Investigating Reinforcement Learning and Evolutionary Computation for Games with Stochasticity and Incomplete Information

by

Xinhe Zhou

S.B. Computer Science and Engineering and Mathematics,
Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author .....
Department of Electrical Engineering and Computer Science
July 20, 2022

Certified by .....
Erik Hemberg
Research Scientist
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Investigating Reinforcement Learning and Evolutionary Computation for Games with Stochasticity and Incomplete Information

by

Xinhe Zhou

Submitted to the Department of Electrical Engineering and Computer Science
on July 20, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis presents an application of reinforcement learning and evolutionary computation for solving complex games with incomplete information and stochasticity. Although there has been significant recent progress on AI game players, traditional deep reinforcement learning methods have mainly shown success in games with simpler properties. In this thesis, we evaluate two deep reinforcement learning methods: policy gradient and evolutionary strategies for training the neural network behind the AI players for Ticket to Ride, a complex strategic board game. By comparing AI players' performance and policies with existing heuristics players, we show that the AI players learn well under both training algorithms. Furthermore, the results indicate that training the AI players under the complete information game environment has a positive influence on their performance under the incomplete information game environment as well.

Thesis Supervisor: Erik Hemberg
Title: Research Scientist

Acknowledgments

First, I would like to thank everyone in the ALFA group for giving me the opportunity to work on this project. I am especially grateful to my advisor Erik Hemberg for his support and guidance throughout the last year. He helped me learn everything I needed for this thesis, from evolutionary algorithms to how to turn an idea into a research question. During the difficult times, his encouragement and valuable feedback have always motivated me to keep trying.

Next, I would like to thank all of my fellow instructors and the students I have had in the last 5 years. Thank you for helping me become a better teacher and reminding me to stay curious, and I have learned so much from each and one of you.

I am also extremely thankful for all of the friends I have made at MIT. They have brought me lots of joy and supported me through my ups and downs. They made the last 5 years much more special and incredible than I could ever imagine. I especially thank my podmate for introducing me to Ticket to Ride, playing Ticket to Ride with me, and cheering me up through it all.

Finally, I would like to thank my parents for bringing me into the world.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Building autonomous agents that can navigate in complex interactive systems is an important challenge in reinforcement learning (RL)[18]. There are many real-world applications surrounding this problem, such as robotics[20] and autonomous driving[29]. Over the last 20 years, many studies have shown games are a powerful tool for tackling this challenge. Games can be used to model an interactive environment with feedback, and their complexity can be controlled by scaling up the size of the state and action spaces[37]. Therefore, games are a natural subject for RL research.

Since the rise of Deep Learning, more and more RL methods involve the use of neural networks, and they are referred to as Deep RL methods [1]. Deep RL has shown success in many games, such as Atari games[24], Go[31], and Mujoco[34]. Some of the techniques, including REINFORCE Policy Gradient[38] and Evolutionary Strategies[28], can even be generalized to master multiple games. However, these games often have simpler designs than other games: they are deterministic; the players are always under complete information, and their rules are relatively simple. In games with more complex designs, researchers tend to create new RL methods tailored for the game[21, 5, 40]. Therefore, we are interested in applying the RL methods that have worked well in simpler games, to a more complex game with incomplete information and stochasticity — namely, Ticket to Ride.

In this thesis, we build a neural network for the AI players and implement two

training algorithms: REINFORCE policy gradient [38] and Evolutionary Strategies[28]. We train and evaluate the players in a 1-player variant (T2R-Solitaire) and a 2-player variant (T2R-Dual) of Ticket To Ride. In both variants, we compare the performance and policies of the AI players with heuristic players. We also explore how incomplete information and increases in the complexity of the game setting can impact AI players' policies, leading to a discussion on the generalizability of the methods.

1.1 Research Questions

1. **How can we encode all of the information in a complex game state with incomplete information?**

   There are many different sources of information in a game state, and not all of them are accessible to every player. For example, the private hand of one player is only accessible to the owner, and none of the players have access to the private deck. We hope to define a set of features for each game state and create an encoding that converts the features into an input matrix for the neural network, with the ability to turn certain features into private information.

2. **What are the differences between players trained under different AI techniques?**

   We will be comparing 2 methods, REINFORCE policy gradient and Evolutionary Strategies, for updating the neural network used in the AI player, and within each method, we also have 2 different training modes: self-play and supervised. We are interested in comparing the advantages and disadvantages of both methods and understanding how different methods arrive at different policies.

3. **How does incomplete information in the game environment affect the performance of the players?**

   We are interested in answering the question from two different perspectives. First, we want to know whether the traditional deep RL methods are capable of

achieving good results when the players are trained in an incomplete information game environment. Next, we want to evaluate how well can the players who are trained under complete information can perform in an incomplete information setting and see if the policies are different from the policies they produced when they are trained under incomplete information.

1.2 Contributions

In summary, we make the following contributions

- We designed and implemented the game engine for two variants of Ticket To Ride: T2R-Solitaire and T2R-Dual. We also created an encoding scheme that converts each game state into a binary feature matrix and supports private features.
- We implemented the REINFORCE Policy Gradient (PG) algorithm and the Evolutionary Strategies (ES) algorithm to train the neural network behind the AI players. We evaluated both algorithms under self-play and supervised modes.
- We evaluated the performance of every AI and heuristic player in T2R-Solitaire and T2R-Dual. The results showed that the PG player can reach a performance close to the greedy heuristic, which is the highest-performing heuristic player.
- We analyzed the performance and policies under different game settings. Specifically, we introduced incomplete information and different objectives in the game environment and compare the players' performance and policies across all settings. The results show that letting the players learn in the complete information variant is conducive to their performance in a game with incomplete information. Moreover, the PG player generates a more generalizable policy that can accommodate changes in the game setting, while the ES player actively updates its policy based on the changes.

The rest of the thesis is organized as follows. Chapter 2 provides a review of some related literature on AI game players and past computational work for Ticket to Ride.

Chapter 3 presents the design of the original game, the process of redesigning two variants of the game, and a summary of the decision flow used in the game engine. Chapter 4 describes the architecture of the neural network used in the AI players and the training algorithms. Chapter 5 reports the experiments, results, and evaluations of the performance and policies of the players. Finally, chapter 6 offers some answers and remarks to our research questions and describes some future directions for this project.

Chapter 2

Related Work & Background

This chapter presents background information on Ticket To Ride and related literature on applications of RL for building AI game players. In particular, we provide a survey of algorithms used to learn games with different complex features, as shown in Table 2.1. We also discuss some recent progress on computational approaches to solve Ticket To Ride.

Ticket To Ride is a multiplayer board game with simple rules yet strategic game play. In this game, players take turns drawing train cards and using them to claim routes on a map to form a path between the desired destinations [26]. It is considered a more complex game because of its complicated state and action spaces, as indicated in Table 2.1. In each game state, there are multiple private and stochastic sources of information, and the availability of actions changes from state to state.

Algorithm	Game(s)	Number of Players	Stochasticity	Incomplete Information	Complexity
Deep Q-learning[22]	Atari Games	1	No	No	Simple
Alpha Go Policy Gradient[31]	Go	2	No	No	Intermediate
Evolutionary Strategies[28]	Atari & Mujoco Games	1	No	No	Simple
ReBeL[5]	Texas' Hold'em	2+	Yes	Yes	Intermediate
Suphx[21]	Riichi Mahjong	4	Yes	Yes	Advanced
Heuristic[10, 32, 6]	Ticket To Ride	2+	Yes	Yes	Advanced

Table 2.1: A high-level summary of the relevant algorithms and games covered in this chapter. For each other algorithm, the summary includes the game(s) it is used on, the number of players, whether it is stochastic, whether it is incomplete information, and the complexity of the game(s) based on the size of the state and action space.

2.1 Deep Reinforcement Learning Algorithms

Over the last decade, there has been significant progress in the field of deep RL, and the most prominent examples are its applications for building AI players for games [21, 5, 40, 31, 24]. In 2013, DeepMind published the first deep RL model to successfully learn from interactions in a high-dimensional game environment. The algorithm used Q-learning with a convolutional neural network (CNN) to approximate the rewards associated with different actions at a certain state, and it showed excellent performance on many single-player Atari games [24]. This approach is later known as Deep Q-learning [22]. Before then, CNNs were used more often in computer vision and graphics [19, 36], but it turned out that its ability to extract information from a local structure in a game state is also an effective tool for training game players. The map in Ticket To Ride can be seen as a graph with many structural properties such as connectivity and path overlaps[39], which makes CNN also a good option for building the architecture for the neural network.

