Partially Observable Monte Carlo Planning in Public Game Tree for complex imperfect information game King UP

Gao Peng, MRes
School of Computing and Communications
Lancaster University

A thesis submitted for the degree of
Master of Science by Research

August, 2022

Abstract

In the past, Game AI has been a challenging area where humans devote much effort to it. The game could be classified by whether players could fully observe the state of the game into two different game types: perfect information game and imperfect information game. While perfect information game has been studied well, like Go and chess, the research on imperfect-information games is limited to poker and dice. The board game King Up is a sort of imperfect information game with more complicated rules and attributes, such as partial observation, multiple players, partly shared goals, mixed strategy (cooperate and compete temporarily for goals), and a mixture of dynamic game and static game (vote and move). Until now, King UP has not been solved by existing methods.

Extended from Partially Observable Monte Carlo Planning, we propose a novel algorithm called POMCP in a Public Game Tree to solve the problem, in which we also combine the techniques from CFR with POMCP. The public game tree is a data structure used to store belief over partial observable information and could be applied in most imperfect information games with multiple players; inspired by CFR, we propose local search and decision estimation processes to compute the value function over correlated state and action; we also implement two decision estimation methods, deterministic policy by Upper Confidence Bound and stochastic policy by Regret Matching.

In the experiment, we evaluate the convergency situation of the algorithm. Also, we evaluate the algorithm's performance by further experiments like tournaments between different decision estimations and verify the generalization by Rock-Paper-Scissors.

At last, we rethink the motivation, insights, and contributions. Our algorithm could work with less prior knowledge and perform well in complex, imperfect-information games with multiple players like King UP and mixture game types (dynamic and static game).

Acknowledgements

I sincerely appreciate the support from Miss Lyuqian Bian, Dr.Leandor Soriano Marcolino, my family, and all my friends. It is a great honor to appreciate you all here.

Contents

List of Figures

List of Tables

Nomenclature

γ	Discounted parameter of reward
Ω	Finite set of possible observation distribution over state
σ	Policy
τ	Regret
A	Action
c	Public Game Tree node category
H	History of a series of action and observation
H_g	The state history of imperfect information game
I	Infoset
O	Observation
obs	private observation function
P	Player
Pr	Probability
Q	Value function
R	Reward value
S	State
T	Transition probability of state
UCB	Upper Confidence Bound value
$visit$	visit time

Z Terminal state

BoardHeight The height of the board

BoardWidth Maximum capacity of one level in the board

Bonus Additional bonus if someone get 0

PieceLocation Location of pieces on the board

PiecesNum The amount of individual pieces

PlayerNum Players involved in the game

RejectionLimit) Rejection cards amount

SetRound Set rounds

TargetCard Players' private targets

TopScore The score for the top piece

VoteCard Vote Cards in hands

Chapter 1

Introduction

1.1 Why game AI? Why King UP?

Since the start of human research about artificial intelligence, computer scientists in all the world, in all history, have considered game AI as a fundamental challenge on the route towards the romantic dream of strong AI. Building a powerful AI in the game is the best example to show agents' ability to learn, make decisions, and plan, which are significant signs of intelligence. Another reason scientists are infatuated with the game AI is that game is a virtual environment where humans can easily interfere and gain data with only a limited economic expense. Furthermore, compared with complicated reality, the game could provide problems defined clearly and worthy of conquering.

Except for the perfect information game that has been solved well by Alpha Zero [35], the progress of imperfect-information games AI is still limited. Most imperfect-information games AI focuses on limited tasks like poker or liar's dice to handle the difficulty of partial observation, but partial observable game AI has other challenges such as multiple players, teammates, and symmetric settings. These attributes could make games more complicated.

King UP is a board game that was published in 2004, and until now, no one has achieved any AI algorithm on it. Despite being a board game that could be played by children older than 8, it is a very complicated imperfect-information game. The Complicated Imperfect-Information Games (CIIG) like King Up should have attributes such as partial observation, multiple players (more than 2), a mixture of behaviors (cooperation and competence), and a mixture of game types (static game and dynamic game).

These settings could not just be used in games but also in robotics or other fields. For example, human-robot collaboration or multi-agent systems always need to solve the same challenges. Solving complicated imperfect-information games like King UP

could promote the development of these problems in reality.

1.2 Game AI techniques

Game AI techniques are the methods to achieve more intelligent performance, especially about smarter decisions in the game environment. Sometimes games would be classified by different rules in different backgrounds. Sometimes we mention games about things that amuse others; sometimes, it refers more to a activity that players try their best to beat each other. From different perspective, different methods are studied. The goal of our research is to achieve better performance in AI. In the paper, partial observation and imperfect information are equivalent.

There is a variety of techniques nowadays to achieve game AI; several main branches about game AI techniques will be introduced here with their advantages and disadvantages briefly.

One main branch of methods in game AI is forward statistical planning, including Monte Carlo Tree Search and Rolling Horizon Evolutionary Algorithm. These kinds of methods take advantage of computers' high-speed computation power to consider the future potential situations as much as possible to get an optimal policy; they could get not terrible action with limited computation resources and require no pre-training; usually, the more simulation they run, the more powerful strategies could be approached. Partially Observable Monte Carlo Planning is a POMDP solver extended from Monte Carlo Tree Search.

Another branch of algorithms is Counterfactual Regret Minimization, dominating traditional imperfect-information games like poker and dice. This series of algorithms are born from classical game theory. The game tree models all the history, including public and private information, and needs lots of training. When playing in the game, every time, CFR requires many resources to store and traverse the whole game tree. According to [7], Baby Tartanian8, an MCCFR program with pruning, used about 2 million core hours and 18 TB of RAM to compute its strategy. Even though CFR requires enormous amounts of computing resources, it could offer the most rational strategy if conditions permit because of the most completed game model.

The last branch of game AI technique is reinforcement learning (RL). RL is a typical learning method that requires many data to train, and the advantage of RL is the ability to learn through online interaction even though the learning process could be slow. On another side, the disadvantages are also apparent: RL lacks explainability, and the learning rate is not as stable as former algorithms.

1.3 Contributions

The state-of-the-art online planning algorithm, Partially Observable Monte Carlo Planning, is to solve Partially Observable Markov Decision Process (POMDP) from one single view, not for multiple players. The state-of-the-art algorithm in the imperfect-information games Counterfactual Regret Minimization series algorithms lose theoretical foundation beyond two persons and take a tremendous amount of computation resources. ML methods like Reinforcement Learning lack the transparency of decisions and require much data; therefore, existing algorithms cannot deal with complicated imperfect-information games like King Up without enough game data.

In our work, we extend POMCP to multiple players' versions to deal with complicated imperfect-information games like King Up. At last, we verify our algorithm Partially Observable Monte Carlo Planning in Public Game Tree(POMCP-PGT) could converge in both King Up for multiple players and Rock-Paper-Scissors; through transferring regret matching to POMCP-PGT to replace UCB, we turn our algorithms from the deterministic policy into stochastic policy. POMCP-PGT is also the first extension of POMCP that synthesize multiagent with independent views in one unified tree; it could offer a different way to solve multiagent decision problems with partial observation.

1.4 Organization of Work

The work is organized into six chapters: introduction part, background (and related work), methodology, result analysis, discussion, and conclusion. The Introduction part will introduce the meaning of our work briefly.

The second chapter is about the background of the research. First, we introduce state of the art about game AI including main methods, outstanding achievements, and challenges. Then we introduce the rule and settings of King UP; we will also give a brief analysis of the game. Later we would introduce several models to understand the problem and algorithms highly related to our work, including Monte Carlo Tree Search, Partially Observable Monte Carlo Planning, and Counterfactual Regret Minimization.

The third part will introduce the methodology in the following sequence: Firstly, the implementation of the game environment and required interfaces. Secondly, we introduce a novel planning algorithm — Partially Observable Monte Carlo Planning in public game tree, from representation to the main framework of the planning. Later the details about challenges and solutions to action selection in a public game tree will be mentioned. At last, we will discuss two decision estimation methods: Upper Confidence Bound and Regret Matching.

The fourth chapter is about the result of our algorithms and analysis. In the research, we implement a new statistical forward planning method POMCP-PGT and synthesize two kinds of decision estimation algorithms into POMCP-PGT. The algorithm's efficiency would be checked by the convergency situation and the performance of the specific algorithm against random choice. In addition, we do some tournament experiments to compare different algorithms' performance. At last, we test our algorithm on rock-paper-scissors to show its generalization.

The fifth chapter discusses our research, and we would rethink the motivation and value. Then we discuss its advantages and drawbacks, trying to analyze them further. Furthermore, extended research directions about this algorithm should be pointed out.

The last part of the research would be a conclusion. This chapter will review the whole dissertation and discuss the next steps.

Chapter 2

Background and Related Work

In this chapter, we offer the relevant background information necessary to comprehend our work. It is arranged as follows: the representation of the partial observable games, the gaming configuration of King UP, the approaches for the partial observable game (MAB, POMCP, and CFR), and related studies.

2.1 Representation of sequential decision under partial observations

2.1.1 Partially Observable Markov Decision Process

Markov Decision Process (MDP) is a standard sequential decision-making process where the current state and inputted action will influence the state's transition. Partially Observable Markov Decision Process(POMDP) is a model originated from Markov Decision Process. POMDP models a sequential decision process for an agent in a situation where it cannot gain the state of the process directly and is only able to obtain partial observation of the state. Therefore, POMDP models policy (or strategy) distribution over observation instead of state in MDP. It suggests that POMDP is a more general form of MDP. Given its generalization, it is applied to model sequential decision problems. Formally, POMDP is defined as a tuple: $\{S, A, T, R, \omega, O, \gamma\}$ [22]:
S means a finite set of states which represents the environment $S = s^1, ..., s^n$;

A is an finite set of actions that agents could take to affect states, $A = a^1, ..., a^n$;

T is the transition function defined as: $T : S \times A \times S \to \{0, 1\}$;

R is the reward value function defined as: $R : S \times A \times S \to \Re$($\Re$ is the possible reward set);

γ is the discounted parameter for reward.

ω is the finite set of possible observations, which means all the observations that agents could perceive from the environment,$\omega = \{o^1, ..o^n\}$;

O is the observation, in POMDP is a distribution over Ω,$O : S \times A \times \omega \to [0, 1]$.

At a specific point in POMDP, the environment is in state $s \in S$, and the agent is performing action $a \in A$; based on the transition function T, the environment transitions to state $s' \in S$. In the interim, the agent receives both an observation o and an instant reward r; o is derived from ω and O, while r equals $R(s, a)$.

The objective of the agent is to maximise the expected long-term reward: $E\left[\sum_{t=0}^{\infty} \gamma^t * r_t\right]$, where r_t equals to the reward the agent receives at some specific moment. It is easy to find that discounted parameter γ decides the degree that the agent care about the future. The smaller it is, the greater the importance an agent places on current reward. A policy or strategy is a probability distribution over observation and action $Pr(a|O)$.

