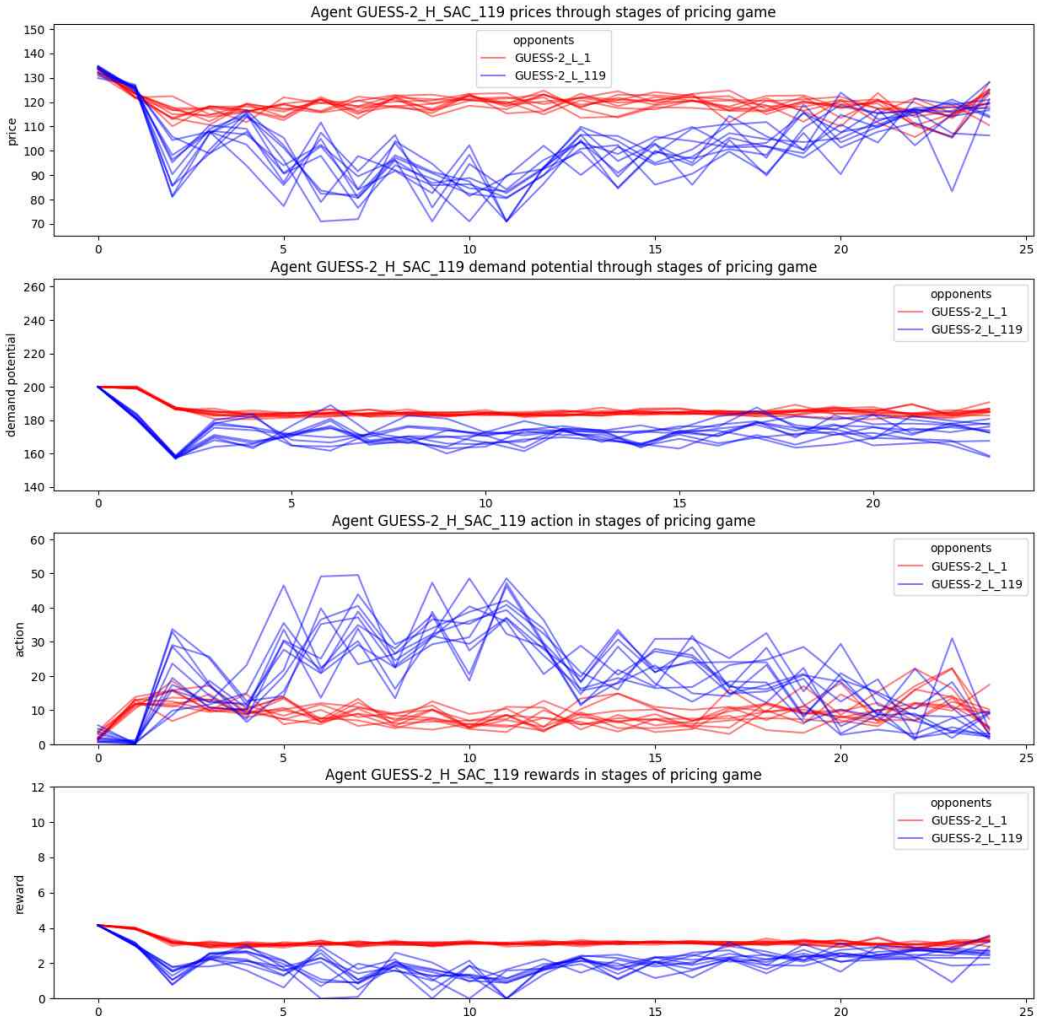


Figure 7.12.: Performance of the high-cost Player 119 in the pricing game, trained using the SAC algorithm in the GUESS-2 experiment.

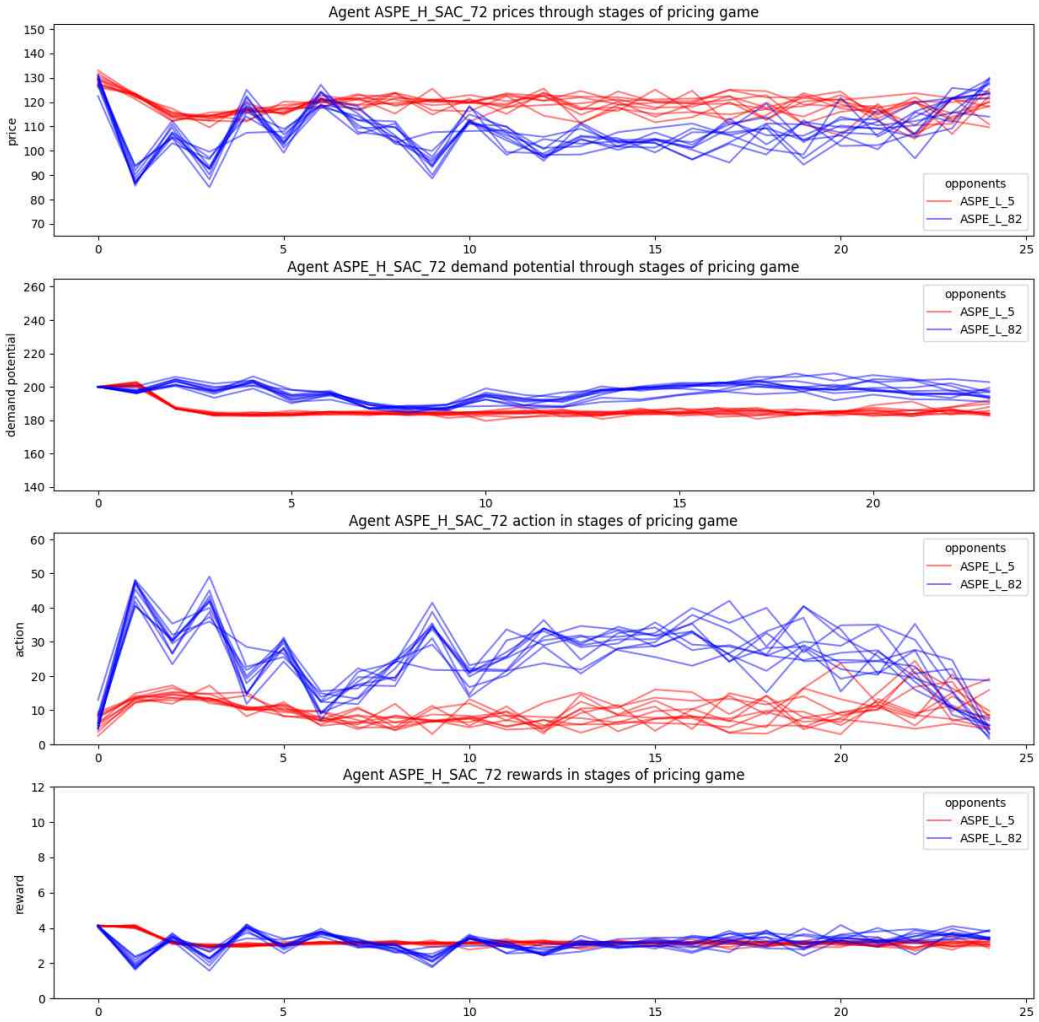


Figure 7.13.: Performance of the high-cost Player 72 in the pricing game, trained using the SAC algorithm in the ASPE experiment.