Even though Deep Q-learning was promising in single-player games, this value-based approach had limitations that made it difficult for games with more complex reward systems[41]. There is another RL technique that directly optimizes the policy, and Alpha Go was one of the famous applications of this technique [31]. The key idea behind Alpha Go is the REINFORCE policy gradient algorithm, which uses a neural network to produce the probabilistic policy vector, and the player samples this vector to take the next action. After a sequence of actions is made, a gradient update is performed on the network using the rewards and policy vectors. This is also the algorithm we use to train our PG players for both T2R-solitaire and T2R-dual.

In 2017, a study by Salimans et al [28] found that compared to other traditional deep RL algorithms that rely on gradient updates, Evolutionary Strategies, an optimization tool that uses evolutionary algorithms [2], was a more scalable method to optimize the neural network in RL problems. Instead of doing a gradient update, the

algorithm adds Gaussian noises to copies of the parameters of the network and conducts rollouts with the new parameters, and the results of the rollouts will produce fitness scores that demonstrate how well the neural network with the new parameters would perform. Then the neural network is updated based on the fitness scores: the Gaussian noises with a higher fitness score are going to have a larger impact on updating the network. The study showed that in Atari and Mujoco Games[34], the performance of computer players trained under Evolutionary Strategies was comparable to the models trained under Deep Q-learning, and the training process was also more parallelizable. We are interested in applying this approach to see how its performance compares to policy gradient in T2R-Solitaire and T2R-Dual.

2.2 AI Players for Games with Stochaticity and Incomplete Information

Compared to the board games mentioned in the previous section, Ticket To Ride is a more complex game that involves stochasticity and incomplete information in its design. The AI players designed for similar games were usually a lot more complex and use multiple techniques tailored for the games. Two prominent examples were Facebook's AI Poker player ReBeL[5] and Microsoft's AI Mahjong player Suphx[21].

ReBeL was designed to solve Heads-up No-Limit Hold'em Poker, a multi-player poker game where the players go through rounds of betting based on their hands. The player with the highest-scoring hand wins the round at the end. The challenge comes from the fact that part of the other players' hands is private information [4]. To predict other players' hands, ReBeL used a notion called Public Belief State, which uses observations of public information to make probabilistic predictions of private information in each game state. This helped convert the game into a complete information game, and they were able to use equilibrium-finding algorithms such as CFR-D[42], to converge to a Nash Equilibrium among the players.

Suphx used a different way to convert the game into a complete information game

in the training process. Mahjong is a more complex game because it has multiple sources of incomplete information (from other players and the deck), and there are many stochastic elements in the game. The approach in ReBeL might not work well here because the dimensions of the game could grow exponentially if all private information elements were converted into probabilistic predictions. Instead, they used an approach called "oracle-telling" that exposes all players to a complete information version of the game (assuming all private information is public) at the beginning and gradually takes away the “oracle" by removing certain features from the input throughout the training process[21]. This technique, along with policy gradient algorithms, showed good performance (i.e. the player demonstrates a good understanding of the rules and played better than average human players). They also used added some look-ahead features and parametric search functions based on the rules of Mahjong to help the player achieve high performance among the top human players later. We are interested in the “oracle" approach here because it can be easily adapted to work with any existing RL algorithm, and we are incorporating this idea when we design the encoding scheme for T2R.

2.3 Previous Work on Ticket To Ride

Previously, researchers have investigated strategies for Ticket To Ride using different computational approaches. Chintha [9] measured the relative importance of different routes and cities by analyzing properties like connectivity and eccentricity (i.e. how close to the center is each city). They found that the cities closer to the center of the map are useful regardless of the starting hand, so one heuristic strategy would be to claim routes near the center at the beginning. They also analyzed the profitability of different destination cards by comparing their cost with their reward, and the results show that destination cards with longer shortest paths are more profitable.

Other researchers have designed and compared the performance of different heuristic algorithms such as graph search algorithms on Ticket To Ride[10, 32]. In [10], Scott Lee et al. tested 4 different heuristic algorithms that have different priorities

in playstyles. For example, the Destination Hungry Algorithm prioritizes destination cards, while Long Route Algorithm prioritizes longer routes. They found that the player that makes greedy decisions (the decision that would give the most immediate reward in one step) is the best player in 2-player Ticket To Ride, while the player that prioritizes longer routes is the best in 4-player Ticket To Ride. This brings up a good observation that certain heuristics might work well in one variant of the game but not all variants of the game. We also observe this trend when we introduce different features in the game environment.

However, not a lot of progress has been made on building AI players using RL due to the complexity of the game. Nguyen et al [11] proposed a method to encode the features of a game state and train an AI player to learn the rules of the game using curriculum learning. However, the model converged to suboptimal policies that did not demonstrate an understanding of the rules.

With Policy Gradient and Evolutionary Strategies algorithms, researchers created AI players that can achieve high performance in many games, but most of the games are not as complex as Ticket To Ride. The previous studies on deep RL applications for Ticket To Ride have also shown that the complexity of this game presents a great challenge for building AI players. In this project, we combine Policy Gradient and Evolutionary Strategies algorithms with other techniques like CNN and binary encoding, to build AI players that can learn in a complex game environment with stochasticity and incomplete information, under the setting of Ticket To Ride.

Chapter 3

Game Design

This chapter provides an overview of the design of T2R-Solitaire and T2R-Dual. We first describe the key components and rules of the original game Ticket To Ride: First Journey. Then we discuss the process of redesigning the game into T2R and the rationale behind the design choices. Finally, we present the setup of T2R-Dual and T2R-Solitaire and the decision flow in the game engine.

3.1 Ticket To Ride: First Journey Game Setup

Ticket To Ride: First Journey is a multiplayer board game released in 2016 as a spin-off of the classic board game series Ticket To Ride.[27] As shown in 3-1, the game board is a map with 19 cities connected by 64 routes of different colors, and the goal of the players is to claim routes on the map to form paths between cities on the given destination cards.

Figure 3-1: Map used in Ticket To Ride: First Journey

We use the following multi-graph in Figure 3-2 to represent a portion of the original map, which is also the simplified map that we will use for T2R later. In this graph, every city is represented as a vertex labeled with an integer, and every route is represented as a colored edge weighted with its length. Later on, the example game play demonstrations will be based on the following map.

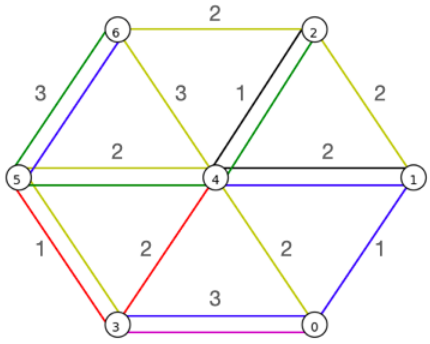

Figure 3-2: Map used in T2R-Solitaire and T2R-Dual. Each city is represented as a vertex labeled with an integer ID, and each route is represented as a colored edge with the weight labeled above.

3.1.1 Game Rules

At the beginning of the game, two decks of cards are shuffled and put face down. The first deck has 72 train cards, with 12 cards for each of the 6 colors: red, blue, white, green, black, and yellow. The second deck has 20 destination cards, each with a different pair of cities that the player needs to build a path between. Every player draws 5 train cards and 2 destination cards and is also given 20 trains.

After the game starts, players take turns performing actions on the game board. In each turn, the player needs to choose between one of the following 3 actions: draw train cards, change destination cards, or claim a route. If the player chooses to draw train cards, the player draws the top 2 cards from the deck. If the player chooses to change the destination cards, the player discards both destination cards on hand and draws 2 from the deck. Finally, if the player chooses to claim a route, the player needs to discard train cards that match the color of the route and use the same number of train cards and trains as the length of the route. For example, a yellow route of length 1 can be claimed using 1 yellow train card and 1 train, as illustrated in Figure 3-3.

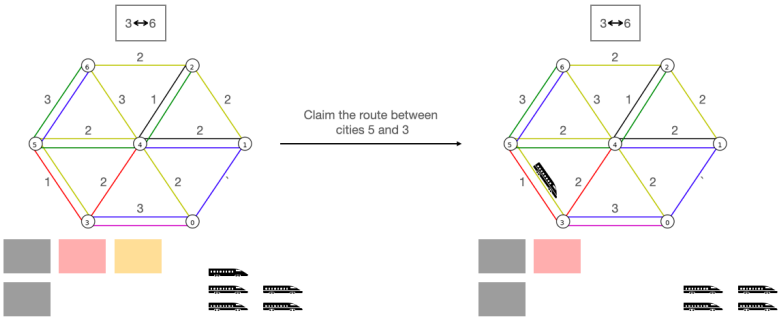

Figure 3-3: Example: claiming a route between cities 3 and 5 using 1 yellow train card and 1 train

To complete a destination card, the player needs to use the routes they have claimed to form a path between the cities, as illustrated in Figure 3-4 Upon completing, the player draws a new destination card from the deck. Once a player has completed 6 destination cards, the player wins the game.