2.1.2 Imperfect Information Extensive-form Game

Imperfect-information extensive-form game is another form to represent partially observable games. Imperfect-information extensive-form game is a model that illustrates the policy interaction in games involving numerous rounds and multiple players with private observation. This model derives from game theory; in this area, the common objective is to approximate a Nash-equilibrium where every player's strategy has been optimal. Apart from some definitions staying the same with POMDP, mentioned above, there are some novel notations as complementary. Imperfect information extensive-form game is portrayed using the following notions: $\{P, A, Z, R, H, I\}$:

P means the set of players;

A means the action sets like in POMDP;

Z is all the terminal nodes of the game where nobody could make any moves, and $z \in Z$ is one terminal state;

R is the reward function like in POMDP;

H is a set of sequential history of actions taken by players, $h \in H$ denotes a certain history; in Imperfect-information games, h could also be viewed as the state of the game; in addition, we use $h \rightharpoonup h'$ to represent a situation where h could leads to h' with a series of action;

I is the information set (also called observation in other models). In particular, for a certian player i get infoset $I_i \in I$, there could be several different $h \in H$ in the tree offering the same observation (infoset) owing to the partial observation; $h \in I$ means that a history h could offer the player a certain infoset I.

Except above notions, there are also some relative terms which are widely used in this area. **Perfect Recall** is an assumption that all the players are assumed to remmember all the information that they have access to since the beginning. The

problem that one single observation (infoset) I could not identify h is called **Non-Locality problem** because of private observation.A policy $\pi(I)$ is a probability vector over actions for player i with infoset I(player i is decided by I).

2.1.3 Multi-Armed Bandit

Multi-armed bandit (MAB) is the last type of model to describe decision problems with uncertainty and partial observation. The setting is straightforward: there are always n options with an unknown reward distribution, and the objective is to optimise the policy while simultaneously deducing the payoff of exploitation and exploration after rounds. This model is consistently applied to online learning problems and focuses primarily on the payoff. As a result of its simple setting, MAB is surrounded by a large number of solid theoretical works. However, the simplicity of the setting makes it difficult to generalise and apply to complex real-world problems.
Even though MAB is not an ideal model for sequential decision-making problems, its performance with theoretical assurance makes it a fundamental component of sequential decision-making algorithms.
Upper Confidence Bound (UCB) is one of the most well-known algorithms for choosing action under uncertainty in MAB settings. It is invented to deal with the explore-exploit tradeoff in uncertain decision-making. UCB provides a way to balance the explore-exploit tradeoff. The idea of UCB is based on probability: Consider a given evaluation over one choice for a decision-maker and how much confidence we could trust in the evaluation. The UCB equation is divided into two parts: average reward and optimal bias. One of the most famous versions of UCB is Upper Confidence Bound 1 [3]:

$$UCB(t) = \begin{cases} \infty, & \text{visit=0} \\ UCB(t-1) + \sqrt{\frac{2*\log N}{visit}}, & else \end{cases}$$

2.1.4 Comparison of the representations

Compared with POMDP, imperfect-information extensive-form games concentrate more on using a game tree to depict the game state precisely and considers more from the set of problems in game theory. Therefore, it ignores some problems that are accounted for in POMDP. Firstly, POMDP uses the transition function to describe that the transition of states is influenced by the environment, action, and state, but imperfect information game considers action and state; secondly, POMDP considers immediate reward after single action, and imperfect information game just computes payoff at the end of one game.

2.2 King Up

King UP is a complicated imperfect information game with attributes like multiple players, partial observation, and a mixture of game types. In the game King UP as shown in Figure2.1: There are several distinguished pieces and a board with several levels; each player is given a private target card with chosen targets of pieces and limited vote cards. The game would pass through 3 stages: 1. players, in turn, set a piece on some level of the board until the end of this stage; 2. recursively, players promote the piece to a higher level until someone reaches the top; 3. players vote to decide whether to calculate the results when all choose yes, or go back to stage 2. Players' target is to earn as many as points according to their target card.

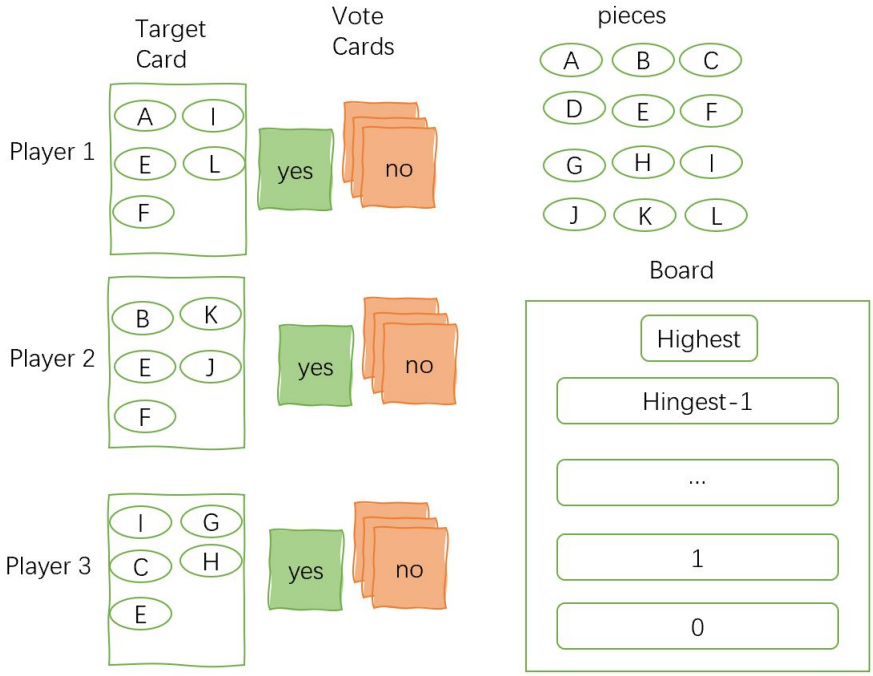

Figure 2.1: King UP

Parameters in the King UP environment to represent the real-time state are defined as below:
$TargetCard$ A array of players' private target;
$< target_1, target_2..target_n >$ where any $target_i$ denotes a set of several independent pieces for player i;
$VoteCard$ A array of players' cards for rejection because each player only has limited chances to refuse;

$PieceLocation$ Piece location function to record every piece's location in the board; $PieceLocation(piece)$ denotes the location for certain $piece$;
Parameters to initialize a game are listed as follows: $RejectionLimit$ Rejection cards amount for a player at the beginning;
$BoardHeight$ The highest level of the board;
$BoardWidth$ The limited number of pieces could share the same level;
$PlayerNum$ Players involved in the game, ranging from 3 to 6;
$PiecesNum$ is the sum of individual pieces;
$SetRound$ Before playing, each player has $SetRound$ chances to set a piece on the board;
$Bonus$ The bonus is awarded to a player who gets 0 credits as an additional bonus;
$TopScore$ The score is awarded to those whose pieces are on the top level.
In later sections, we will use these notations to define the experiment environment. After introducing the game, we discuss the playing rules (Figure 2.2 shows the game

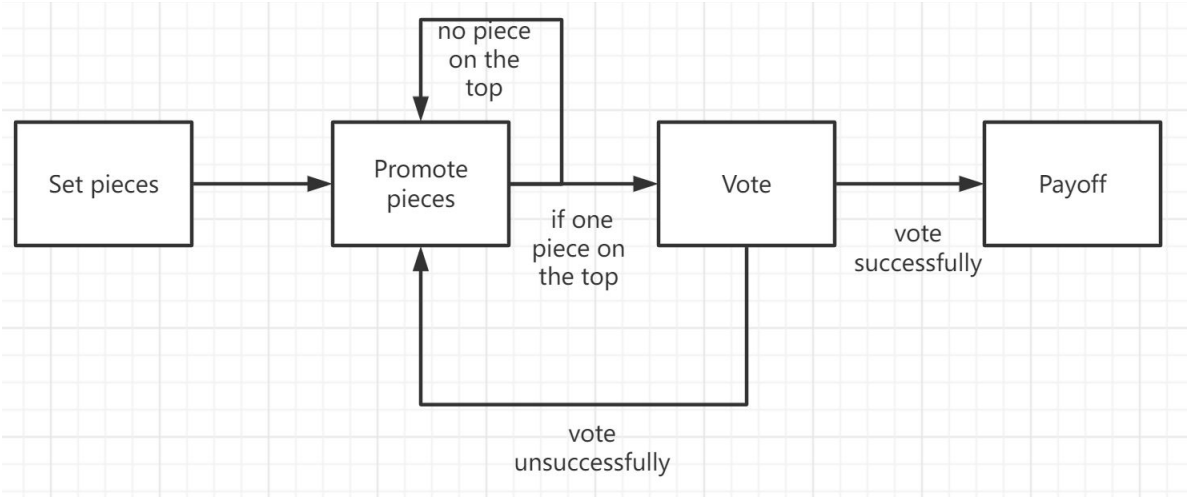

Figure 2.2: King UP-Game Process

process): 1. At the beginning of the game, each player has randomly received a card $TargetCard$ that contains the names of several pieces; 2. Players, in turn, choose one piece from the pieces stack to put in a level on the board, noticing that players could only put it in a level no higher than $BoardHeight - 2$ and, except for the 0th level, each level could contain limited pieces $BoardWidth$; After rounds, there would be pieces left out of the board. 3. Players recursively move one piece up from floor i to floor $i + 1$. Continue this stage till one piece is on the top floor; 4. Players independently choose 'yes' or 'no' ('no' would consume $VoteCard$) to decide whether to end the game. Actions would be public observable after the vote. If at least one

'no' is played, the piece at the top would be discarded, then go back to stage 3; 5. If all approved the top piece or there is no other action to take, we calculate each player's credit. The player's credit sum is related to pieces in their card, and each piece staying on the board would add credit according to its position. There is a supplement rule: if some players get no piece on the board, a considerable amount of bonus points *Bonus* would be given (Special Rule). Three rounds of games would be played, and each round begins with the lost player in the last round.

2.3 Partially Observable Monte Carlo Planning

Partially Observable Monte Carlo Planning (POMCP) is the most important extension of Monte Carlo Tree Search for imperfect information game. Joel Veness and David Silver[26] put forward this algorithm to solve to solve large Partially Observable Markov Decision Processes; in POMCP, they put forward Partial Observable Upper Confidence Tree with Monte Carlo Planning which is extended from Monte Carlo Tree Search in Markov Decision Process; they also proved the convergence of POMCP in POMDP based on the theoretical work by Szepesvári [21] that UCT could converge in MDP.