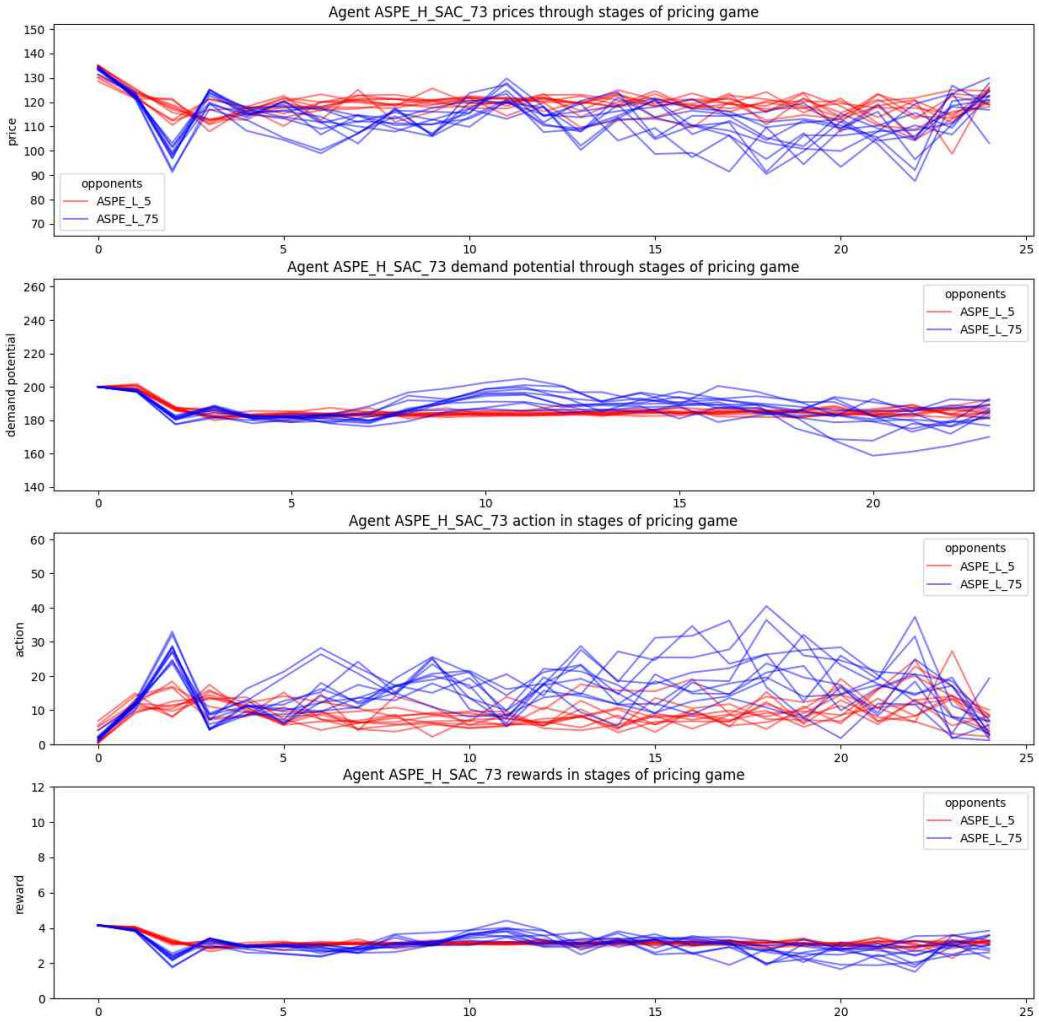


Figure 7.14.: Performance of the high-cost Player 73 in the pricing game, trained using the SAC algorithm in the ASPE experiment.