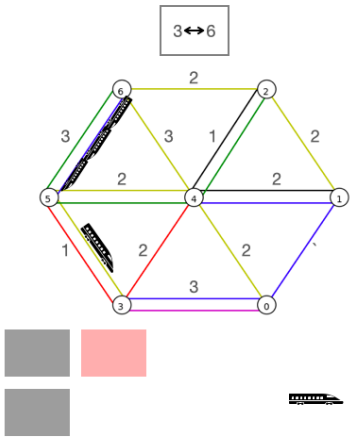

Figure 3-4: Example: completing a destination card between cities 3 and 6. There is a path between city (vertex) 3 and city 6 using edges $(3, 5)$ and $(5, 6)$.

There are some special destination cards called coast-to-coast cards. Each player can win this card by completing a path that connects a city on the west coast and a city on the east coast, and it counts as 1 destination card.

3.2 Redesigning Ticket To Ride to T2R

We redesigned the game to allow us to create benchmark experiments that focus on our research questions of interest. Specifically, we modified the game setting so that it maintains the essential elements and rules of the game and can be played as a single-player game as well. There are 2 key changes: we simplified the goal by making players only need to finish the destination cards they are given at the beginning of the game, and we also added a point-based reward system.

The deck of destination cards significantly increases the complexity of the game. This information is private, and it is also a source of stochasticity that changes the objective of each player multiple times during the game, so it's hard to make predictions on other players' behavior when the players can change destination cards[8]. Although the deck of train cards is also stochastic private information, we can have a better estimate of the probability distribution of the train cards using the discarded

cards. Therefore, we removed the deck of destination cards so that the players can focus on one objective throughout the game and only need to deal with one source of stochasticity.

In the original game, the result of the game is binary for each player: you either win or lose. It does not tell us how much progress each player has made. To remedy this, we added a point-based reward system. This system also draws inspiration from the point-based reward system from other Ticket To Ride variants[26].

Instead of having a binary score at the end, we let the players receive incremental rewards after every action. To compute the reward, we convert the map into a weighted graph, where the unclaimed routes are weighted by their lengths, and the claimed routes (which have been claimed by this player) have 0 weight. We define progress made by an action on a destination card to be the change in the shortest path distance between the cities on the card, and the total progress is the sum of the progress made on all destination cards on hand. We measure progress this way because having a smaller shortest path distance means it would take fewer trains to complete the destination card in the future, hence the player is closer to completing the destination card. We also added a decay factor $\gamma < 1$ for the reward to incentivize the players to complete the destination cards as soon as possible, and it has also been shown to help reduce the number of training steps[13].

We also made some minor changes such as using a smaller map, reducing the number of trains the player has, and removing the special coast-to-coast cards. These changes simplified the game without affecting the game play.

3.3 Game Setup and Decision Flow

Now we describe the game setup and decision flow of the redesigned game that we use to train and evaluate our players in the experiments. This game can be played with any number of players. We specifically focused on the single-player and two-player versions, namely, T2R-solitaire and T2R-dual.

Similar to Ticket To Ride: First Journey, we start by shuffling the 72 train cards

and puttting them face down. Every player draws 5 train cards and is given 10 trains and 2 destination cards. Then the players take turns performing actions until one player finishes their destination cards or no train card left in the deck. Since there is no destination card deck, the player can only choose between drawing 2 cards or claiming a route in each turn. After an action is performed, the player immediately receives a reward computed based on the system mentioned earlier.

Figure 3-5 illustrates the decision flow of each player in a game of T2R-Solitaire/T2R-Dual.

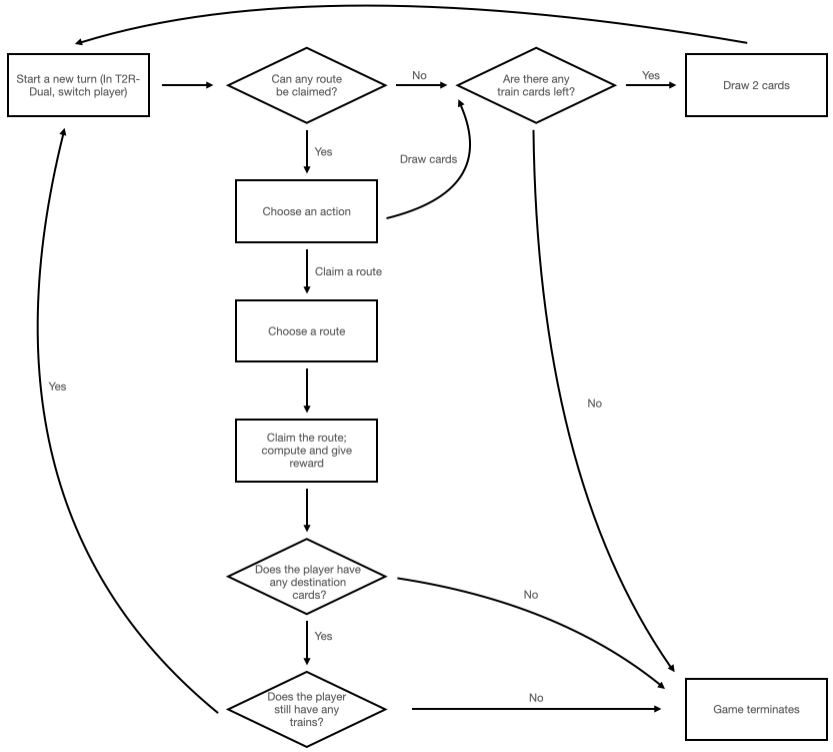

Figure 3-5: Decision flow in T2R-Solitaire and T2R-Dual. At the beginning of a turn, the current player starts with the top left program. Each rectangular entry represents a program where the player needs to choose an action, and each rhombus entry represents a decision that needs to be made by the game engine to make the right transitions for the player.

The main difference between T2R-Solitaire and T2R-Dual is that in T2R-Solitaire, the player only needs to navigate in a stochastic environment and learn to collect rewards as soon as possible, while T2R-Dual is incomplete-information from private train cards and destination cards, and potentially different objectives.

Chapter 4

Model Architecture and Algorithms

In this chapter, we describe the design of the neural network for the AI players and the encoding scheme used for the input feature matrix. We also present 2 algorithms used to train each AI player: the REINFORCE policy gradient algorithm for the PG player and the Evolutionary Strategies algorithm for the ES player.

4.1 Model Architecture

Symbol	Description	Value in T2R
V	Number of cities (vertices)	7
C	Number of colors	7
L	Length of Longest Route	3
K	Number of players	1 for T2R-solitaire, 2 for T2R-Dual
N_t	Number of initial trains per player	10
N_c	Number of train cards per color	12

Table 4.1: The list of symbols and notation used in this chapter

The remainder of this chapter will adopt the notation in Table 4.1.

We use a convolutional neural network as a decision-making “brain" of the AI agents. It takes in the feature matrix as the input and outputs a policy vector of size

$(1 + V \cdot V \cdot C) \times 1$. This policy vector represents a probability policy vector with a distribution of probabilities of different actions, where the 0^{th} value corresponds to the probability of drawing 2 cards, and each of the next $V \cdot V \cdot C$ values corresponds to the probability of claiming a specific route on the map. For the hidden layers, we use 1 Conv2D layer with channels of size $2 \times 2 \times 1$ and 1 fully connected layer before the output layer, as illustrated in Figure 4-1.

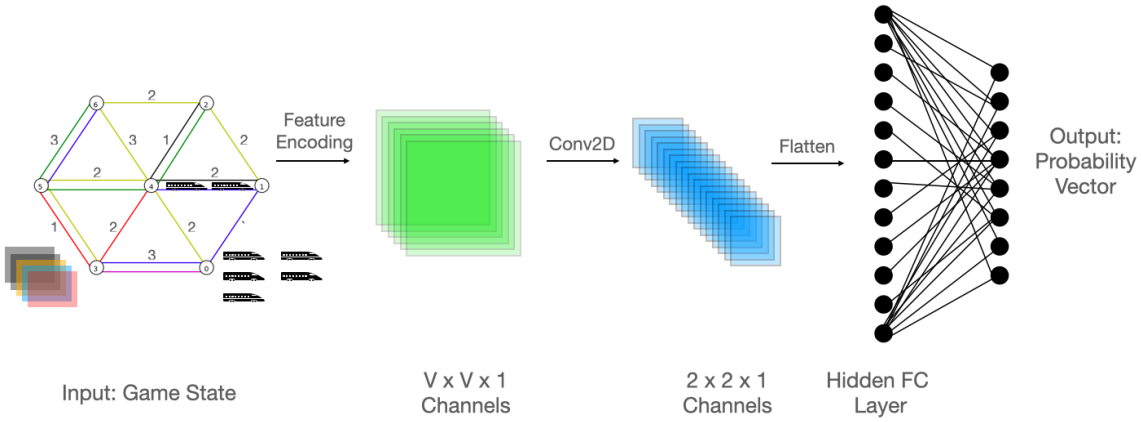

Figure 4-1: Architecture of the neural network behind the AI players. Each game state is first translated into a binary feature matrix consisting of channels of dimensions $V \times V \times 1$. Then it gets sent into a Conv2D CNN layer. The output has dimensions $2 \times 2 \times 1$ and gets flattened before the final linear dense layer. After this layer, the network outputs a probability vector that represents the policy.