In the POMCP, the tree is described by two sorts of nodes: observation node and action node. In the POMCP tree (shown in Figure 2.3.1), every node holds a set of belief particles to simulate the state distribution for certain observations represented by the node and also stores related value functions. At each observation node, the agent chooses the appropriate action, but the same action could lead to different observations representing the transition function in POMDP. To deal with the uncertainty about unobservable information, POMCP uses the belief sampling technique to choose the right belief state at the tree's root to simulate and help compute the payoff. By using Monte Carlo Simulation to repeat the process of belief sampling and game simulation from the root, the algorithm could master the game approximately and converge to a quite optimal policy for the root node. As an online planning method, the algorithm could start a search from any new observation node every time.

2.4 Counterfactual Regret Minimizatoin

Counterfactual Regret Minimization (CFR) was invented by Zinkevich[32]. Since its invention, CFR has played a significant role in imperfect information game like poker and dice. It has played a fundamental role in the state of the art of poker AI Libratus[[8]] and DeepStack [[25]], both widely believed to be milestones in AI history. Regret Matching, one of the essential element algorithms in CFR, and some

Algorithm 1 Partially Observable Monte Carlo Planning

procedure SEARCH(h, World Model)
  repeat
    if h=empty **then**
      $s \sim h$
    else
      $s \sim B(h)$
    SIMULATE(s,h,0)
  until time use up **return** $\arg\max_b Q(hb)$

procedure ROLLOUT(h,s,depth)
  if $\gamma^{depth} < \epsilon$ **then return** 0
  $a \sim \pi_{rollout}(h, \bullet)$
  $(s', o, r) \sim \mathcal{G}(s, a)$ **return** $r + \gamma \cdot rollout(s', hao, depth + 1)$

procedure SIMULATE(state, h, WorldModel)
  if $\gamma^{depth} < \epsilon$ **then return** 0
  if $h \notin T$ **then**
    for each $a \in A$ **do**
      $T(h, a) \leftarrow (N_{init}(ha), Q_{init}(ha), \emptyset$
    return Rollout(S,h,depth)
  $a \leftarrow \arg\max_b Q(hb) + c\sqrt{\frac{\log N(h)}{N(hb)}}$
  $(s', o, r) \sim \mathcal{G}(s, a)$
  $R \leftarrow r + \gamma \cdot rollout(s', hao, depth + 1)$
  $B(h) \leftarrow B(h) \cup s$
  $N(h) \leftarrow N(h) + 1$
  $N(ha) \leftarrow N(ha) + 1$
  $Q(ha) \leftarrow Q(ha) + \frac{R - Q(ha)}{N(ha)}$ **return** R

ideas to deal with Non-Locality problems (introduced in 2.1.2) in CFR are applied in our work; so in this part, we will introduce CFR's implementation.

The algorithm of Counterfactual Regret Minimization is based on the imperfect information extensive-form game that we mentioned in the previous section, not POMDP. All the notions from above are applied here, and the algorithms are implemented on a game tree according to the setting.

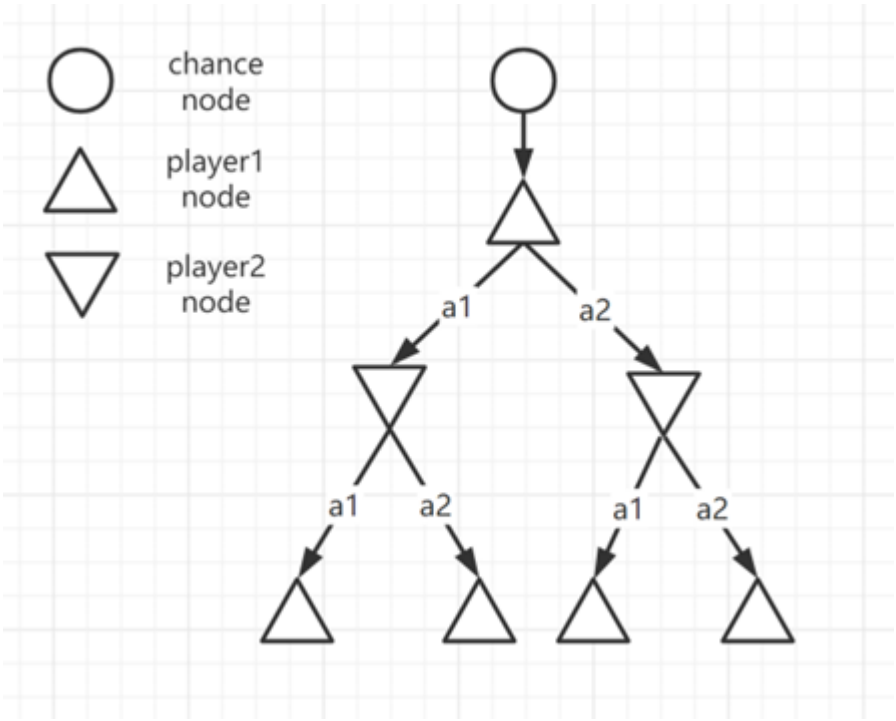

Figure 2.3: CFR

In CFR's game tree (shown in Figure2.3), the chance node means the action taken by the environment, like rolling the dice or shuffling the cards, and the player's node means that the player takes action at this node; a certain game state is a node in the tree, and its information is made up of the history of the action from the root node; of course, for a certain player in some node, some edges to the root could be invisible for it (partial observation).

Here are some notions about CFR, most of them are extending from imperfect-information extensive-form games: 1. History node h represents a action sequence that denotes all the information that a certain player knows or does not know; 2. Information set I_i denotes all the information player i could observe, including both public and private information for player i; 3. Policy set $\sigma \leftarrow I$ is a set of strategy distribution over every single action for independent decision-makers; 4. Q is also

the expected value;in addition, $Q(I)$ means the expectation at infoset I also called the average expectation;$Q(I, a)$ means the expected value when you choose a at I. 5. Regret τ is regret defined as the difference between average expected value $Q(I)$ and $Q(I, a)$.

No regret learning originates from game theory and plays an important role in CFR. The core idea behind no regret learning is that through the learning process, the action election always approximates the regret decline direction. No regret learning method is a series of algorithms that choose the policy to minimize the regret; in other words, you always tend to choose the policy with lower regret; this process could be stochastic (by choosing according to regret distribution after normalization) or deterministic (by choosing the one with min regret). It can also be understood in a way that uses average utility $Q_i(I)$ as a baseline, immediate regret is defined as $\tau_i(I, a) = Q_i(I, a) - Q_i(I)$, and the player tends to choose action with less regret. The policy $\sigma_i(s, a)$ in training is updated by the following equation:

$$\sigma_i(s, a) = \frac{\tau_i(s, a)}{\sum_{j=1}^{n} \tau_i(s, a_j)}$$

(If $\tau_i(s, a) < 0$ then $\tau_i(s, a) = 0$, sometimes a slight possibility is given)

This equation is called Regret Matching, a very useful no-regret learning algorithm. CFR uses no regret learning like Regret Matching in each training iteration to update policy.

The key to computing policy as no regret learning is to get right $Q(I)$. Non-Locality is a problem in CFR that there are possibly several nodes h that share the same I leading to a result that a player cannot localize itself in the tree precisely. To deal with the problem, CFR takes the following method to compute the right utility Q: once the player faces a particular situation I, traverse the whole game tree to identify all the nodes where $h \in I$, then it could get the correct utility over action $a \in A$ by the formula:

$$Q(a|I) = \frac{\sum_{h \in I} Pr(h|I) * Q(a|h)}{Pr(I) * Q(I)}$$

According to [32], if CFR takes no regret learning to choose action in each round of training, the average policy of all the rounds could converge to a Nash equilibrium in a 2-person's imperfect information game. Therefore, the last step of CFR is to compute the average strategy $\bar{\sigma}(I)$ distribution after training as the final result to play:

$$\bar{\sigma}_i^T(I) = \frac{\sum_{t=1}^{T} (Pr_i^t(I)\sigma_i^t)}{\sum_{t=1}^{T} Pr_i^t(I)}$$

2.5 Related Works

Game AI has always been an active area where lots of groundbreaking development for AI happens regularly. As DeepMind significant works [27, 28, 29] outperforming humans in Go and several related perfect information games by MCTS with Reinforcement Learning, the conclusion that given enough resources, RL with search could handle all the perfect information games is accepted by the community; the interests about perfect information game is moving to other problems. For partial observable games, there are several limited subjects attracting attention. There is a branch of traditional imperfect information games, which usually are represented in the model from game theory; these games include poker [6, 7, 8, 19, 20], dice [14, 15, 16], etc. have been researched on for as much time as Go, and AI for those imperfect information game with strict setting and comparatively simple rules has also outperformed human being by algorithms based on game theory. Following limitations of these algorithms based on game theory like Counterfactual Regret Minimization [8, 22, 23, 25, 32] and Fictitious Self Play [18, 19, 20, 29, 31] are obvious: 1. Their theoretical foundation is only applied to 2 players; 2. They are designed to compute Nash Equilibrium which could be useless when more than two irrational players are involved. Only a limited number of imperfect information games with complex rules are researched, like Hanabi [4, 13] and Settlers of Catan [12].

Because of the complexity of partial observable games, there is no general but powerful algorithm now. King Up has very challenging settings, including complicated behaviors, multiple players, and confidential information; this kind of game has never been solved before. Existing techniques, including RL, CFR, and MCTS, are not applicable in King Up. Therefore, we combine POMCP with CFR developing a more powerful algorithm to handle the problem.

Since POMCP was invented in 2010, it has become the state-of-the-art planning method for POMDP problems in different environments. [5] introduces parallelizing POMCP to speed up in larger and more complicated scenarios.[17] extend POMCP to a continuous real-time environment to realize robots' find-and-follow tasks. [2] introduce POMCP into multiagent settings where all the agents share the unified partial observation; following works about POMCP in a multiagent setting, they all follow the same settings like [11, 30] where they might share the joint history, joint observation, and joint action.

Most of the extensions from POMCP focus on the application area and do not violate the strict setting of POMDP for one single decision-maker. For two players game, POMCP could be changed in a way like a minimax tree does. Nevertheless, for the situation with more than two players, the solutions are rare. The only extension for more than two players imperfect information game is that [12] apply POMCP with human preference in Settler of Catan; in this work, they use a feature function

$\omega(B, S)$ to represent the feature from belief B and state S in substitution of observation node and use human preferences to generate behaviors from other agents; despite Settler of Catan is multiagent setting, POMCP in Settlers of Catan[12] admits their algorithm is designed for one single observer.[1] introduce another POMCP based on the theory of mind. In their method, POMCP for different players communicates and interacts during the playing time. The algorithm is very complicated, provides limited assurance about performance (the analysis of how the planning process is influenced by distributed setting is lacking), and leads to difficulty in giving feedback from a unifying view. Nowadays, even though POMCP is proved a powerful POMDP solver, no one has really extended it to multiple players' views.