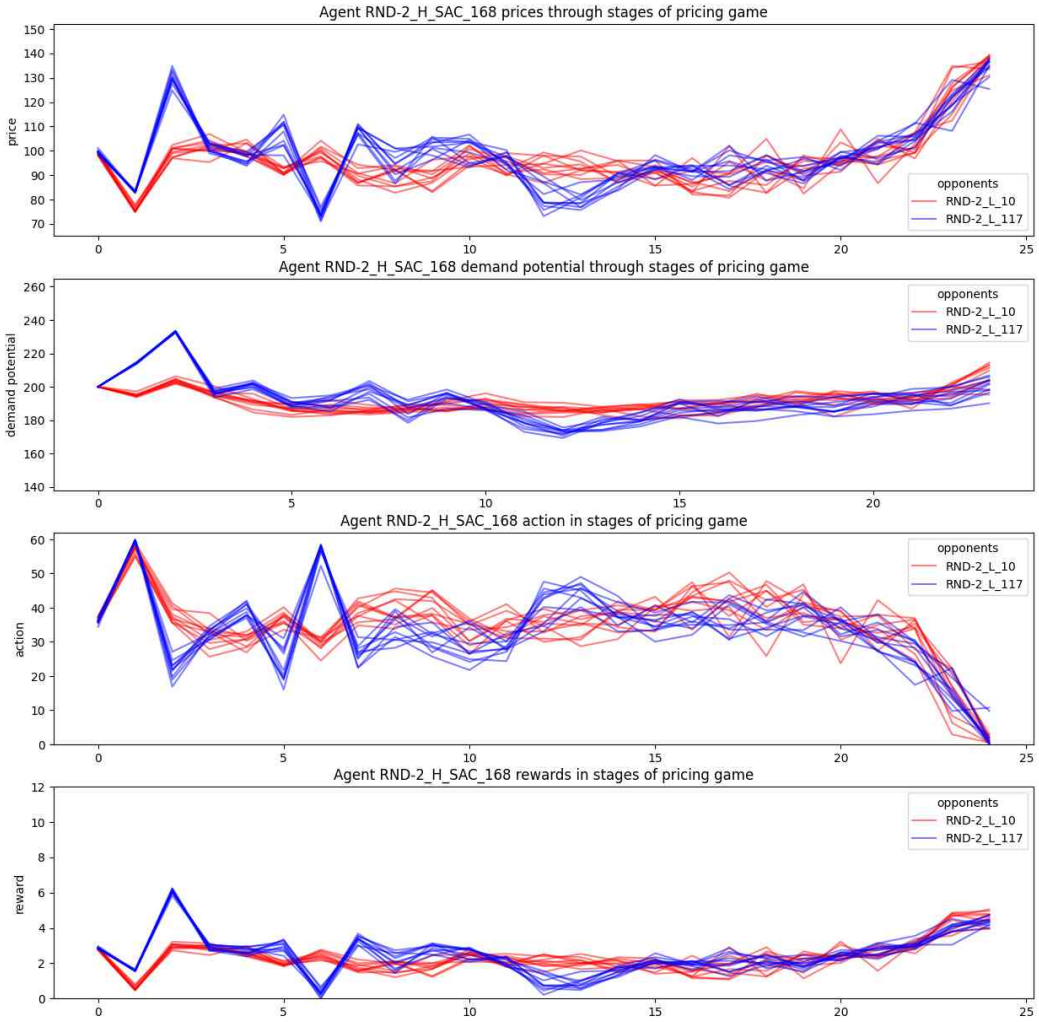


Figure 7.15.: Performance of the high-cost Player 168 in the pricing game, trained using the SAC algorithm in the RND-2 experiment.

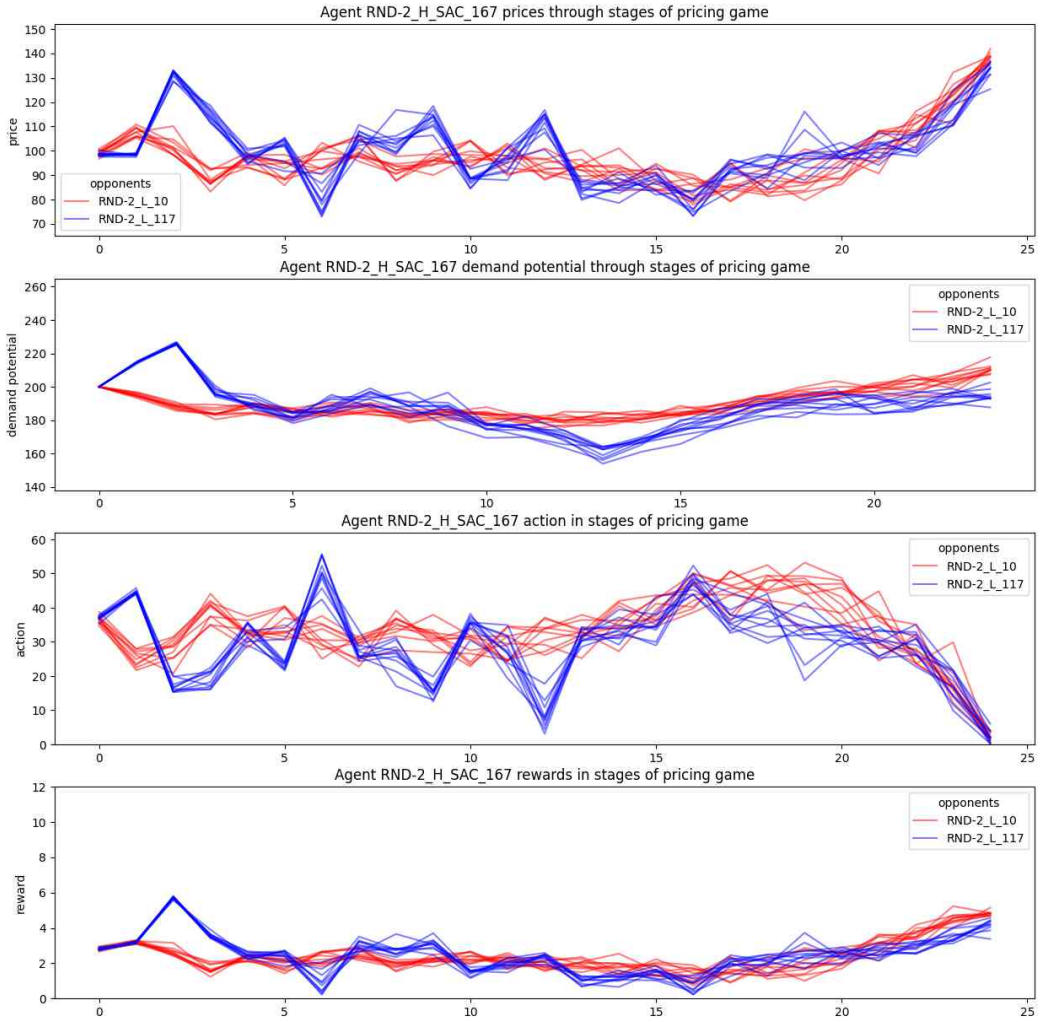


Figure 7.16.: Performance of the high-cost Player 167 in the pricing game, trained using the SAC algorithm in the RND-2 experiment.

The above agents were all trained using the SAC learning algorithm. To compare the behaviour of SAC and PPO, the following two plots display the trained low-cost and high-cost strategies among the last five strategies added to the meta-games in ASPE and GUESS-2, using PPO.