4.1.1 Input Encoding

In order to help the neural network understand the information in a game state, we convert each game state into a binary feature matrix, with an encoding scheme similar to the one from [21]. Binary encoding also allows us to easily turn certain features from public to private: we simply replace the channels for private features with channels of all zeros.

As shown in Figure 4-2 feature matrix is concatenated with 4 groups of matrices: map structure matrices, occupancy matrices, private hand matrices, and train matrices. Each group is built using channels of size $V \times V \times 1$. Now we describe how information is encoded in different groups.

We represent map structure and occupancy using 2 multi-graphs, where the vertices are the cities and the edges are the routes. To encode the map structure, the weight on each edge is the length of the route, and if there is no route between two cities, this edge would have 0 weight. To encode occupancy, the weight on each edge is the ID of the player who claimed this route. Similarly, unclaimed routes would have 0 weight. The matrix representation of each multi-graph has dimensions $V \times V \times C$, which can be seen as a matrix of C channels

We then use binary encoding to convert both multi-graphs into categorical features: we create $W_{max}+1$ copies of each graph matrix, where W_{max} is the max weight of any edge in this graph. Therefore, we use $L+1$ matrices for map structure and $K+1$ matrices for occupancy. If the route (v, v, c) has a length of l, then the l^{th} copy has the value 1 at entry (v, v, c), while the other copies have value 0. Similarly, we use each occupancy matrix to encode the routes that have been claimed by each player. If the route (v, v, c) has been claimed by the k^{th} player, then the k^{th} claimed route matrix has value 1 at entry (v, v, c).

We encode information on private hand (the private train cards) and trains owned by each player as numerical features. Both private matrices and train matrices consist of K matrices (one matrix for each player). For each private matrix, we use N_c channels to encode the number of train cards in each color. If the player has x train cards of that color, we let the first x channels consist of all 1's, and the rest of the channels consist of all 0's. Similarly, we use N_t channels to encode the train cards owned by each player. If the player has x trains at the moment, the first x channels are all 1's, and the rest are all 0's.

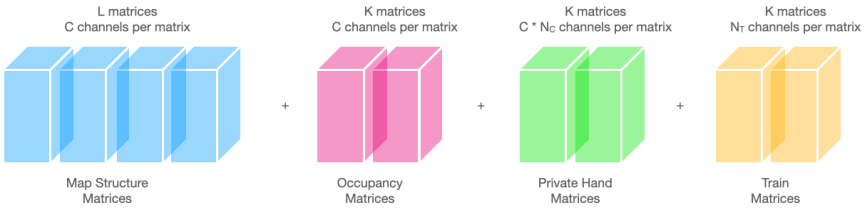

Figure 4-2: Input encoding for a given game state. The 4 groups of feature matrices are concatenated along the horizontal axis to ensure that the overall feature matrix is consisted of channels of size $V \times V \times 1$, and new features can be easily included in the future.

4.1.2 Masking

Before we sample from the policy vector, we need to mask the unavailable actions[16]. We create a binary matrix that represents the availability of the routes based on the occupancy, train cards, and the trains owned by the player. Then we do an element-wise multiplication with the policy vector to generate a new policy vector. If all actions are 0, then the player chooses to draw 2 cards; otherwise, we take an action sampled from the new policy vector.

4.2 Evolutionary Strategies Training Algorithm

Algorithm 1 Evolutionary Strategies Algorithm

function ES(θ_0^π, s_0, α, N, T, γ, ϵ)
  $\theta^\pi \leftarrow \theta_0^\pi$. ▷ Initialize the parameter vector of the policy network
  for $n \in \{1, \dots N\}$ **do**
    $\epsilon_n \sim \text{NORM}(0, \epsilon)$
    $\theta^{\pi_\mathbf{n}} \leftarrow \theta^\pi + \epsilon_n$ ▷ Add Gaussian noises to build similar parameter vectors
    $f_n \leftarrow 0$ ▷ Initialize the fitness score
    for $t \in \{1, \dots T\}$ **do** ▷ Evaluate each parameter vector with rollouts
      $G_t = \text{ROLLOUT}(s_0, \theta^{\pi_n})$
      $f_n \leftarrow f_n + \frac{G_t}{T}$ ▷ Increment the fitness score with expected returns
    end for
    $a_n \leftarrow \frac{f_n - \mathbb{E}[f]}{\sigma(f)}$ ▷ Compute the update factor based on fitness scores
    $\theta^{pi} \leftarrow \theta^{pi} + \frac{\alpha}{\epsilon N} \sum_{n=0}^{N} a_n \epsilon_n$ ▷ Update the parameter vector
  end for
  Return θ^π.
end function

We use an algorithm similar to the Evolutionary Strategies algorithm used in [28] to train the ES Player. We customize the fitness function to reflect the expected rewards the current model gets over many games. Note that ROLLOUT simulates a full game starting at initial state s_0 using the current policy θ^{π_n}) and outputs the total reward recorded in this game. A visualization of this algorithm can be found in Figure 4-3.

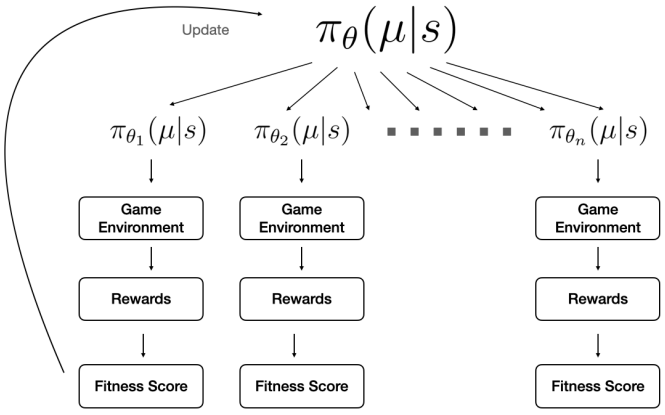

Figure 4-3: ES player training flow. In each round, n copies of the current policy are created, each with some Gaussian noise added. Then the policy is evaluated in the game environment, and based on the rewards, we compute the fitness score and update the current policy accordingly.

4.3 Reinforcement Learning Training Algorithm

Algorithm 2 REINFORCE Policy Gradient Algorithm

function REINFORCE(θ_0^π, s_0, α, T, γ)
  $\theta^\pi \leftarrow \theta_0^\pi$. ▷ Initialize the parameter vector of the policy network
  $i \leftarrow 0$
  while $i < T$ AND !ISTERMINAL(s_i) **do** ▷ Generate a game trajectory from s_0
    $a_i \sim \theta^\pi(s_i)$
    $s_{i+1}, r_i \leftarrow$ NEXTSTATE(s_i, a_i)
  end while
  for $t \in \{1, \dots T\}$ **do**
    $G_t = E[\sum_{t=0}^{T} \log \pi_\theta(a_t|s_t)(\sum_{t'=t}^{T} \gamma^{t'} r_{t'})]$ ▷ Compute the expected returns
    $\theta^\pi \rightarrow \theta + \alpha r_t \nabla_{\theta^\pi} \ln(\pi(a_t|s_t))$ ▷ Update the parameter vector
  end for
  Return θ^π.
end function

We use REINFORCE[38], a popular policy gradient algorithm known for its Monte-Carlo nature, to train the PG player. In this algorithm, we use !ISTERMINAL to determine if a state is a terminate state and NEXTSTATE to transition to the next state given the current state and action. A visualization of this algorithm can be found in Figure 4-4.