The research about CFR is very active. Here are some key developments in the history of CFR: [24] introduces Monte Carlo CFR, where Monte Carlo Sampling is used to do pruning in the tree; [9] introduces deep CFR, which uses deep learning to symbolize the behavior tabularly and proved it outperform Neural Fictitious self-play[20] in two-person imperfect information game; [10] introduced the ReBeL algorithm in which deep reinforcement learning is combined with CFR search. The CFR has the theoretical assurance to approach Nash Equilibrium in 2 players' game, and, compared with all other algorithms in partial observation, it has the most detailed opponent model leading to a strong performance. However, the CFR model leads to unacceptable resource assuming when traversing the tree, even for small tasks in life.

Chapter 3

Methodology

3.1 Partially Observable Monte Carlo Planning in Public Game Tree

As introduced in former chapters, decision-making problems in partial observation are usually modeled according to POMDP. When planning method in POMDP like Partially Observable Monte Carlo Planning deal with multiple players' situation, independent agents will play at the same time with a different process which could make the analysis of strength difficult; methods like counterfactual regret minimization takes a lot of time when traversing all the game tree and could only deal with either dynamic or static extensive game.

In this part, we introduce a novel algorithm, Partially Observable Monte Carlo Planning in Public Game Tree (POMCP-PGT), which is an extension of Partially Observable Monte Carlo Planning (POMCP). Vanilla POMCP, proposed by D. Silver and J.Veness[26] executes planning on a tree consisting of action nodes and observation nodes so that it could represent a strict Partially Observable Markov Decision Process (POMDP) for a single player's view. For a game with multiple players, the other player's action could only be part of the observation and could lead to difficulties when sampling without knowing others' strategies.

POMCP-PGT is an extension of POMCP to do planning on a type of game tree that we call Public Game Tree. POMCP-PGT is one novel forward planning method for multiple players in one unified game tree and could perform well on extensive imperfect information game with dynamic parts and static parts simultaneously.

3.1.1 Representation of POMCP-PGT

POMCP-PGT inheritances notations from both POMDP and extensive imperfect information game to introduce the algorithm itself. While the notations were

mentioned in the former chapter 2, the related notations in our algorithm will be introduced in detail as follow:

H: Like in POMDP, H is also a series of action and observation,$h =< o_0, a_1, o_1 \ldots, a_n, o_n >$. Specifically, in our Public Game Tree, all the h could only contain history which is observed to the public; therefore,h could get a abbreviated form, $h =< a1, .., an >$.

S: As defined in POMDP, S is the set of state. In original defination, $s \in S$ contains all the information to identify a specific state.

I : Information set is the information that players could observe; $I_i(s)$ is what a certain player i could observe at a particular state s;

obs private observation function. $obs_i(s)$ denotes what player i observes privately at state s;

B : What we call belief B here is a probability distribution over state s given public information history $h : B(s, h) = Pr(s|h)$

P: Players set

σ: Strategy is a distribution over actions from some certain I_i which denotes as $\sigma_i(a, I_i)$

Q : Same expected value function in POMDP. It is a function related to Infoset I_i. Our planning goal is to optimize the expected value given observation $Exp(Q_i(I_i))$ for the decision maker i

Z : terminal state set, where no move could be taken.

To help understanding, several important notions would be explained with King UP setting: infoset I_i could contain $TargetCard$,$VoteCard$,$PieceLocation$ and public action history $h \in H$; state s should be the union of all players' $TargetCard$,$VoteCard$,$PieceLocation$ and public action history $h \in H$; $obs_i(s)$ would be $TargetCard$ for player i.

Public Game Tree we use for planning is namely a game tree, but the tree node is only identified by the public information node $h \in H$, which is different from the game tree identified by H_g in CFR(it contains chance nodes). As we set, node h could contain a series of information, including belief distribution over state $B(s, h)$, value function $Q(s, h)$ and node type c. $B(s, h)$ is the possibility of the state s given particular h; $Q_i(s, h)$ denotes the expected value for player i given state s and history h; node type c would be either static(vote stage in King UP), dynamic(make a move or set piece in King UP) or terminal(the end of the game).

The mixture of static and dynamic game makes POMCP-PGT different from other game trees. In an extensive form game mixed with both dynamic and static game, the subtree in a PGT made up of consecutive static nodes could create a temporarily invisible subtree. In this case, players could only decide according to the view of the last visible ancestor node. After this stage of the static game finishes, the subtree becomes publicly observable to all.

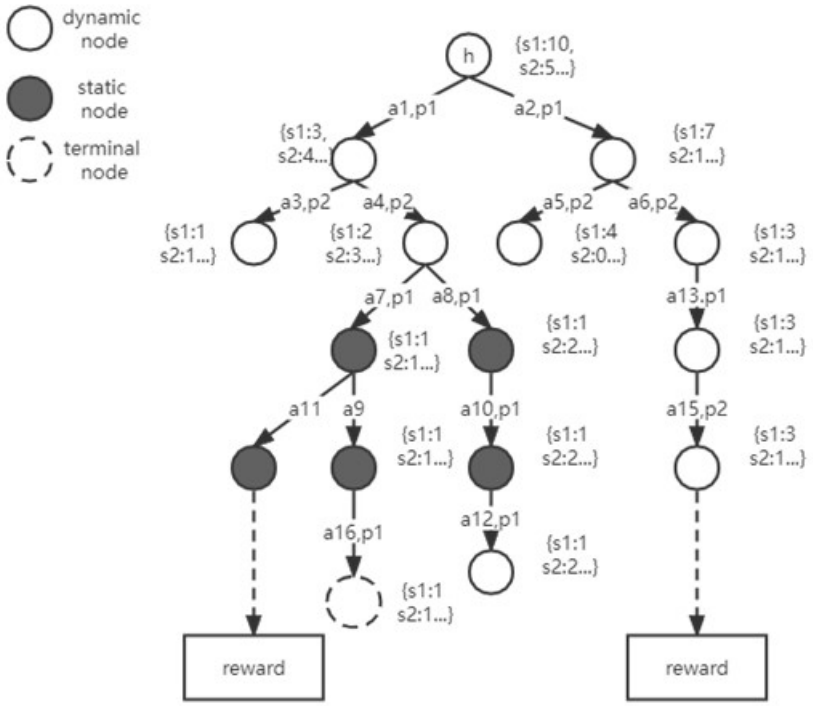

Figure 3.1: Public Game Tree

Figure3.1 is an example of PGT. As shown in it, only three types of nodes are represented by a circle in the image: dynamic node, static node, and terminal node; terminal nodes provide the payoff of one game; we always execute planning from one dynamic node or static node; the root node could be either of nodes which containing history information; every edge in the tree contains the taken action information; each node also contains the belief over state $s \in S$ as well as the related value function $Q_i(s)$ for all players; players take action along the tree.

The representation difference between my work and POMCP is that they modeled game tree in a POMDP from one single player's, others' action is contained in observation node; therefore, in POMCP for simulation part, you could only use Monte Carlo Simulation or run the algorithm again; but in our algorithm, each node at the same time offers positions for right players, the simulation also takes advantage of belief from before rounds. In the program, to save memory, only store $Q_i(h, s)$ and belief distribution Belief in each node is recorded. It could not be hard to deduct other information like $Q_i(s)$ and $Q_i(s, a)$ from existing information.

3.1.2 POMCP in Public Game Tree

In this part, the main procedure of POMCP-PGT will be introduced, especially the framework, and several component algorithms will be introduced in the latter part.

Similar to POMCP, the POMCP-PGT comprises three main essential procedures: Search, Simulate, and Rollout. Algorithm2 shows the specific pseudo code of POMDP-PGT.

The algorithm starts with the procedure Search. Public information history h is given and begins its search from the root node identified by h. The algorithm requires an independent virtual game to play the role of a world model so that we could know the control information (who should be following and available action list) and the final payoff situation at the terminal node; a little bit different here from POMCP is that our world model does not contain others' strategy. In King UP, the action list would be decided by inner parameters $TargetCard$,$VoteCard$ and $PieceLocation$ according to the rules in Chapter 2.2.

In the main Search procedure: before planning time use up, the agent sample state s(involving its private information and others') from both world model and past belief in root node h; the sampled hidden state would be the only actual state until the calculation of payoff and be kept in world model sandbox; in each node, corresponding decision-maker could only observe part of it; then it could come into Simulate decision process from the node; update belief and value function at h($Q_i(h,s) = Q_i(h,s) + \frac{reward_i}{visit(s|h)}, i \in 1..n$); recursively execute the procedure until the time use up; at last, return chosen optimal action.

In simulate part: the agent makes decisions at points on behalf of corresponding players with limited observation until a leaf node of the tree; if the node has never been explored, then we should expand child nodes linked with independent actions and use the rollout procedure to get a coarse evaluation over this node(it could give us an initial value expectation to optimize); if not leaf node or terminal node, we should use ChoosePolicy procedure(choosing optimal action would be discussed later and it is the core technique of our algorithm) to generate action a and instant reward r; then take action of a in world model to get new node h' simulate from h' and get $reward$; update $belief$ and Q and return $reward$.

In the rollout part: Doing Monte Carlo Simulation from given h, count the average payoff as the primary evaluation $Q(h)$. Here could play rollout simulation more times to get a more precise evaluation at the very beginning.

Figure3.2 contains one complete round of POMCP-PGT, the pink nodes show the h where decisions are made, and the red edges show the action is executed. It shows a process consisting of **simulation** from node h to h': h, (a2,p1), (a5,p2), **expansion**(h') and **rollout**(h').