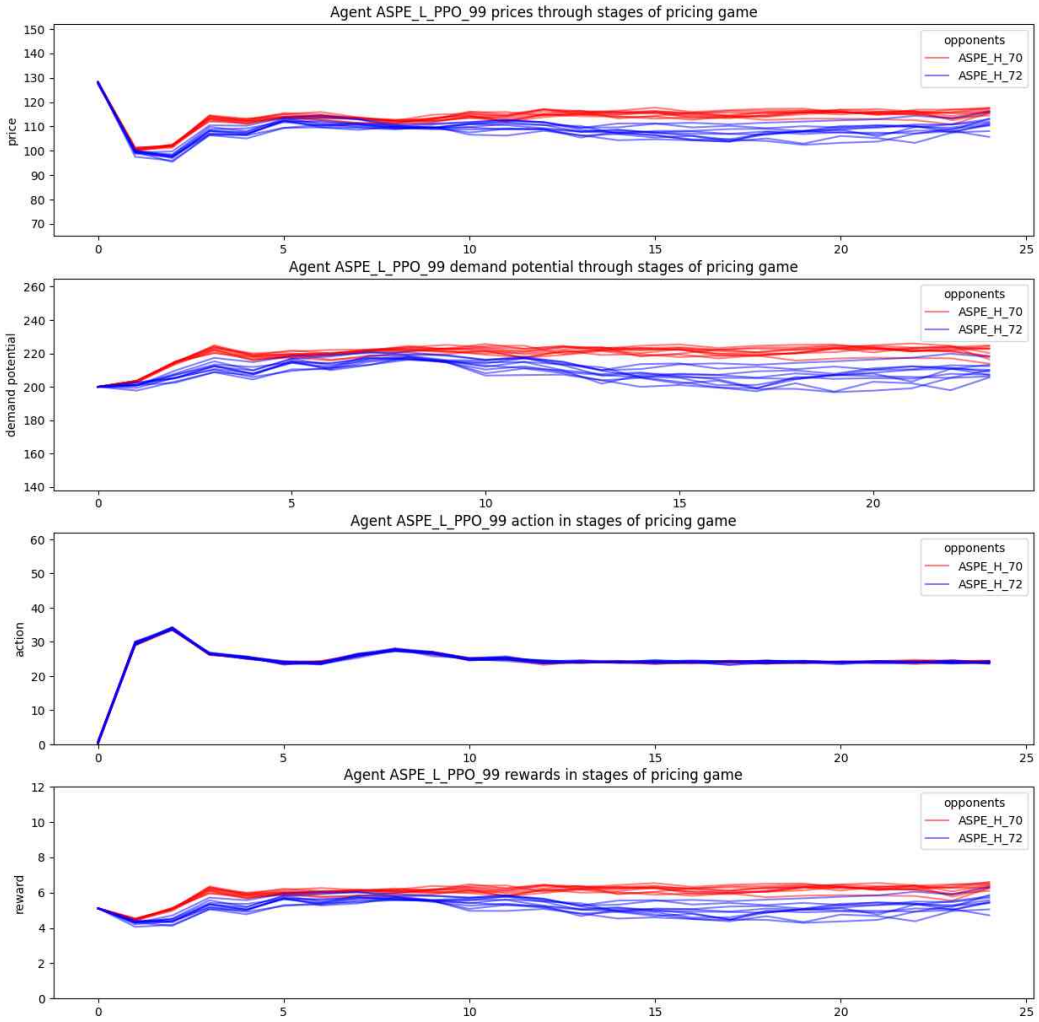


Figure 7.17.: Performance of the low-cost Player 99 in the pricing game, trained using the PPO algorithm in the ASPE experiment.

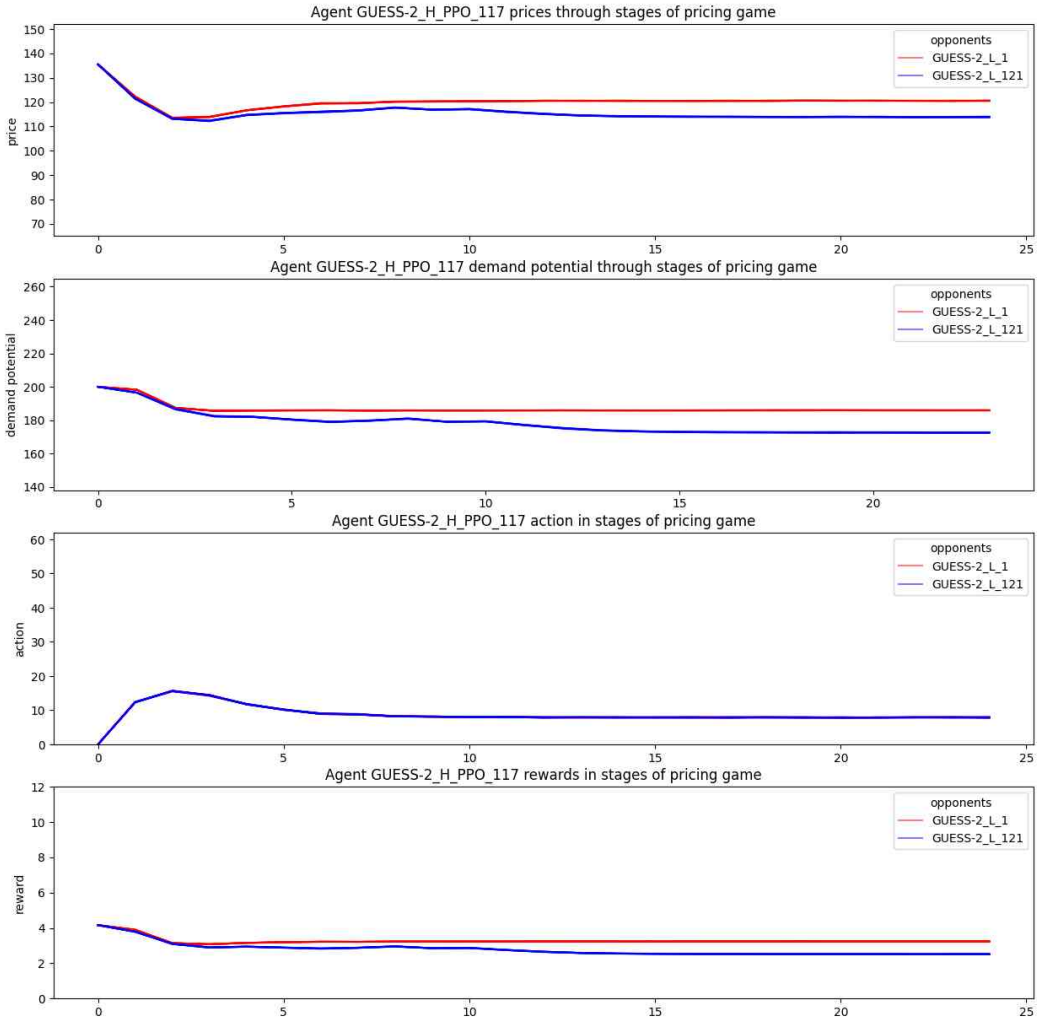


Figure 7.18.: Performance of the high-cost Player 117 in the pricing game, trained using the PPO algorithm in the GUESS-2 experiment.

7.2. Software Guide

We implemented the PSRO framework, as well as basic learning algorithms such as vanilla policy gradient, in Python 3.9. As the project progressed, we utilised more complex learning algorithms from the Stable-Baselines3 [54] library, which provides reliable implementations of various reinforcement learning algorithms.

In this section, we describe the structure of our implementation, the logging details, and how to extend and apply our framework [35] to other pricing games.

Environments

The pricing game we study is implemented as the environment in the reinforcement learning setting. We have defined two environment classes, *DisPricingGame* and *ConPricingGame*, both derived from the *gym.Env* class.

In our initial experiments, we considered the action space to be discrete, meaning agents' prices had discrete values.

$$\text{price} = \text{myopic} - n * \textbf{step}, \quad n \in \{0, 1, ..., [60/\text{step}]\}, \text{step} \in \mathbb{N} \tag{7.1}$$

We discretised the action space to reduce complexity for our basic learning methods. The class *DisPricingGame* was the environment applied in this setting.

An important parameter that influences behaviour in other classes using an environment is the *action_step*. This value is set to None for continuous environments and is set to an integer (*step* in 7.1) in discrete environments.