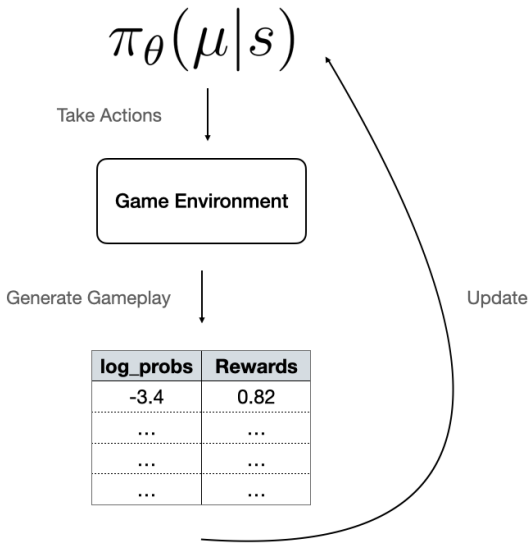

Figure 4-4: PG player training flow. In each round, the player generates a game play trajectory using the current policy, then a gradient update is performed on the current policy using the probabilities from the policy vectors used and the rewards in the trajectory.

Chapter 5

Experiments and Results

In this section, we describe the evaluation of AI players using the architecture and training algorithms presented in Chapter 4 in T2R-Solitaire and T2R-Dual. For T2R-Solitaire, the players are trained and evaluated in a simpler, complete-information game environment, hence the results allow us to develop some basic intuition for the relative competency and the policy of each player [33]. For T2R-Dual, we evaluate the players using 2 different training methods common for multi-player games: self-play and supervised. We also introduce incomplete information and different objectives in the experiments to understand how our AI players adapt their policies in a game with complex features. Table 5.1 provides an overview of the evaluation covered in this chapter.

Experiments	Training Method(s)	Incomplete Information	Different Objectives
T2R-Solitaire PG vs. ES	Selfplay	No	No
T2R-Dual Complete Info & Same Obj	Selfplay vs. Supervised	No	No
T2R-Dual Incomplete Info & Same Obj	Selfplay	Yes	No
T2R-Dual Incomplete Info & Diff Obj	Selfplay	Yes	Yes

Table 5.1: Evaluation overview. In each experiment, one of the 3 features (training methods, information access, and objectives) is being evaluated, while the other features are fixed.

5.1 Experiment Setup

We create test suites for both T2R-Solitaire and T2R-Dual to evaluate and compare the performances of all of the players. For each test suite, we run 1000 simulation games under the same game environment that the model is trained under: in the default map, the player starts with 5 trains per destination card given. To control the advantages/disadvantages introduced by the private deck during the evaluation stage, for each simulation, we only shuffle the cards once and let each player play under the same initial order of cards in the private deck.

We then record the rewards and policies generated by each model. We are interested in the distribution of rewards over 1000 simulations because both games are stochastic from the shuffled private deck at the beginning. We define a policy as the set of frequently claimed routes, which are the ones that were claimed in more than 10% of the games, by the same player. In the policy diagrams, the routes in the policy are shaded, and the darker the shade is, the more frequent the route is claimed. We also hope to draw connections between the objectives (the destination cards given to each player) and the policies generated by different models.

5.2 Heuristic Players

To evaluate the performance of the AI players, we need some baseline performance to compare the AI players to. Therefore, we create 3 heuristic-based players: Random, Frugal, and Greedy.

5.2.1 Random player

The random player claims a route whenever a route becomes available. When there are multiple routes available, this player picks one route out of all available routes with equal probability. When none of the routes are available, the random player draws 2 cards.

5.2.2 Frugal Player

This is a heuristic that a human player might use. The frugal player keeps drawing (and saving up, hence the name frugal) train cards until there is a path between a pair of destination cards that can be completed with the cards from the player's hands. When it's time to claim a route, the frugal player again randomly chooses a route out of the ones that are on the path. If another player "interrupts" the plan by claiming a route on the path, the frugal player needs to wait until there is another path available between 2 cities.

5.2.3 Greedy player

This player chooses the action associated with the most amount of immediate reward, which is the total progress made by this action. Since the map can be seen as a graph with positive weights, we use Dijkstra's algorithm[17] as a subroutine to run an All-Pair-Shortest-Path [30] algorithm to compute the progress for each destination card, and then we sum up the progress for all destination cards to measure the total progress made by this action.

5.3 T2R-Solitaire Evaluation

We want to use T2R-Solitaire to help us evaluate whether our AI players are capable of learning a reasonable policy to accomplish the given destination cards. In this test suite, every player is evaluated under the same destination cards: (1, 5), (2, 3). Note that the AI players are also trained under this objective.

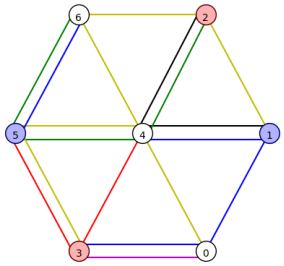

Figure 5-1: T2R-Solitaire destination cards. Every player needs to build a path between cities (vertices) 1 and 5 and another path between cities (vertices) 2 and 3.

Now we examine the overall reward distribution and the policies.

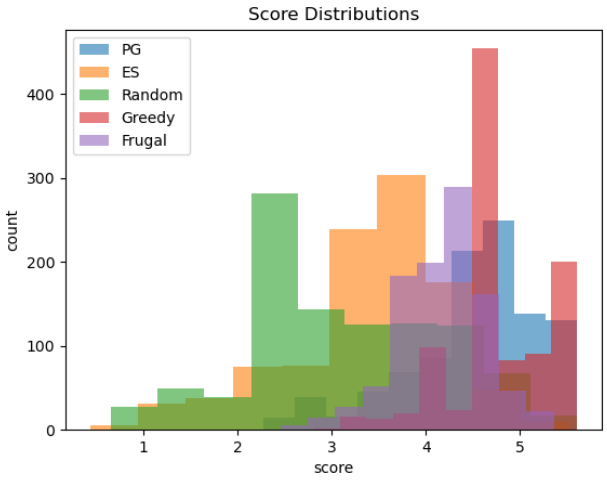

Figure 5-2: T2R-solitaire reward distribution. Since the game has a stochastic element – the private deck, the performance is measured by the distribution of scores (total rewards) over 1000 games. For each player, the final scores that it received over 100 games are then graphed as a histogram in the corresponding color.

From Figure 5-2, we can see that the rewards for the Random, Frugal, and ES players follow Gaussian distributions, which can be explained by the randomness factor in their decision-making process[14]. The distribution for the PG and Greedy models is more left-skewed, which indicates higher expected rewards and potentially more effective policies generated by these players.

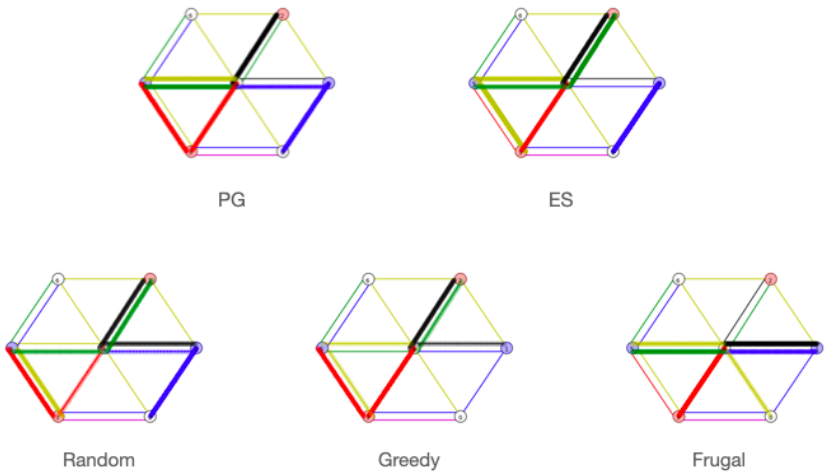

Figure 5-3: T2R-Solitaire Policy for every player. For each player, the policy is the set of frequently claimed routes, which are the ones that were claimed in more than 10% of the games, by this player. In each policy diagram, the routes in the policy are shaded, and the darker the shade is, the more frequent the route is claimed.

As shown in Figure 5-3, even though both PG and ES converge to policies that connect cities on the destination cards, the ES player has weaker performance. From the logs, we see that the ES player tends to wait longer to claim a route (i.e. spends more turns drawing cards), which makes the ES player suffer from the reward decay factor more than other players.

5.4 T2R-Dual Evaluation

5.4.1 Complete Information & Same Objective

We are interested in comparing the effectiveness of each model when it comes to learning to play T2R-Dual, a more complex game. In this test suite, we train both PG and ES players using 2 approaches: supervised and self-play. Under the supervised training mode, we let the player take turns with the Greedy player [12], and under the self-play mode, we let the same player take turns with itself to play as two players[15]. We use the same destination cards from Figure 5-1 for both players to compare the performance of different models under the same objective. For this test suite, we are

also assuming complete information for both players, which means both players have access to their opponents' hands.

After training each model, we let the AI models and heuristic models play against each other under the same initial game setting. For every pair of players, we run 1000 simulation games as described earlier. Then we combine all of the games that each player has played to graph the policies.