Algorithm 2 POMCP-PGT

procedure SEARCH(h,I_i, World Model)
  repeat
    $s \leftarrow\ Sample(World\ Model, I_i)\ or\ Beliefs(h)$
    $reward \leftarrow SIMULATE(s, h, WorldModel)$
    update(h,s,reward)
  until time use up
return ChoosePolicy(h, exp=false)

procedure SIMULATE(s, h, WorldModel)
  if node h is leaf **then**
    visit(s,h)+=1
    $Q(s,h) = Q(s,h) + \frac{payoff(s,h)}{visit(s,h)}$
    return payoff(s,h)
  if node h is new node **then**
    Expansion from h
    return ROLLOUT(h,s)
  a $\leftarrow$ ChoosePolicy(h , $obs_i(s)$, player))
  $s', h', r \leftarrow WorldModel(player, a, s)$
  $B(h) = B(h) \cup s$
  $reward = r + discount * Simulate(s', h', WorldModel)$
  $visit(s,h) += 1$
  $Q(s,h) = Q(s,h) + \frac{reward}{visit(s,h)}$
  return reward

procedure ROLLOUT(h,s)
  random choose action until payoff
  return payoff

procedure CHOOSEPOLICY(h,$obs_i(s)$,exp=False)
  $Q(I), I \leftarrow LocalSearch(h, obs_i(s))$
  $a \leftarrow DecisionEstimation(Q, h, obs_i(s), I, exp)$

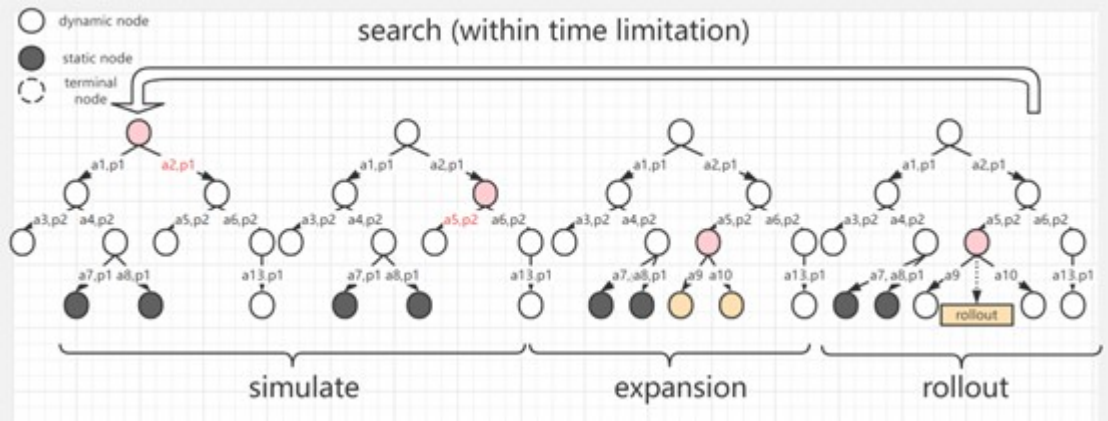

Figure 3.2: POCMP-PGT process

3.2 Choose Policy

The Choose Policy is a core part of algorithm2 POMCP-PGT; as described in the former part, the procedure's input would be node h and personal observation $obs_i(s)$ which denotes private observation that player i could obverve from the certain state $s \in S$. Our algorithm is value-based, so our design goal would be to get the value function $Q(a|\ I(h,\ obs_i(s))$ for all possible a is available at $obs_i(s)$.
As shown in algorithm POMCP-PGT, the procedure of **Choose Policy** comprises two parts: **Local Search** and **Decision Estimation**.

3.3 Local Search–Decision in Dynamic Subgames and Static Subgames

Expanding POMCP to multiple players' versions in a PGT will lead to a problem called Non-Locality. Non-Locality is a notion from Counterfactual Regret Minimization in section 2.4 which appears again in our algorithm POMCP-PGT. Non-Locality describes the situation that at one node in the game tree with a particular history h for player i, if there exists any hidden information in the history h, the player i should not decide according to $Q(h)$ but according to personal knowledge $Q(I_i)$. In the CFR's game tree, whether public and private information is depicted; therefore, CFR traverses all the tree to find all possible node h' with the same infoset I_i.
Our algorithm POMCP-PGT depresses hidden information in one node to reduce the size of the tree. However, each node h contains public information; the problem, like the King UP, synthesized both dynamic games and static games in one game tree;

in the static games part, players' actions should be temporarily invisible, which also leads to a non-Locality problem in POMCP-PGT.

Figure3.3 helps understand this non-Locality problem: in the right tree, when the

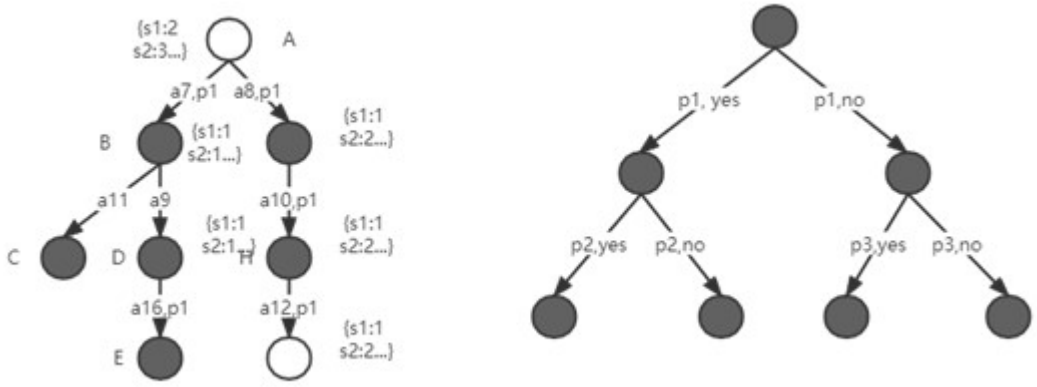

Figure 3.3: Non-Locality Problem

player makes decisions as p_2, it gets only two potential options: 'yes' or 'no' ; because the static sub-game stage is not over; so you should not observe others' actions. If you only compute the belief and expectation value at your node, you could not compute the correct expectation value $Q(h\ yes(p2))$ instead of $Q(h\ yes(p1)\ yes(p2))$ or $Q(h\ no(p1)\ yes(p2))$. The image also shows two trees with different root node types, and the Local Search method works for both.

Our algorithm uses a similar trick, **LocalSearch**, to fix this problem in POMCP-PGT (shown in Algorithm LocalSearch). According to the description of the tree representation in the previous chapter, each node contains the belief distribution and corresponding Q value. Intuitively, we understand that for a static decision node, we need to observe from the nearest node with public information, which could contain all the infoset I needed for a decision. In Algorithm LocalSearch, we use the function

Algorithm 3 LocalSearch

procedure LocalSearch(h,$obs_i(s)$,player i)
  h' SearhAncestor(h)
  $I = h + obs_i(s)$
  for $h'' \in subtree(h')$ **do**
    if $I(h') == I(h'')$ **then**
      $Q_i(I,a) = Q_i(I,a) + Q(obs_i(s), h'', a) * Pr(\frac{visit(h'')}{visit(h')})$

SearchAncestor to get the nearest ancestor node h' from current node h; In this

function, the program checks the parent node recursively from the current node until the root node or first dynamic node. Getting the nearest ancestor node h' (we could consider the set of all the child nodes linked to h' only through static node as a sub-game tree, and it could be this sub-game tree's root) is not efficient, and it could only offer you the infoset I consisting of public observation h' and private observation obs that player i should observe at node h; according to our design, this subtree would share the same public information h'; owing to the action information is defined in the edge, and the player at h' may not be player i, so for $Q_i(I,a) \neq Q_i(h',a)$. Our target is to calculate $Q_i\,(I,a)$; it is easy to come to the equation to calculate Q value:

$$Q_i\,(I,a) = \sum_{h\prime\in h',\ I(h')==I} Q\,(s_i,h'',a) * Pr(\frac{visit(h'')}{visit(h')})$$

Define node h' as a new root and traverse nodes in its subtree where its route to the root are all static nodes, then get all the h'' that share the same information I.

3.4 Decision Estimation

The motivation of this part is to deal with the explore-exploit dilemma in decision making. This part will introduce two decision estimation methods I implement inside our algorithms: Upper Confidence Bound and Regret matching.

3.4.1 Upper Confidence Bound

In our algorithm, we implement UCB1[3], which is a widely used version of UCB. In our case, we calculate the UCB value of $Q(a|\ I)$ and choose the action a with the biggest UCB value; N is the visiting times of infoset I and visit counts how many time a is taken from I; after UCB estimation, we would choose the action a with biggiest UCB value.

$$UCB_Q(I,a) = \begin{cases} \infty, & visit = 0 \\ Q(I,a) + \sqrt{\frac{2*\log visit(I)}{visit(I,a)}}, & else \end{cases}$$

3.4.2 Regret Matching

Another algorithm implemented in this work as a substitution of UCB to select action is Regret Matching (RM and the detailed definition are in Section 2.4). Regret Matching is also known as no-regret learning, a powerful algorithm in an

imperfect information game. The motivation I apply this algorithm to replace UCB is that while the forward planning method usually uses Monte Carlo simulation to sample randomly, they always choose action with the max reward by algorithms like UCB. Policy generated by this would lead to a problem that if the opponent could approximate the game tree, the decisions given by the algorithm would be deterministic and transparent to the opponent. Stochastic algorithms like RM could make the policy more elusive.

In our case, King Up is played among multiple players. So, we just use vanilla regret matching as a decision estimation method. In our algorithm's description, the regret τ is defined as:

$$\tau_i(a, I) = Q_i(a, I) - Q_i(I)$$

The policy $\sigma_i(I, a)$ is updated by the following equation:

$$\sigma_i(I, a) = \frac{\tau_i(I, a)}{\sum_{a' \in Action(I)} \tau_i(I, a')}$$

(If $\tau_i(I, a) < 0$ then $R_i(I, a) = 0.01$, a small number is given)

In addition, if all the action at the beginning is lower than baseline Q(I), our algorithm would give all the action with negative regret a very slight possibility to be chosen (0.01 we give in the experiments). In later experiments, it seems to influence the performance, so it is not as good as UCB. This detail will be researched further in later work.

Chapter 4

Result Analysis

4.1 Simulation Environment of King UP

In our simulation, we set the game size smaller than the real game to compress the searching space so that we could have fewer data to check if this algorithm could evaluate the expectation thoroughly after enough training.

The environmental parameters we choose in the simulating experiments are shown in Table 4.1. In King UP, a player could get max scores in a situation where except for one target piece on the top, all other target pieces stay in the $BoardHeight - 1$ level. Therefore the best situation could be calculated as below:

$$(BoardHeight - 1) * (TargetNum - 1) + TopScore$$

In our game setting shown in Table4.1, the best score could be 22. But in the real game setting, the parameters would be a little bit different: $BoardHeight$ and $TargetNum$ would be 7 and 6; in some versions of the game, $TopScore$ could be 15 or 20. Therefore, in real game rules, the max score could be $50((7-1)*(6-1)+20)$.

In order to normalize the value, all the reward is always divided by 50; even though in our simulation, the max score is not that bigger, to stay consistent with the future expansion of the actual game setting, we still choose 50, not 22 as the hyperparameter to normalize the reward. Therefore all the Q value and $reward$ in the simulation would be between $[0, 0.44]$.

Details about notions and rules in King UP are introduced in section 2.3. The link between POMCP-PGT and King UP's parameters has been illustrated in Chapter 3. The game state of King UP would involve these parameters: $TargetCard$,$VoteCard$ and $PieceLocation$. Once these three parameters are decided, the state of the game will be determined.

$PlayerNum$	3
$TargetNum$	4
$PiecesNum$	7
$Bonus$	15
$TopScore$	10
$SetRound$	2
$BoardHeight$	5
$Boardwidth$	3
$RejectionLimit$	1

Table 4.1: Parameters Table of King UP environment

4.2 Convergence Analysis

The key foundation of our POMCP-PGT is based on Monte Carlo simulation. The core philosophy of it is that if the agent could play as many rounds as possible, the agent could master the game very well. We suppose that if the agent could master the game, they could evaluate their expectation of reward more precisely after simulation. To check the efficiency of our algorithm, we verify the convergence situation under different conditions. The results show no matter whether POMCP-PGT with UCB or POMCP-PGT with RM converges quickly.