Later, as we utilised more complex learning algorithms, we decided to continue our experiments with a **continuous** action space over the same range. The class *ConPricingGame* is employed in this setting.

$$\text{price} = \text{myopic} - a, \quad a \in [0, 60]$$

Our pricing game is implemented by overriding the reset() and step() functions. After each step, the pricing game variables are updated, and the stage is incremented until the final stage, where the game resets, and the parameters are initialised again.

The constant values in our pricing game are defined as global variables in the "globals.py" file. At the start of our PSRO game, these global variables are initialised, and all other classes have access to them.

Hence, to experiment with other pricing games, if the new game's characteristics, such as the payoff function and stopping condition, are the same, we suggest only changing the values of global variables. However, if the pricing game is different, for example, if it has a different payoff function, a new environment class should be implemented using our implementation as a guide.

In the file *classes.py*, all classes used for implementing the PSRO framework are gathered.

ConPricingGame

This class, defined in *environments.py*, represents the pricing game used in our experiments. An instance of this class accepts a tuple of costs, an adversary's mixed strategy, and an integer called *memory*. The *costs* parameter specifies the production costs of the players in order. For example, if *costs* = $[57, 71]$, it means the production cost of the first player in the environment is 57, and for the second player, it is 71.

The adversary's mixed strategy defines the opponent's mixed strategy in this environment, against which we will train our agent. It is an instance of the *MixedStrategy* class. At each episode, a specific opponent (not mixed) is selected by drawing from this mixed strategy to train our agent. Finally, *memory* determines the length of the memory of previous prices that will be included in the state representation for training the agent.

BimatrixGame

The PSRO framework in our project is implemented as a bimatrix game between low-cost and high-cost strategies. This bimatrix game expands by adding new successful strategies after each iteration. We have implemented this game by defining the "BimatrixGame" class in the *classes.py* file. This class includes two lists, *low_strategies* and *high_strategies*, where each component is an instance of the **Strategy** class. Additionally, this class includes *matrix_A* and *matrix_B*, which store the payoffs of strategies playing against each other. For example, $A_{i,j}$ is the low-cost player's total payoff in the pricing game between low-cost strategy i and high-cost strategy j. $B_{i,j}$ represents the payoff for the high-cost player.

One of the important properties we implemented for this class is its ability to be serialised. When running the PSRO framework, we might need to stop and resume later due to various reasons, such as encountering an error or needing to continue on a different server. As the game size increases, rebuilding the matrix by having all strategies play against each other becomes time-consuming. Additionally, since the strategies are stochastic, the values of these matrices might not be the same in subsequent runs (despite averaging the results of multiple pricing game runs), which increases the error. Therefore, using the methods *save_game()* and *load_game()*, the game can be saved as a pickle file and loaded later to continue running. For this purpose, each strategy will be serialised as well.

Each entry of the payoff matrices is computed by calling the *update_matrix_entry(i,j)* method, in which the two strategies play the pricing game for *NUM_MATRIX_ITER* times, and the average payoff becomes the matrix entry. To add new low-cost strategies, the payoffs are computed, and the rows are added to both *matrix_A* and *matrix_B* using the *add_low_cost_row(row_A, row_B)* method. Similarly, for adding a column, *add_high_cost_col(col_A, col_B)* can be used.

Strategy

As we mentioned before, each low-cost and high-cost strategy in the PSRO setting is defined as an instance of the Strategy class. This class is serialisable as needed for serialising the BimatrixGame class. There are two types of strategies: static and sb3_models. The static strategies are those that are not trained through the learning process, and they are deterministic given the state of the game. All the strategies explained in Section 4.3.2 are of this type.

The models we train using reinforcement learning, employing neural networks as function approximators, are referred to as sb3_models. This name arises from the fact that these models are trained using stable-baseline3 implementations. However, in our earlier implementations, we used models by defining neural networks directly, and those models are referred to as neural_net.

The argument *model_or_func* can either accept the name of a trained model to be loaded later or, in the case of a static strategy, specify the static function that takes the environment, player index, and first-round price as inputs.

To compute the price that the strategy will play, you should call the method *play(env, player)* on an instance of this class. This method requires the environment and player index as inputs, where *player=0* indicates playing as the first player and *player=1* as the second player in the defined environment. It's important to note that the first player is not necessarily the low-cost player. For instance, if *env.costs* is defined as $[71, 57]$, the first player in the environment would be the high-cost player.

When the *play* method is called, if the strategy is of type *sb3model*, the model will be loaded. The model will then compute and return the price to play at the current state of the environment. In this method, the environment is assumed to be already initialised, and the method only computes the price without updating the environment.

The method *play_against(env, adversary)* can be used to play a full episode in the environment against another strategy, referred to as the adversary. By using this method, you can obtain the resulting payoff of two strategies competing against each other. The output will be a list of their payoffs at every stage.

MixedStrategy

The Nash equilibrium strategies of the PSRO game are not necessarily pure, so we need a structure to represent mixed equilibrium strategies. The *MixedStrategy* class represents a mixed strategy over a set of strategies. It is defined by specifying a list of strategies and a corresponding list of probabilities, called *strategy_probs*. In the environment, the adversary is an instance of this class. At each episode, one strategy is selected according to the defined probabilities. The method *choose_strategy()* draws a strategy from the mixed strategy based on these probabilities and returns it.

As mentioned, we need the *BimatrixGame* class to be serialisable, which means that strategies and mixed strategies must be serialisable as well. The method *copy_unload()* unloads the models loaded into each strategy within the current mixed strategy and returns a new instance of this class. This ensures the class is ready for serialisation.

BaseDataBase and DataBase

The *BaseDataBase* class serves as the foundational class from which the databases for each of the projects can be derived. The *DataBase* class inherits from *BaseDataBase* and is specific to this project and the pricing game.

Starting with the *BaseDataBase* class: when an instance of this class is initialised with a name as input, a SQLite database is created if it does not already exist. This class provides methods for executing insert, update, or delete queries, such as *execute_insert_query(query)* and *batch_insert_to_table(table, values_list)*. The method *execute_select_query(query, fetch_one)* can be used to select rows from the database, while *dataframe_select(query)* returns the selected rows as a DataFrame. These methods are then utilised in the derived classes for actions specific to the game.