1st Player \ 2nd Player	SupervisedPG	SelfplayPG	SupervisedES	SelfplayES	Random	Greedy	Frugal	Total
SupervisedPG	517	483	745	886	805	290	690	4416
SelfplayPG	538	551	780	900	842	353	719	4683
SupervisedES	307	290	509	674	609	140	445	2974
SelfplayES	144	143	311	512	404	72	271	1857
Random	208	186	380	563	531	111	352	2331
Greedy	764	691	864	947	912	553	894	5625
Frugal	368	376	573	762	663	148	529	3419

Table 5.2: Number of games the 1st player wins in T2R-Dual with incomplete information & Same Objective. In the last column, the total number of games each player has won (as the 1st player) is presented to show the overall performance, and it is also color-coded to show which performance tier it falls in. Red corresponds to tier 1; green corresponds to tier 2, and blue corresponds to tier 3

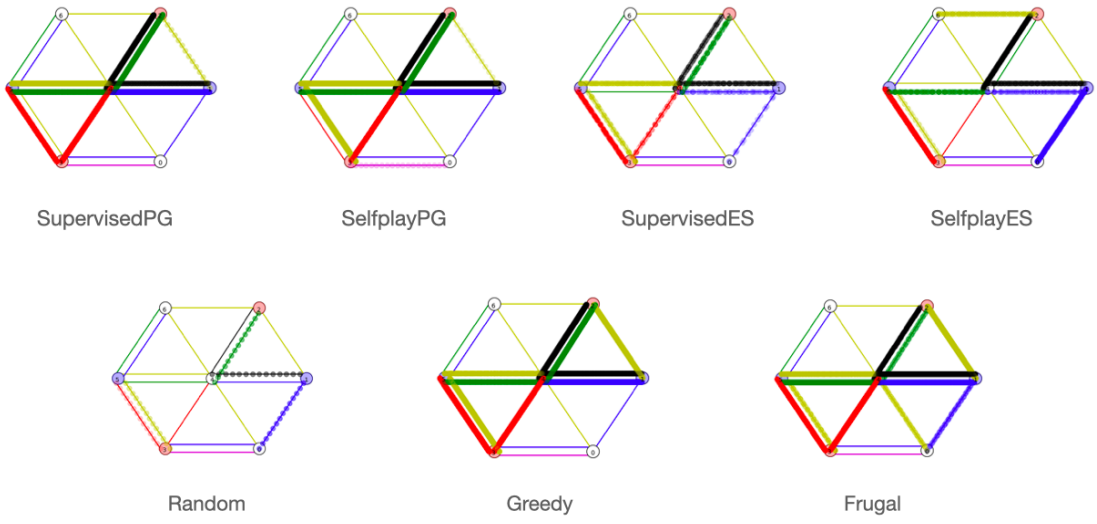

Figure 5-4: T2R-Dual complete info & same objective policies

As shown in Table 5.4.1, there are approximately 3 tiers of performance: the PG players and Greedy are in tier 1; SupervisedES and Frugal are in tier 2; SelfplayES and Random are in tier 3. It is also interesting to note that SupervisedPG and SelfplayPG have comparable performances, but SupervisedES performs significantly better than its self-play counterpart, which means the ES player benefits more from learning the rules through playing against an existing heuristic.

For the policies, we can see from Figure 5-4 that both PG players converge to policies similar to the greedy policy, which covers the shortest paths between the cities on the destination card. The ES players converge to policies that are quite different from each other, but both still cover some paths that connect the cities. From the fact that the routes are lighter in the ES policies, we can deduce that the ES players claim routes less often compare to other players except for Random, which again shows that the ES players tend to wait longer.

The policy for each individual set of simulation games that happens between every pair of players can be found in the appendix.

5.4.2 Incomplete Information & Same Objective

For this test suite, we are interested in investigating the impact of the availability of information during the training stage, when the models are evaluated under incomplete information during testing. While many studies have shown when the players have incomplete access to information, traditional learning algorithms might fail to help the player learn, recent results have shown that the complete-information variant of the game can still play an important role in getting the players familiar with the rules. We are comparing the performance (under an incomplete information game) of the AI players trained under incomplete information and models trained under complete information. We still use the game setup from Section 6.2.1. For the PG player, we train it on self-play mode, and for the ES player, we train it on supervised mode because from the last section, these two players perform better than their counterparts. Then we let the players play against each other like before.

1st Player \ 2nd Player	CompletePG	IncompletePG	CompleteES	IncompleteES	Random	Greedy	Frugal	Total
CompletePG	491	484	781	809	789	300	645	4299
IncompletePG	551	553	826	863	815	308	769	4685
CompleteES	219	190	470	565	489	83	388	2404
IncompleteES	244	202	482	522	452	111	292	2305
Random	233	212	492	570	486	102	330	2405
Greedy	712	718	928	936	909	573	889	5665
Frugal	386	312	717	753	708	143	550	2569

Table 5.3: Number of games the 1st player wins in T2R-dual with incomplete information & same objective

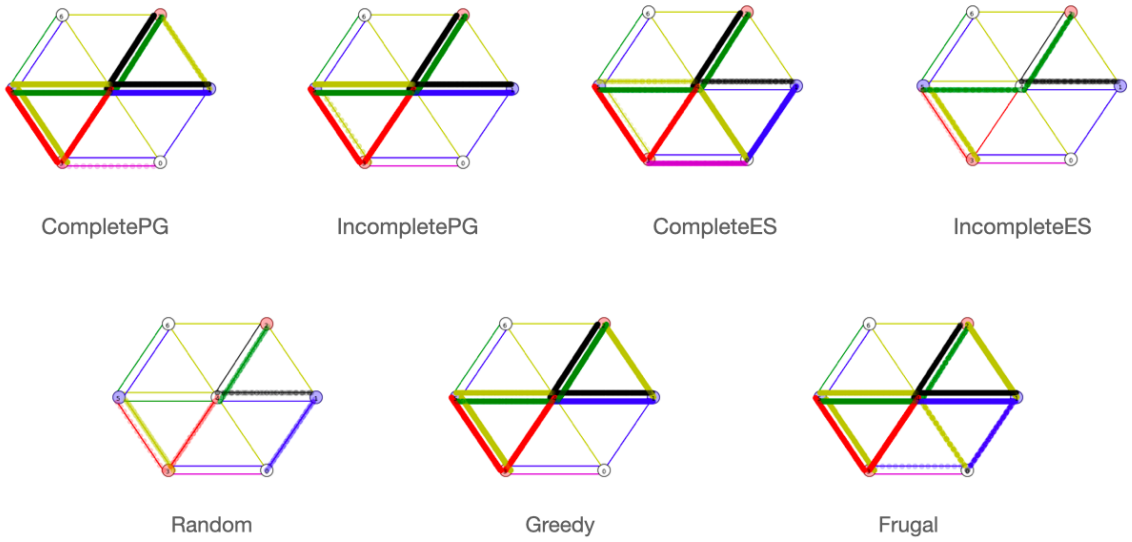

Figure 5-5: T2R-Dual incomplete info & same objective policies

Under this new setting, the Greedy player outperforms all other players. The PG players and the Frugal are in tier 2, while the ES players and Random are in tier 3, as shown in Table A.2.

Furthermore, we can see from Figure 5-5 that the gap between the performance of the greedy player and the AI player widens under incomplete information. In other words, AI players receive fewer rewards on average. This is expected because with less information available, the players are less likely to make good judgments and predictions. In addition, the difference between the performance of the same AI

player trained under complete and incomplete information is small, which supports our claim that letting the players learn under the complete information variant could also help the players gain an understanding of a game with incomplete incomplete information.

5.4.3 Incomplete Information & Different Objectives

When two player starts with different destination cards, the complexity of the game increases because now the player also needs to predict other players' objectives, and there is more private information. We are interested in how players learn to adapt under such an environment. We use the same models from and evaluate them by letting them play against the same heuristic player that starts with a different set of destination cards: (3, 6) and (0, 2), as illustrated in Figure 5-6. We evaluate how well our AI players' policies generalize in similar game settings through this test suite.