4.2.1 Performance of POMCP with UCB

To demonstrate the convergence of POMCP-PGT with UCB, we randomly generate 1000 game state by generating $TargetCard$,$VoteCard$ and $PieceLocation$. Then we could run the algorithm since these game state, and they all converge quickily.

Figure4.1 4.2 4.3 4.4 are selected from 1000 games to show the results. Despite results among 1000 experiments demonstrating more obvious convergence, we randomly choose these four figures to show a more general situation. We verify that our algorithm could converge successfully no matter what node to start. In each images, there are 3 lines for 3 players showing how value function $Q_i(s, h)$ varies through iterations; nodes in POMCP-PGT restore other players' Q value as well except the current decision maker, so we could see the other two players' expected value in the image as well. The generative parameters of the environment are listed in appendixA.

Unlike Figure4.1 4.2 4.3 4.4 show the situation where simulation start from four mutually independent and randomly generated game states including not specified player to start, Figure4.5 4.6 4.7 4.8 illustrate four sequential results that in one single game, player 2 takes continuing four actions according to game states (the other two players make move in turn with player 2); the other two lines show the reward

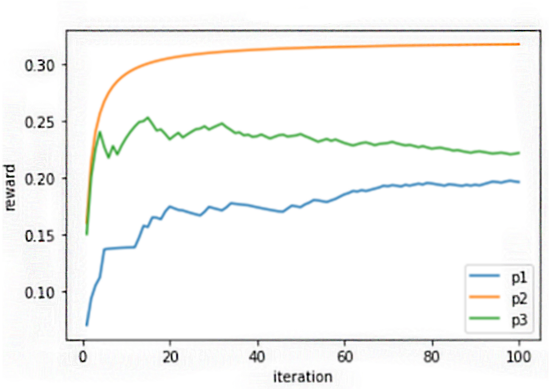

Figure 4.1: POMCP-PGT with UCB(a)

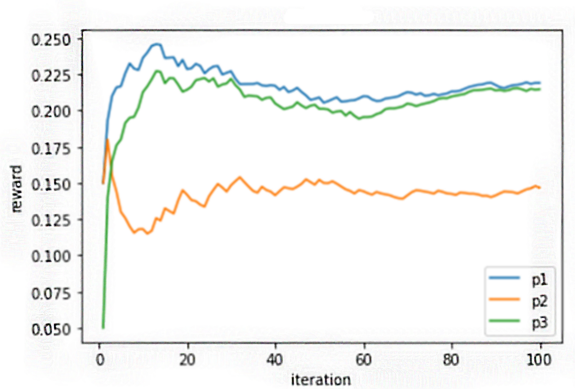

Figure 4.2: POMCP-PGT with UCB(b)

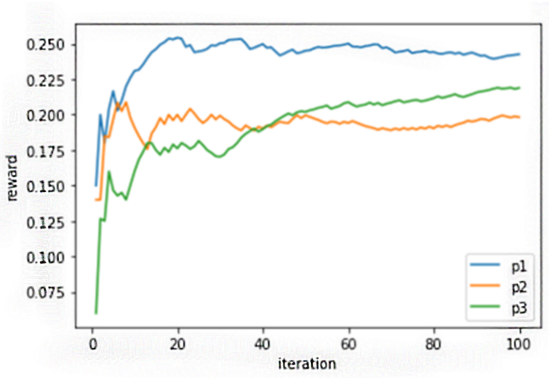

Figure 4.3: POMCP-PGT with UCB(c)

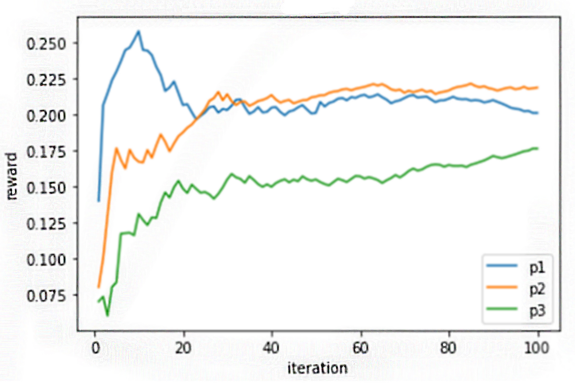

Figure 4.4: POMCP-PGT with UCB(d)

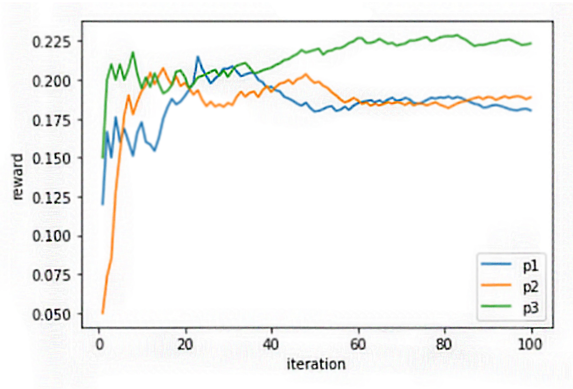

Figure 4.5: step 1(a)

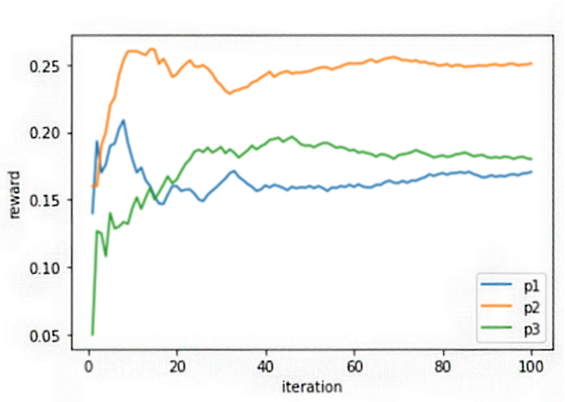

Figure 4.6: step 2(b)

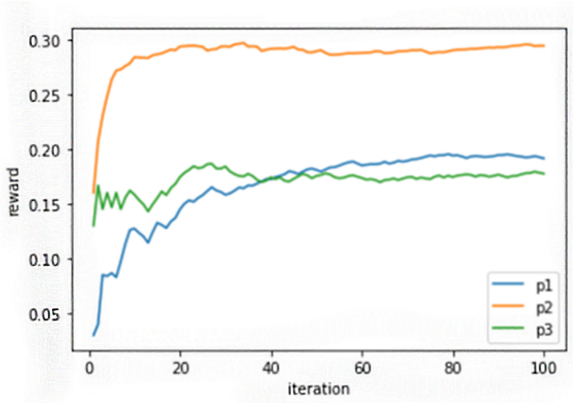

Figure 4.7: step 3(c)

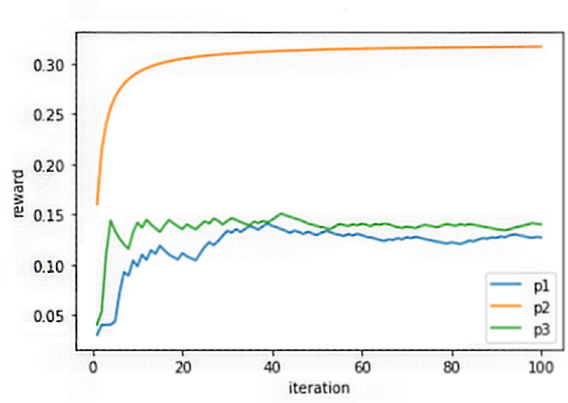

Figure 4.8: step 4(d)

expectation of player 1 and player 3 when player 2 make a move first in corresponding states; it is easier to find that the closer to the end, the player2's evaluation is more precise. There is a special case; the last picture shows that player 2 gets a very smooth curve, and it converges to 0.3 which exactly denotes a bonus situation that player 2 get bonus because of 0 credits. The parameters are listed in the AppendixA.

4.2.2 Performance of POMCP with RM

In our work, we implement Regret Matching as a replacement for UCB to compute action according to the action probability distribution. I use a random choice strategy to generate a game state and apply POMCP with Regret Matching; it turns out that they all convergence after a while; because each planning could generate images leading to a huge amount of them, there are selected convergence situations shown in Figure4.9 4.10 4.11 4.12; the results show the convergence again.

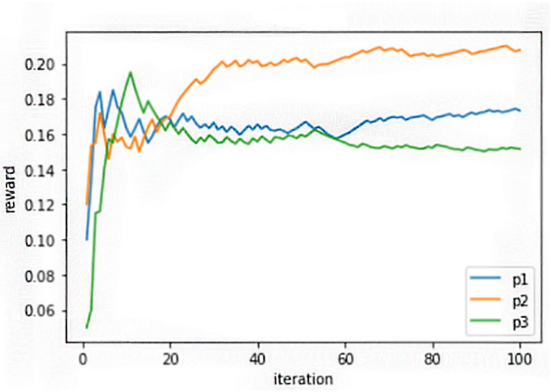

Figure 4.9: POMCP-PGT with RM(a)

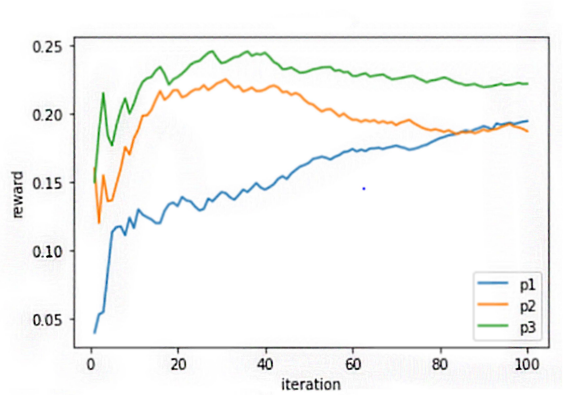

Figure 4.10: POMCP-PGT with RM(b)

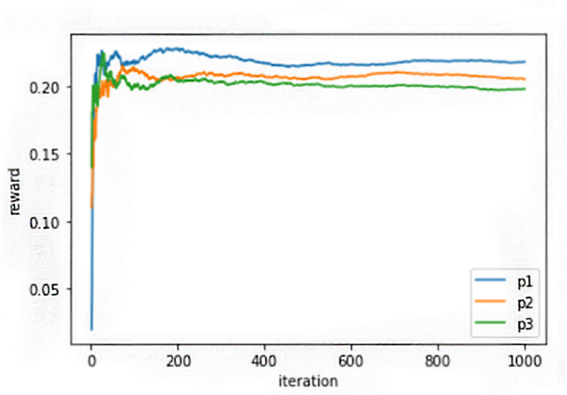

Figure 4.11: POMCP-PGT with RM(c)

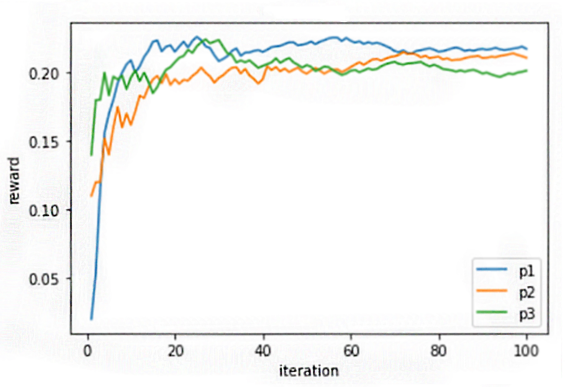

Figure 4.12: POMCP-PGT with RM(d)

4.3 Tournament Performance

We set a tournament by chosen algorithms to evaluate the performance of each algorithm by comparing them. In the following part, we use 'UCB' referring to vanilla POMCP-PGT with UCB and 'RM' referring to POMCP-PGT with regret matching. We set the environment as mentioned in section 4.1. After 1000 games, we use a binomial confidence interval to compute the winning rate(z=1.96, confidence level=95%).