In the *DataBase* class, we have defined all the methods specific to our pricing and PSRO games. Our database consists of the following tables:

- *trained_agents*: This table records all the details of any low-cost or high-cost agent that has been trained, whether or not they are added to the PSRO game. These details include the learning model, the number of episodes, the base agent from which parameters were preset, the expected payoff, the payoff threshold, the equilibrium used for training, and more.
- *agent_iters*: This table contains records of different iterations of each successfully trained agent against various adversaries. These data are recorded for ease of analysis and to have the data ready for running analytics models. However, these data can also be accessed by loading the trained model and playing against each adversary. As the PSRO game size increased later in our experiments, the database became too large, so we became selective about which iterations to record. Some of these details include the total return of both agents, the list of rewards at each stage of the pricing game, the actions played at each stage, and the prices and demand of each player.
- *game_equilibria*: This table stores the details of each equilibrium found by the tracing procedure in each round of the PSRO game, whether or not these equilibria were used

in the learning process. Some columns in this table include the game size, low-cost and high-cost strategies, the expected payoff of each agent, whether the equilibrium was used, and how many low-cost and high-cost agents were trained and added to the game using this equilibrium.

- *strategy_average_probs*: This table records the average probability of each low-cost and high-cost strategy appearing in the equilibria used for training agents. This table helps us identify common agents in the equilibria and better analyse the game.

When an instance of this class is initialised, the SQLite database and its tables are created if they do not already exist. The exact structure of the database is shown in Figure 7.19.

The names of the tables and their columns are defined as constant parameters in the *DataBase* class. Additionally, methods are provided for inserting new rows into each table or updating existing ones when necessary, such as for the *strategy_average_probs* table.

Logging

When the PSRO framework is executed, various pieces of game-related information are stored in different files within the base folder. This section provides an explanation of each of these files and their contents.

During the execution of the main code, the experiment is identified by the *job_name* variable. This value is used in the logging files' names to specify the experiment they belong to.

"error.log"

This file records any errors that occur during the execution of the project. The Python "logging" library is utilised for this purpose, with the main code wrapped in a "try-except" block that logs any exceptions raised into this file. This feature is particularly important when running the project on an external server, as it allows for effective tracking and troubleshooting of issues.

"game_[job_name].txt"

The latest bimatrix game in the PSRO setting is recorded in the "game_[job_name].txt" file. With each round of the meta-game, whenever new strategies are added, this file is updated with the new bimatrix game. It is crucial not to delete this file, as it is necessary for restoring the last game from the previous stopping point.

The file begins with the number of rows and columns, separated by a space, on the first line. Following this, the payoff matrix A, representing low-cost strategies, is printed. The columns of each row of the matrix are separated by a tab, with each new row starting on a new line. After matrix A, the payoff matrix B, corresponding to high-cost strategies, is printed in the same format.

agents_iters		
id	integer	"id" integer
agent_id	integer	"agent_id" integer NOT NULL
adv	text	"adv" text NOT NULL
agent_return	text	"agent_return" text
adv_return	text	"adv_return" text
agent_rewards	text	"agent_rewards" text
adv_rewards	text	"adv_rewards" text
actions	text	"actions" text
agent_prices	text	"agent_prices" text
adv_prices	text	"adv_prices" text
agent_demands	text	"agent_demands" text
adv_demands	text	"adv_demands" text
game_equilibria		
id	integer	"id" integer
time	text	"time" text
game_size	text	"game_size" text NOT NULL
freq	real	"freq" real
low_strategy	text	"low_strategy" text NOT NULL
high_strategy	text	"high_strategy" text NOT NULL
low_payoff	real	"low_payoff" real NOT NULL
high_payoff	real	"high_payoff" real NOT NULL
used	INTEGER	"used" INTEGER DEFAULT 0
num_new_low	integer	"num_new_low" integer DEFAULT 0
num_new_high	integer	"num_new_high" integer DEFAULT 0
strategy_average_probs		
id	integer	"id" integer
time	text	"time" text
game_size	text	"game_size" text
equi_count	integer	"equi_count" integer
last_equi_id	integer	"last_equi_id" integer
cost	integer	"cost" integer
strategy_probs	text	"strategy_probs" text
trained_agents		
id	integer	"id" integer
time	text	"time" text
name	text	"name" text NOT NULL
base_agent	text	"base_agent" text DEFAULT NULL
n_ep	integer	"n_ep" integer NOT NULL
cost	integer	"cost" integer NOT NULL
mixed_adv	text	"mixed_adv" text NOT NULL
alg	text	"alg" text NOT NULL
seed	integer	"seed" integer
num_procs	integer	"num_procs" integer DEFAULT 1
running_time	integer	"running_time" integer
return_std	real	"return_std" real
expected_payoff	real	"expected_payoff" real
payoff_treshhold	real	"payoff_treshhold" real
added	integer	"added" integer
equi_id	integer	"equi_id" integer

Figure 7.19.: The structure of the database, including tables and columns of the recorded data.

games folder

At each iteration of the meta-game, the bimatrix game representation is saved in this folder. The game representation is as explained in the last section. These files are not updated but are saved separately to track the changes in the bimatrix game at each iteration.

Moreover, in the last lines of the text files, the names of all the strategies in the bimatrix game are printed.

"game_[job_name].pickle"

To continue the meta-game from the last stopping point, we need to regularly (at the end of each iteration) save the serialised bimatrix class instance. When we start running the game again, we load the last saved file and continue from there. Saving and loading of the game are done using the “pickle" library. This pickle file includes the matrix of the game as well as the serialised strategies for both players.

This file is crucial for continuing the process from the last point; otherwise, the bimatrix game would have to be reinitialised, and all entries would need to be recomputed. For large games, this can take hours.

"progress_[job_name].txt"

In this file, all the progress is recorded. At each iteration, all the equilibria found for the bimatrix game are written. For each equilibrium, all the new strategies trained and added to the game are recorded along with their details, such as the learning algorithm and expected payoff. The progress file will not be reset, and whenever the game is paused and continued, the new logs will be appended to the previous logs, allowing all details to be tracked in this file.

The content of this file is for us to follow the progress of the game. The running code does not use this data; it is only logged.

Bibliography

[1] Askenazi-Golan, G., Cecchelli, D. M. and Plumb, E., *Reinforcement Learning, Collusion, and the Folk Theorem*, 2024, arXiv: 2411.12725 [cs.GT].

[2] Aumann, R. J., Correlated Equilibrium as an Expression of Bayesian Rationality, *Econometrica* 55.1 (1987), 1–18.

[3] Avis, D., Rosenberg, G. D., Savani, R. and von Stengel, B., Enumeration of Nash Equilibria for Two-Player Games, *Economic Theory* 42.1 (2010), 9–37.

[4] Bertrand, Q., Duque, J., Calvano, E. and Gidel, G., *Q-learners Can Provably Collude in the Iterated Prisoner's Dilemma*, https://arxiv.org/pdf/2312.08484, arXiv:2312.08484 [cs.LG], 2024.