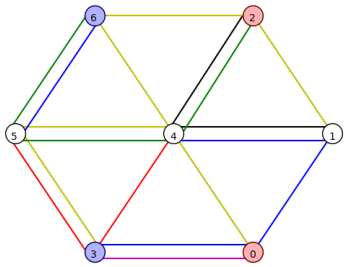

Figure 5-6: T2R-Dual opponent's destination cards

We use the PG and ES models trained under incomplete information (from the previous section) and let them play against all three heuristic players. Then we compare the policies generated by these players in the new game environment

2nd Player / 1st Player	Random	Greedy	Frugal	Total
PG	970	743	869	2582
ES	703	370	511	1584

Table 5.4: Number of games the 1st player wins in T2R-Dual with incomplete information & same objective

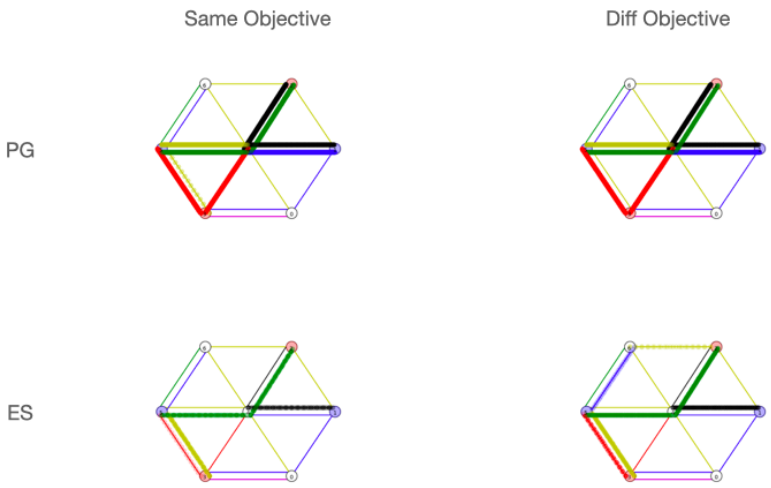

Figure 5-7: T2R-Dual incomplete info & different objectives policy adjustments

Both AI players adjusted their policies to adapt under the opponent's new game play, as illustrated in Figure 5-7. The PG player only made a minor adjustment to focus on the red route between (3, 5) and still achieved a better performance than the ES players. The ES player started claiming routes that were not in the original policy more often. One possible reason could be that these routes are key routes that could be used to build alternative paths when the shortest paths were taken by the opponent, so the ES player could be using this route to build an alternative path. This shows that the PG player's policy can accommodate minor changes in the environment while still maintaining good performance, while the ES player actively adjusts its policies when the game environment is changed.

w

Chapter 6

Conclusion & Future Work

The goal of this project is to evaluate the effectiveness of deep RL methods in complex games with incomplete information and stochasticity. We used Ticket to Ride as a prototype to build two games with these features, and we implemented 3 heuristic players and 2 AI players to learn the game using REINFORCE Policy Gradient and Evolution Strategies, respectively. In addition, we designed an encoding scheme to convert the features in a game state into binary matrices, which can then be interpreted by the neural network behind the AI players. This encoding scheme can work for a game board of any dimension, and we can extend it to include other categorical or integral features as well.

Through our experiments, we discover that the REINFORCE policy gradient algorithm can train our network to produce a stable policy that receives relatively high rewards, while the Evolution Strategies algorithm produces various different policies in different games. While the ES player receives a reasonable amount of reward, it is usually lower than the reward received by the PG Player in all game settings. In addition, the Evolutionary Strategies algorithm makes significant changes to its policies when the game setting is modified, while the REINFORCE algorithm still converges to the same policy.

Finally, we conclude that both of our AI players perform slightly better under complete information, compared to the incomplete information variant of the game, which means the incomplete information feature does present some challenges to the

players. However, the AI players trained under complete information play almost as well as the AI players trained under incomplete information, and the policies generated under complete information are also similar to the ones generated under incomplete information, which shows that the complete information variant of a game can help the AI players develop an understanding of game rules that is also applicable in the incomplete information variant.

6.1 Future Directions

In this project, we analyzed the relative performance and policies based on the observed actions taken by the AI players. Some analysis could be done in the future to understand how did each AI player arrive at the policies. For example, since the input matrix is binary, we can find some insights from the parameters of the network after training to rationalize the decisions made by the AI players (i.e. which features have a positive influence on which action).

As we mentioned earlier, some minor elements in the game are not part of the game environment at the moment, and the map we are using is a partial map of the original game, so a natural extension would be to implement all features of the game and train the players under the complete map. Another direction for future research would be to incorporate more rollout-based methods such as look-ahead features [35] and Monte-Carlo Tree Search[7]. These methods have been shown to improve the performance of AI players in other applications of reinforcement learning on board games. [23, 3] We also believe that our architecture and algorithms can be applied in other more complex variants of Ticket to Ride, and for those variants with more active communities, hence the training process can also make use of real gameplay data to improve the performance[25].

Appendix A

Appendix

A.1 Reward Trajectories

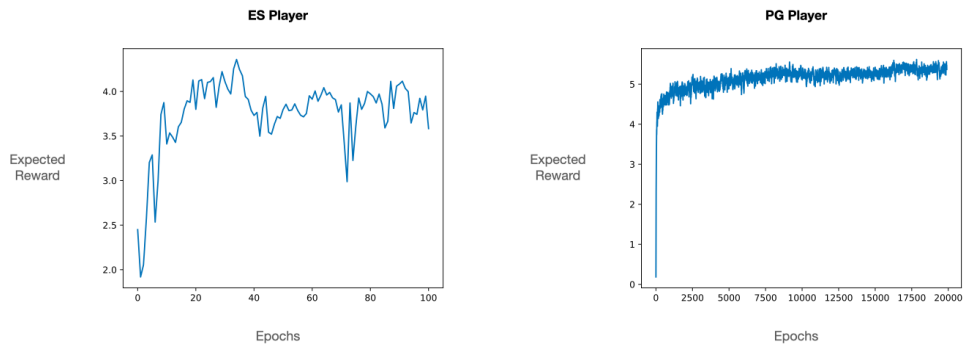

Figure A-1: Reward trajectories for ES and PG Players in T2R-Solitaire, during the training phase

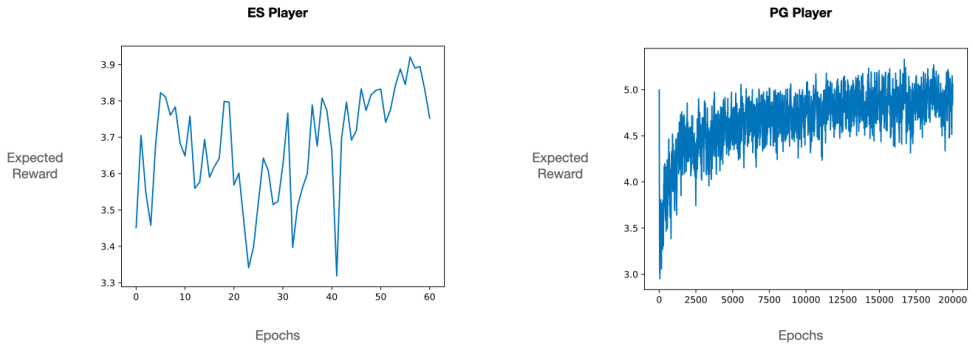

Figure A-2: Reward trajectories under supervised mode in T2R-Dual with complete information

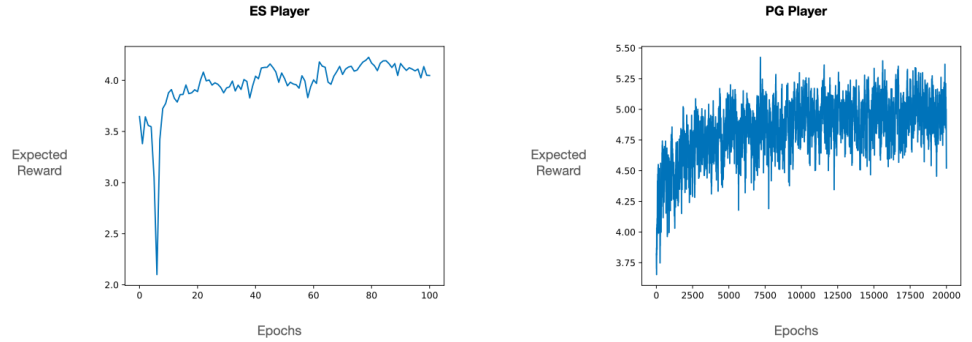

Figure A-3: Reward trajectories under selfplay mode in T2R-Dual with incomplete information

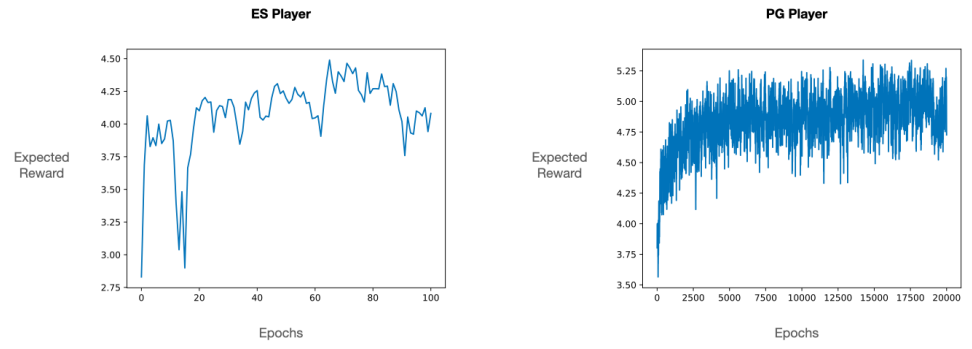

Figure A-4: Reward trajectories under supervised mode in T2R-Dual incomplete information