2*	**Winning rate**(95%$confidenceinterval$)			**Average score**			2***Rounds**
	P1	**P2**	**P3**	**P1**	**P2**	**P3**	
(UCB, UCB, UCB)	0.366 ± 0.029	0.334 ± 0.029	0.3 ± 0.028	13.29	12.412	12.026	1000
(RM, RM, RM)	0.412 ± 0.03	0.304 ± 0.028	0.284 ± 0.028	11.8	11.7	11.6	1000
(UCB, RM, RM)	0.602 ± 0.030	0.204 ± 0.024	0.194 ± 0.024	13.05	11.14	10.83	1000
(RM, UCB, RM)	0.334 ± 0.029	0.465 ± 0.031	0.201 ± 0.025	11.994	13.778	11.228	1000
(RM, RM, UCB)	0.317 ± 0.029	0.26 ± 0.027	0.423 ± 0.031	11.996	11.152	12.852	1000
(UCB, UCB, RM)	0.457 ± 0.031	0.363 ± 0.030	0.18 ± 0.024	13.133	12.932	11.335	1000
(RM, UCB, UCB)	0.239 ± 0.026	0.415 ± 0.031	0.346 ± 0.029	11.13	12.888	12.082	1000
(UCB, RM, UCB)	0.495 ± 0.031	0.189 ± 0.024	0.316 ± 0.029	13.648	11.299	12.053	1000

Table 4.2: Tournament of POMCP-PGT with UCB and POMCP-PGT with RM

As shown in Table 4.2, we could receive straightforward feedback that for all the games that involve both POMCP-PGT with UCB and POMCP-PGT with RM, the player who uses POMCP-PGT with UCB has a higher winning rate than those who use POMCP-PGT with regret matching. Another finding is that POMCP-PGT with regret matching's performance is highly based on the position that it performs a lower winning rate as player 3 compared with the result it achieves with the role player 1 and player 2. However, POMCP-PGT with UCB gets higher winning rates. The average points POMCP-PGT with RM gets very closed (the difference is around 1). It suggests that POMCP-PGT with RM shows an effective performance.

2*	**Winning rate**(95%$confidenceinterval$)			**Average score**			2***Rounds**
	P1	**P2**	**P3**	**P1**	**P2**	**P3**	
(UCB, RM, random)	0.609 ± 0.030	0.262 ± 0.027	0.129 ± 0.020	13.727	11.597	9.051	1000
(RM, UCB, random)	0.390 ± 0.030	0.490 ± 0.031	0.12 ± 0.020	12.157	13.398	9.193	1000
(random, RM, UCB)	0.183 ± 0.024	0.348 ± 0.030	0.469 ± 0.031	9.238	12.054	13.56	1000
(RM, random, UCB)	0.350 ± 0.030	0.164 ± 0.023	0.486 ± 0.031	12.387	9.438	13.484	1000
(UCB, random, RM)	0.616 ± 0.030	0.137 ± 0.021	0.247 ± 0.027	13.79	9.27	11.736	1000
(random, UCB, RM)	0.199 ± 0.025	0.525 ± 0.031	0.276 ± 0.028	9.553	13.471	12.370	1000

Table 4.3: Tournament of random, POMCP-PGT with UCB and the one with RM

Another tournament among three agents is shown in Table4.3; they independently use three different methods: POMCP-PGT with UCB, POMCP-PGT with regret

matching, and the random choice method. The same parameters of the game are set like the former one. It could be concluded that no matter where the positions they are at, the random choice strategy shows the worst performance, and POMCP-PGT with UCB gives the best performance.

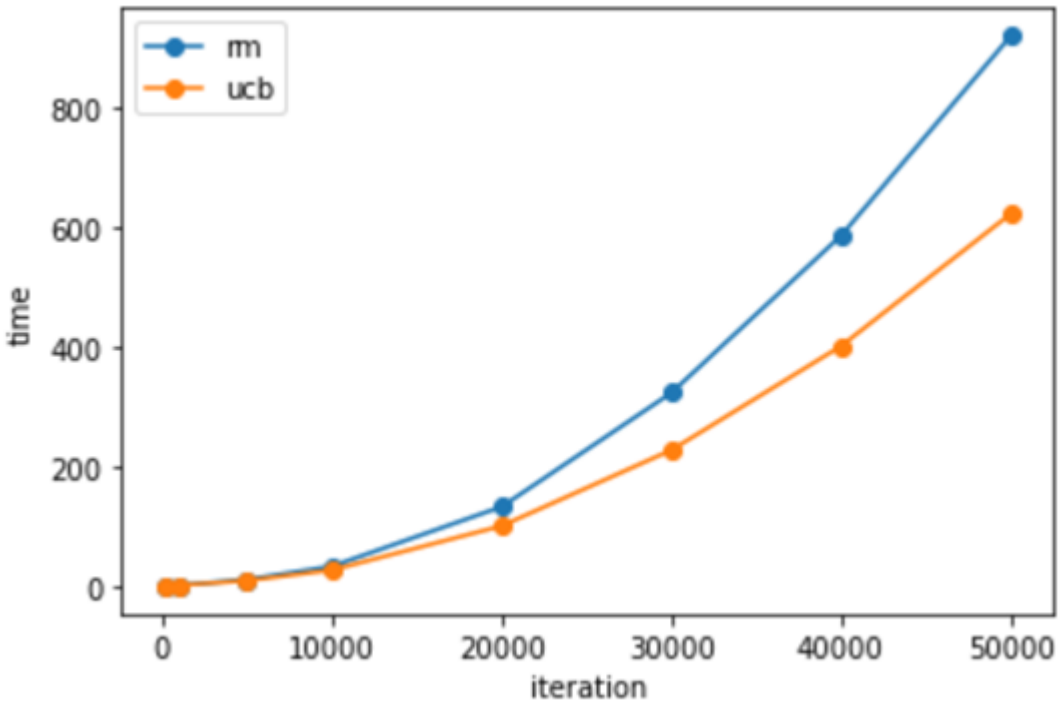

Figure 4.13: time analysis

4.4 Computational Time

Figure4.13 shows the time consumption for both POMCP-PGT with RM and POMCP-PGT with UCB. The result suggests that when expanding the exploration scale, both two algorithms show an increase in time consumption, while the one with RM always consumes more time than the one with UCB. The cause of this phenomenon should be explored in future work.

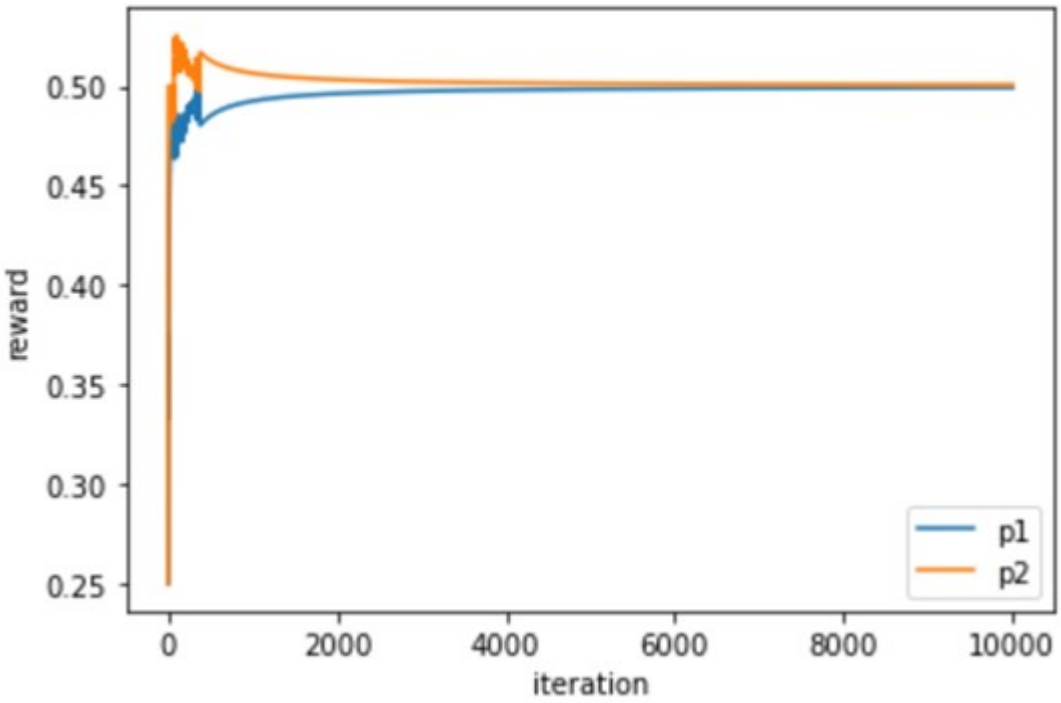

Figure 4.14: Convergency of Rock-Paper-Scissors

4.5 Generalisation

We also verify POMCP-PGT's generalization by applying it to rock-paper-scissors. We do not change the code of the POMCP-PGT and just apply it in a new game setting.
Rock-Paper-Scissor is a classical static imperfect information game where three actions could beat each other recursively. We set the reward function as following 4.4

From Game theory, we already know that the expected value of one of the Nash equilibriums is 0.5 for both players if they are all rational. Through the figure4.14, it is

	rock	paper	scissor
rock	0.5,0.5	0,1	1,0
paper	1,0	0.5,0.5	0,1
scissor	0,1	1,0	0.5,0.5

Table 4.4: reward function of Rock-Paper-Scissor

found that the expected value converges to 0.5 after 20000 rounds of self-play, which conformed to the theoretical result. Therefore, it could demonstrate that without additional feature engineering, POMCP-PGT could be applied in other imperfect information games only if they have a public observation.