[5] Bighashdel, A., Wang, Y., McAleer, S., Savani, R. and Oliehoek, F. A., *Policy Space Response Oracles: A Survey*, Accessed: 2024-12-26, 2024, arXiv: 2403.02227.

[6] Brightwell, G., Kenyon, C. and Paugam-Moisy, H., 'Multilayer Neural Networks: One or Two Hidden Layers?', *Proceedings of the Conference on Neural Information Processing Systems*, 1996.

[7] Brown, Z. Y. and MacKay, A., Competition in Pricing Algorithms, *American Economic Journal: Microeconomics* 13.3 (2021), 270–305.

[8] Cai, Y., Candogan, O., Daskalakis, C. and Papadimitriou, C., Zero-sum Polymatrix Games: A Generalization of Minmax, *Mathematics of Operations Research* 41.2 (2016), 648–655.

[9] Calvano, E., Calzolari, G., Denicolo, V. and Pastorello, S., Artificial Intelligence, Algorithmic Pricing, and Collusion, *American Economic Review* 110.10 (2020), 3267–3297.

[10] Chen, X., Deng, X. and Teng, S.-H., Settling the Complexity of Computing Two-Player Nash Equilibria, *Journal of the ACM* 56.3 (2009), Article 14.

[11] Chin, H., Parthasarathy, T. and Raghavan, T., Structure of Equilibria in N-person Non-Cooperative Games, *International Journal of Game Theory* 3.1 (1974), 1–19.

[12] Daskalakis, C., Goldberg, P. W. and Papadimitriou, C. H., The Complexity of Computing a Nash Equilibrium, *SIAM Journal on Computing* 39.1 (2009), 195–259.

[13] Datta, R. S., Finding all Nash Equilibria of a Finite Game Using Polynomial Algebra, *Economic Theory* 42.1 (2010), 55–96.

[14] DeMichelis, S. and Germano, F., On the Indices of Zeros of Nash Fields, *Journal of Economic Theory* 94.2 (2000), 192–217.

[15] Demichelis, S. and Ritzberger, K., From Evolutionary to Strategic Stability, *Journal of Economic Theory* 113.1 (2003), 51–75.

[16] Douglas, C., Provost, F. and Sundararajan, A., *Naive Algorithmic Collusion: When Do Bandit Learners Cooperate and When Do They Compete?*, 2024, arXiv: 2411.16574.

[17] *Fabian: High-Performance Computing, London School of Economics*, Accessed: 2024-12-26, 2023, URL: https://info.lse.ac.uk/staff/divisions/dts/services/fabian/Fabian.

[18] Fujimoto, S., Hoof, H. van and Meger, D., *Addressing Function Approximation Error in Actor-Critic Methods*, 2018.

[19] Game Theory Explorer, *Game Theory Explorer Software*, http://gametheoryexplorer.org/, Accessed: 2024-12-26, 2024.

[20] Gleave, A., Taufeeque, M., Rocamonde, J., Jenner, E., Wang, S. H., Toyer, S., Ernestus, M., Belrose, N., Emmons, S. and Russell, S., *imitation: Clean Imitation Learning Implementations*, arXiv:2211.11972v1 [cs.LG], 2022, arXiv: 2211.11972 [cs.LG].

[21] Govindan, S. and Wilson, R., Computing Nash Equilibria by Iterated Polymatrix Approximation, *Journal of Economic Dynamics and Control* 28.7 (2004), 1229–1241.

[22] Gül, F., Pearce, D. and Stacchetti, E., A Bound on the Proportion of Pure Strategy Equilibria in Generic Games, *Mathematics of Operations Research* 18.3 (1993), 548–552.

[23] Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S., *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, 2018, arXiv: 1801.01290.

[24] Harsanyi, J. C., Oddness of the Number of Equilibrium Points: A New Proof, *International Journal of Game Theory* 2.1 (1973), 235–250.

[25] Harsanyi, J. C. and Selten, R., *A General Theory of Equilibrium Selection in Games*, The MIT Press, 1988.

[26] Harsanyi, J. C., The Tracing Procedure: a Bayesian Approach to Defining a Solution for Noncooperative Games, *International Journal of Game Theory* 4.2 (1975), 61–94.

[27] Hasselt, H. v., 'Double Q-Learning', *Proceedings of the 24th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'10, 2010, 2613–2621.

[28] Hertling, C. and Vujic, M., *Maximal Number of Mixed Nash Equilibria in Generic Games where Each Player has Two Pure Strategies*, Accessed: 2024-12-26, 2024, arXiv: 2412.17890.

[29] Hertling, C. and Vujic, M., *Mixed Extensions of Generic Finite Games Embedded into Products of Real Projective Spaces*, Accessed: 2024-12-26, 2024, arXiv: 2412.17638.

[30] Hofbauer, J., *Some Thoughts on Sustainable/Learnable Equilibria*, Plenary lecture, Conference of the 15th IMGTA (Italian Meeting on Game Theory and Applications), Urbino, Italy, July 9–12, 2003. Accessed: 2024-12-26, URL: https://homepage.univie.ac.at/josef.hofbauer/03sustain.pdf.

[31] Howson Jr, J. T., Equilibria of Polymatrix Games, *Management Science* 18.5, Part I (1972), 312–318.

[32] Ickstadt, C., Theobald, T. and von Stengel, B., *A Stable-Set Bound and Maximal Numbers of Nash Equilibria in Bimatrix Games*, Accessed: 2024-12-26, 2024, arXiv: 2411.12385.

[33] Jacot, A., Gabriel, F. and Hongler, C., 'Neural Tangent Kernel: Convergence and Generalization in Neural Networks', *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018, 8580–8589.

[34] Jahani, S., *2×2×2 Games: Equilibrium Enumeration and Graphical Representation*, Accessed: 2025-07-13, GitHub, 2025, URL: https://github.com/sahar-jahani/2x2x2_Games_Equilibria.

[35] Jahani, S., *Pricing Game Strategy Evolution via PSRO and Reinforcement Learning*, Accessed: 2025-07-13, GitHub, 2025, URL: https://github.com/sahar-jahani/PSRO_RL_Pricing.

[36] Jahani, S. and von Stengel, B., 'Automated Equilibrium Analysis of $2 \times 2 \times 2$ Games', *Algorithmic Game Theory, 15th International Symposium, SAGT 2022*, ed. by P. Kanellopoulos, M. Kyropoulou and A. Voudouris, vol. 13584, Lecture Notes in Computer Science, Springer, Cham, 2022, 223–237.

[37] Keser, C., *Experimental Duopoly Markets with Demand Inertia: Game-Playing Experiments and the Strategy Method*, vol. 391, Lecture Notes in Economics and Mathematical Systems, Berlin: Springer Verlag, 1992.