A.2 Reward Comparison Tables

2nd Player / 1st Player	SupervisedPG	SelfplayPG	SupervisedES	SelfplayES	Random	Greedy	Frugal
SupervisedPG	(4.99, 4.93)	(4.94, 4.94)	(5.19, 4.17)	(5.26, 3.18)	(5.18, 3.68)	(4.84, 5.41)	(5.18, 4.56)
SelfplayPG	(4.99, 4.89)	(5.04, 4.86)	(5.17, 4.12)	(5.24, 3.12)	(5.21, 3.58)	(4.86, 5.34)	(5.24, 4.45)
SupervisedES	(4.21, 5.07)	(4.23, 5.05)	(4.40, 4.34)	(4.55, 3.69)	(4.51, 4.01)	(3.99, 5.47)	(4.55, 4.75)
SelfplayES	(3.41, 5.26)	(3.29,5.24)	(3.74, 4.56)	(3.91, 3.88)	(3.78, 4.21)	(3.21, 5.55)	(3.80, 4.79)
Random	(3.74, 5.16)	(3.67, 5.16)	(4.03, 4.54)	(4.16, 3.84)	(4.14, 4.07)	(3.51, 5.49)	(4.15, 4.70)
Greedy	(5.51, 4.75)	(5.43, 4.74)	(5.56, 4.03)	(5.59, 3.04)	(5.56, 3.42)	(5.34, 5.14)	(5.61, 4.29)
Frugal	(4.67, 5.07)	(4.64, 5.08)	(4.82, 4.52)	(4.88, 3.76)	(4.73, 4.17)	(4.42, 5.57)	(4.87, 4.77)

Table A.1: Average rewards in T2R-Dual with complete information & same objective. In each entry, the average rewards over 1000 simulation games between the corresponding pair of players is displayed. The value in blue represents the average reward of the 1st player, and the value in red represents in average of the second player.

2nd Player / 1st Player	CompletePG	IncompletePG	CompleteES	IncompleteES	Random	Greedy	Frugal
CompletePG	(4.67, 4.77)	(4.74, 4.92)	(4.97, 3.79)	(5.03, 2.85)	(5.00, 3.68)	(4.59, 5.35)	(4.96, 4.53)
IncompletePG	(4.98, 4.68)	(5.03, 4.87)	(5.22, 3.75)	(5.31, 2.87)	(5.20, 3.67)	(4.89, 5.38)	(5.33, 4.51)
CompleteES	(3.86, 4.95)	(3.81, 5.16)	(4.18, 4.36)	(4.33, 3.31)	(4.04, 4.19)	(3.54, 5.50)	(4.34, 4.65)
IncompleteES	(2.79, 5.06)	(2.86, 5.28)	(3.26, 4.33)	(3.47, 3.56)	(3.17, 4.34)	(2.72, 5.56)	(3.34, 4.94)
Random	(3.73, 4.97)	(3.79, 5.17)	(4.17, 4.17)	(4.28, 3.36)	(4.08, 4.13)	(3.56, 5.49)	(4.19, 4.71)
Greedy	(5.41, 4.54)	(5.45, 4.78)	(5.58, 3.46)	(5.58, 2.78)	(5.53, 3.42)	(5.36, 5.17)	(5.61, 4.22)
Frugal	(4.59, 4.90)	(4.64, 5.20)	(4.77, 4.17)	(4.95, 3.30)	(4.81, 4.08)	(4.37, 5.54)	(4.88, 4.76)

Table A.2: Average Rewards in T2R-Dual with Incomplete Information & Same Objective

Bibliography

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[2] Torsten Asselmeyer, Werner Ebeling, and Helge Rosé. Evolutionary strategies of optimization. *Physical Review E*, 56(1):1171, 1997.

[3] Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In *IJCAI*, pages 672–677, 2007.

[4] Nolan Bard, John Hawkin, Jonathan Rubin, and Martin Zinkevich. The annual computer poker competition. *AI Magazine*, 34(2):112–112, 2013.

[5] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. *Advances in Neural Information Processing Systems*, 33:17057–17069, 2020.

[6] Wen-Chih Chang, Te-Hua Wang, and Yan-Da Chiu. Board game supporting learning prim's algorithm and dijkstra's algorithm. *International Journal of Multimedia Data Engineering and Management (IJMDEM)*, 1(4):16–30, 2010.

[7] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217, 2008.

[8] Bor-Sen Chen. *Stochastic Game Strategies and Their Applications*. CRC Press, 2019.

[9] Rakesh Chintha. Playing Ticket-to-Ride like a computer programmer. `https://towardsdatascience.com/playing-ticket-to-ride-like-a-computer-programmer-2129ac4909d9`. [accessed 12-July-2022].

[10] Fernando de Mesentier Silva, Scott Lee, Julian Togelius, and Andy Nealen. Ai-based playtesting of contemporary board games. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, pages 1–10, 2017.

[11] Daniel Dinjian and Cuong Nguyen. The difficulty of learning ticket to ride.

[12] Marina Dubova and Arseny Moskvichev. Effects of supervision, population size, and self-play on multi-agent reinforcement learning to communicate. In *ALIFE 2020: The 2020 Conference on Artificial Life*, pages 678–686. MIT Press, 2020.

[13] Vincent François-Lavet, Raphael Fonteneau, and Damien Ernst. How to discount deep reinforcement learning: Towards new dynamic strategies. *arXiv preprint arXiv:1512.02011*, 2015.

[14] Nathaniel R Goodman. Statistical analysis based on a certain multivariate complex gaussian distribution (an introduction). *The Annals of mathematical statistics*, 34(1):152–177, 1963.

[15] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

[16] Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*, 2020.

[17] Donald B Johnson. A note on dijkstra's shortest path algorithm. *Journal of the ACM (JACM)*, 20(3):385–388, 1973.

[18] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[19] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and Mohammed Bennamoun. A guide to convolutional neural networks for computer vision. *Synthesis lectures on computer vision*, 8(1):1–207, 2018.

[20] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[21] Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. Suphx: Mastering mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590*, 2020.

[22] Tambet Matiisen. Demystifying Deep Reinforcement Learning. `https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/`. [accessed 12-July-2022].

[23] Naoki Mizukami and Yoshimasa Tsuruoka. Building a computer mahjong player based on monte carlo simulation and opponent models. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 275–283. IEEE, 2015.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[26] Alan R. Moon. Ticket To Ride - a Board Game Series by Alan R. Moon. `https://www.daysofwonder.com/tickettoride/en/`. [accessed 12-July-2022].

[27] Alan R. Moon. Ticket To Ride First Journey- a Board Game Series by Alan R. Moon. `https://towardsdatascience.com/playing-ticket-to-ride-like-a-computhttps://www.daysofwonder.com/tickettoride/en/first-journey/`. [accessed 12-July-2022].

[28] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[29] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[30] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of computer and system sciences*, 51(3):400–403, 1995.

[31] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[32] Iain Smith and Calin Anton. Artificial intelligence approaches to build ticket to ride maps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 12844–12851, 2022.

[33] Vasilis Syrgkanis. Bayesian games and the smoothness framework. *arXiv preprint arXiv:1203.5155*, 2012.

[34] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[35] Yoshimasa Tsuruoka, Yusuke Miyao, et al. Learning with lookahead: Can history-based models rival globally optimized models? In *Proceedings of the fifteenth conference on computational natural language learning*, pages 238–246, 2011.

[36] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[37] Jörgen W Weibull. *Evolutionary game theory*. MIT press, 1997.

[38] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[39] R Teal Witter and Alex Lyford. Applications of graph theory and probability in the board game ticket to ride. In *International Conference on the Foundations of Digital Games*, pages 1–4, 2020.

[40] Chih-Kuan Yeh, Cheng-Yu Hsieh, and Hsuan-Tien Lin. Automatic bridge bidding using deep reinforcement learning. *IEEE Transactions on Games*, 10(4):365–377, 2018.

[41] Fei Zhang, Chaochen Gu, and Feng Yang. An improved algorithm of robot path planning in complex environment based on double dqn. In *Advances in Guidance, Navigation and Control*, pages 303–313. Springer, 2022.

[42] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. *Advances in neural information processing systems*, 20, 2007.