[38] Keser, C., Some Results of Experimental Duopoly Markets with Demand Inertia, *The Journal of Industrial Economics* 41.2 (1993), 133–151.

[39] Kingma, D. P. and Ba, J., *Adam: A Method for Stochastic Optimization*, Accessed: 2024-12-26, 2017, arXiv: 1412.6980.

[40] Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Perolat, J., Silver, D. and Graepel, T., 'A Unified Game-Theoretic Approach to M Reinforcement Learning', *Advances in Neural Information Processing Systems*, 2017, 4193–4206.

[41] Lemke, C. E., Bimatrix Equilibrium Points and Mathematical Programming, *Management Science* 11.7 (1965), 681–689.

[42] Lemke, C. E. and Howson Jr, J. T., Equilibrium Points of Bimatrix Games, *Journal of the Society for Industrial and Applied Mathematics* 12.2 (1964), 413–423.

[43] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D., *Continuous Control with Deep Reinforcement Learning*, 2019, arXiv: 1509.02971.

[44] McKelvey, R. D. and McLennan, A., The Maximal Number of Regular Totally Mixed Nash Equilibria, *Journal of Economic Theory* 72.2 (1997), 411–425.

[45] McMahan, H. B., Gordon, G. J. and Blum, A., Planning in the Presence of Cost Functions Controlled by an Adversary, *Carnegie Mellon* (2003).

[46] Miklós-Thal, J. and Tucker, C., AI, Algorithmic Pricing, and Collusion, *CPI Antitrust Chronicle* (2024).

[47] Nash, J., Non-Cooperative Games, *The Annals of Mathematics* 54.2 (1951), 286–295.

[48] Neumann, J. von and Morgenstern, O., *Theory of Games and Economic Behavior*, Princeton University Press, 1944.

[49] Nobel Foundation, *Nobel Prize 1994 – Economic Sciences: John C. Harsanyi, John F. Nash Jr., Reinhard Selten*, Accessed: 2024-12-26, 1994, URL: https://www.nobelprize.org/prizes/economic-sciences/1994/selten/facts/.

[50] Ockenfels, A. and Moldovanu, B., *Reinhard Selten: Pioneering Analyst of Rationality and Human Behaviour*, Accessed: 2024-12-26, 2016, URL: https://cepr.org/voxeu/columns/reinhard-selten-pioneering-analyst-rationality-and-human-behaviour.

[51] OpenAI Contributors, *Proximal Policy Optimization (PPO)*, Accessed: 2024-12-26, OpenAI, 2024, URL: https://spinningup.openai.com/en/latest/algorithms/ppo.html.

[52] OpenAI Contributors, *Spinning Up in Deep Reinforcement Learning: Soft Actor-Critic (SAC)*, Accessed: 2024-12-26, OpenAI, 2024, URL: https://spinningup.openai.com/en/latest/algorithms/sac.html.

[53] PyTorch Contributors, *Adam Optimizer: torch.optim.Adam*, Accessed: 2024-12-26, PyTorch Foundation, 2024, URL: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html.

[54] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M. and Dormann, N., Stable-Baselines3: Reliable Reinforcement Learning Implementations, *Journal of Machine Learning Research* 22.268 (2021), 1–8.

[55] Ritzberger, K., The Theory of Normal Form Games from the Differentiable Viewpoint, *International Journal of Game Theory* 23 (1994), 207–236.

[56] Rosenmüller, J., On a Generalization of the Lemke–Howson Algorithm to Non-cooperative N-person games, *SIAM Journal on Applied Mathematics* 21.1 (1971), 73–79.

[57] Savani, R. and Turocy, T. L., *Gambit: The Package for Computation in Game Theory, Version 16.3.0*, Accessed: 2025-05-06, 2025.

[58] Schulman, J., Chen, X. and Abbeel, P., *Equivalence Between Policy Gradients and Soft Q-Learning*, 2018, arXiv: 1704.06440.

[59] Schulman, J., Levine, S., Moritz, P., Jordan, M. I. and Abbeel, P., *Trust Region Policy Optimization*, 2017, arXiv: 1502.05477.

[60] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., *Proximal Policy Optimization Algorithms*, 2017, arXiv: 1707.06347.

[61] Selten, R., Reexamination of the Perfectness Concept for Equilibrium Points in Extensive Games, *International Journal of Game Theory* 4.1 (1975), 25–55.

[62] Selten, R., Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit: Teil I: Bestimmung des dynamischen Preisgleichgewichts, *Zeitschrift für die gesamte Staatswissenschaft / Journal of Institutional and Theoretical Economics* 121.2 (1965), 301–324.

[63] Shapley, L. S., A Note on the Lemke-Howson Algorithm, *Mathematical Programming Study 1: Pivoting and Extensions,* (1974), 175–189.

[64] Sutton, R. S. and Barto, A. G., *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018.

[65] van den Elzen, A. H. and Talman, A. J. J., A Procedure for Finding Nash Equilibria in Bi-matrix Games, *Zeitschrift für Operations Research* 35.1 (1991), 27–43.

[66] von Stengel, B., *Finding Nash Equilibria of Two-Player Games*, 2021, arXiv: 2412.17638.

[67] von Stengel, B., *Game Theory Basics*, Cambridge, UK: Cambridge University Press, 2022.

[68] von Stengel, B., van den Elzen, A. and Talman, D., Computing Normal Form Perfect Equilibria for Extensive Two-Person Games, *Econometrica* 70.2 (2002), 693–715.

[69] Vujić, M., 'Geometry of the Sets of Nash Equilibria in Mixed Extensions of Finite Games', PhD thesis, Universität Mannheim, 2022.

[70] Waltman, L. and Kaymak, U., Q-learning Agents in a Cournot Oligopoly Model, *Journal of Economic Dynamics and Control* 32.10 (2008), 3275–3293.

[71] Watkins, C. J. and Dayan, P., Q-learning, *Machine Learning* 8.3 (1992), 279–292.

[72] Wellman, M. P., 'Methods for Empirical Game-Theoretic Analysis', *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, AAAI Press, 2006, 1552–1556.

[73] Wilson, R., Computing Equilibria of N-person Games, *SIAM Journal on Applied Mathematics* 21.1 (1971), 80–87.

[74] Winkels, H. M., 'An algorithm to Determine All Equilibrium Points of a Bimatrix Game', *Game Theory and Related Topics*, ed. by O. Moeschlin and D. Pallaschke, Amsterdam: North-Holland, 1979, 137–148.

[75] Zai, A. and Brown, B., *Deep Reinforcement Learning in Action*, New York, NY: Manning Publications, 2020.

[76] Zhu, A., 'Solving $2 \times 2 \times 2$ Games', Undergraduate Dissertation in Mathematics, London School of Economics, Candidate number 13905, 2021.