Chapter 5

Further Discussion

5.1 Belief Model - The core of decisions under uncertainty

During this project, much time is spent going through all techniques as much as possible. Too many voices and ideas look attractive in solving the game AI in partial observation. What are the similarity and differences between CFR, MCTS, and RL? What are the motivation behind MAB, POMDP, and extensive form imperfect information games? Why do we deal with the same task in so many different ways? Comparing all these methods, the fundamental idea behind all these fantastic techniques is to get the belief over the uncertainty; in other words, how possible the state could be according to observations. The idea conforms to intuition and decision theory that the clearer the states are, the easier decision can be made.
Specifically, MAB considers only the most straightforward situation of decisions under uncertain rewards but ignores the consecutive change in the environment. POMDP is a specification of MDP that the current decisions are decided by the observation; an extensive game tree could be considered as a particular model only to represent POMDP in a more game theory form; similarly, could we consider RL in a way that it depresses the model knowledge in huge table or neuron network? The common thing in these algorithms is they all try to use belief to transfer partial observation into certain states so that the problem could turn into a decision-making problem in full observation. Similiarly, POMCP-PGT also models the belief over observation for especially public information.

5.2 Why POMCP-PGT?

As introduced before, several routes exist to solve partial observable games like King Up. The first choice is chosen from learning and planning, two distinct and effective technical branches to deal with such problems.

The learning method is a series of algorithms aiming at getting a model with training data. In game AI, Reinforcement Learning (RL) is a very powerful tool to approach rational policy by interacting with the environment. The advantage of RL here is that they could represent tremendous value tabular through neural networks, and RL does not necessitate a world model; on the contrary, it cannot be explainable, and the policy is approached slowly by training(sometimes, RL is known as delayed interactive learning).

Planning usually requires a world model to simulate the task and the techniques like simulation and search to explore the optimal strategy. Usually, the precise extent of the final results by planning would depend on the extent of the exploration. Planning methods like MCTS and CFR could also give us an understanding of the decisions from a statistical way that approximates the possible solution in a large-scale searching space.

Because our game has not been played by AI before (without data), we chose the planning route with more decision transparency. Compared with CFR, POCMP requires fewer computing resources and has been proved a powerful tool in POMDP tasks. Therefore, we choose POMCP as the foundation of our research.

The nature of our POMCP-PGT is a multiple players' POMCP with some skills from CFR to deal with non-Locality problems. We can easily imagine: if there are all dynamic nodes, the whole search would be no different with POMCP; if there are all static nodes, the procedure will degrade to CFR; but whatever, the tree size of POMCP-PGT would be smaller than both POMCP and CFR because we compress many states into belief storing in fewer tree nodes. Another advantage is that the Public Game Tree we get after planning could be viewed as a world model generated by Monte Carlo Simulation.

5.3 Contribution

Our contributions are listed below:

1. In the past, research about imperfect information games is rare. Most of the research is concentrated on limited types of games like poker and dice. For other games with complicated rules, there is only a limited number of games involved, like Hanabi and Settlers of Catan. King UP is a new game that has never been solved. It contains the property, including multiple players, partial observation, partly shared goals, and a mixture of game types(dynamic and static game).

2. POMCP has been a very successful online planning algorithm for POMDP problems in the past ten years. This work is designed for one single observer, and there is no efficient solution to expand it to multiple independent observers. Multiagent settings like Dec-POMDP would use a joint view, unlike multiple independent observers. POMCP-PGT extends POMCP to a version where multiple observers make a decision in one tree by designing PGT and dealing with non-locality problems caused by multi-observers.

3. In POMCP-PGT, we evaluate that no regret learning (from CFR) could be a replacement for UCB and shows equivalent performance. It suggests that in MCTS-like algorithms, no regret learning like regret matching could be a substitution. Moreover, it is a vital step to combine the advantages of CFR and POMCP.

4. We implement a version of POMCP-PGT for a general imperfect information game. When we test it on Rock-Paper-Scissors without any change, it converges to a 50-50 winning rate; furthermore, it offers a policy reaching Nash equilibrium.

5.4 Future Works

Because of the time limitation, several works are left to be done in the future. Frist, POMCP-PGT should evaluate performance on more common tasks(games) like liar's dice or poker with state-of-the-art algorithms like ReBEL; Secondly, theoretical proof is still needed despite its convergency in the experiments.

Another significant challenge in the imperfect information game is that all the planning method requires a known model to simulate the process. In other words, we need to have prior knowledge about the state behind the observation. Perhaps we should explore a way to learn the possible world model instead of being given one.

Reinforcement learning method could compute the policy without approaching the belief model but with less precision. However, all the reinforcement learning method is based on value function, and few of them considers regret. There is a possibility to transfer CFR's idea into RL.

At last, we could generalize our POMCP-PGT to other areas like robotics, computing system, or even art. In the human-robot interaction area, human-aware planning also requires a belief model over humans in partial observation.

Chapter 6

Conclusions

To review, research about complex imperfect-information games is rare in the game AI area, and most of the research is within two persons. The board game King Up is a complicated game with partial observation, partly shared goals, multiple players, and a mixture of different game types (dynamic game and static game). The current solutions for a partially observable game like Counterfactual Regret Minimization(CFR) and Partially Observable Monte Carlo Planning (POMCP) cannot solve this game well. Inspired by the task, we propose a new algorithm called Partially Observable Monte Carlo Planning in Public Game Tree (POMCP-PGT), which extends POMCP from one player's view to multiple independent players' views. In POMCP-PGT, we make three parts of changes from vanilla POMCP: 1. We define the data structure of the Public Game Tree to represent multiple players' views in one tree; 2. To deal with the non-Locality problem when extending to multiple players in PGT, we adopt the 'Local Search' procedure to deal with temporarily hidden information; 3. Apart from Upper Confidence Bound (UCB), which is generally used, regret matching (RM) is implemented as a replacement for UCB.

As the results shown in experiments, in a three-person setting, no matter whether POMCP-PGT with UCB or POMCP with RM is proved to converge to approximate value and outperform random choice strategy obviously; through the tournament among different algorithm settings, POMCP-PGT with UCB outperform POMCP-PGT with RM in winning rate, whereas their expected reward is approximately equivalent. In addition, to demonstrate our algorithm's generalization, we tested the POMCP-PGT in rock-paper-scissors, converging to a known Nash equilibrium.

POMCP-PGT is a general algorithm for multiple player imperfect information games, extending from POMCP; furthermore, it offers regret matching instead of UCB to turn POMCP from deterministic to stochastic strategy. In further work, we should compare it with more robust algorithms in King UP and try to prove the convergence of our algorithm from a theoretical level.

Appendix A

King UP setting

The game state of 4.1
TargetCard: ['D','E','F','G'], ['A','B','F','G'], ['B','C','D','E']
VoteCard:1,1,0
LocationPiece:D:3,E:2,G:3
The game state of 4.2
TargetCard:['A','B','F','G'],['B','C','D','E'],['A','B','C','G']
VoteCard:1,0,0
*LocationPiece*A:0,B:0,D:4,E:4,F:3
The game state of 4.3
TargetCard['A','E','F','G'],['B','C','D','E'],['A','B','C','D']
*VoteCard*1,0,1
*LocationPiece*A:0,B:0,D:4,E:4,F:3
The game state of 4.4
TargetCard: ['D','E','F','G'],['A','E','F','G'],['B','C','D','E']
VoteCard1,1,1
LocationPieceA:3,B:0,D:4,E:4,G:2

The game state of 4.9
TargetCard: ['A','B','F','G'],['B','C','D','E'],['A','E','F','G']
VoteCard:0,0,0
LocationPiece:A:,B:2,C:3,D:3,E:3,G:3
The game state of 4.10
TargetCard:['D','E','F','G'],['B','C','D','E'],['A','B','F','G']
VoteCard:1,1,1
*LocationPiece*A:4,B:4,D:3,E:4,F:2,G:2
The game state of 4.11

$TargetCard$['A','B','F','G'],['A','B','C','D'],['B','C','D','E']
$VoteCard$0,0,1
$LocationPiece$A:4,B:0,C:3,E:0,G:2
The game state of 4.12
TargetCard: ['A','B','C','D'],['A','B','F','G'],['B','C','D','E']
VoteCard1,1,1
LocationPieceA:0,B:0,C:1,E:0,G:1

The game state of 4.5
$TargetCard$: ['A','B','C','G'], ['D','E','F','G'], ['A','B','C','D']
$VoteCard$:1,0,1
$LocationPiece$:A:2,B:2,D:0,E:4,F:4
The game state of 4.6
$TargetCard$: ['A','B','C','G'], ['D','E','F','G'], ['A','B','C','D']
$VoteCard$:1,0,0
$LocationPiece$:A:3,B:3,D:0,E:4
The game state of 4.7
$TargetCard$: ['A','B','C','G'], ['D','E','F','G'], ['A','B','C','D']
$VoteCard$:0,0,0
$LocationPiece$:A:3,B:3,D:0
The game state of 4.8
$TargetCard$: ['A','B','C','G'], ['D','E','F','G'], ['A','B','C','D']
$VoteCard$:0,0,0
$LocationPiece$:A:3,B:4,D:0

References

[1] Sarit Adhikari and Piotr Gmytrasiewicz. "Communicative Interactive Partially Observable Monte Carlo Planning". In: (2021).

[2] Christopher Amato and Frans Oliehoek. "Scalable planning and learning for multiagent POMDPs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47.2 (2002), pp. 235–256.

[4] Nolan Bard et al. "The hanabi challenge: A new frontier for ai research". In: *Artificial Intelligence* 280 (2020), p. 103216.

[5] Semanti Basu et al. "Parallelizing POMCP to solve complex POMDPs". In: *Rss workshop on software tools for real-time optimal control*. 2021.

[6] Michael Bowling et al. "Heads-up limit hold'em poker is solved". In: *Science* 347.6218 (2015), pp. 145–149.

[7] Noam Brown and Tuomas Sandholm. "Baby Tartanian8: Winning Agent from the 2016 Annual Computer Poker Competition." In: *IJcAI*. 2016, pp. 4238–4239.

[8] Noam Brown and Tuomas Sandholm. "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals". In: *Science* 359.6374 (2018), pp. 418–424.

[9] Noam Brown et al. "Deep counterfactual regret minimization". In: *International conference on machine learning*. PMLR. 2019, pp. 793–802.

[10] Noam Brown et al. "Combining deep reinforcement learning and search for imperfect-information games". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 17057–17069.

[11] Shushman Choudhury et al. "Scalable Online Planning for Multi-Agent MDPs". In: *Journal of Artificial Intelligence Research* 73 (2022), pp. 821–846.

[12] Mihai S Dobre and Alex Lascarides. "POMCP with human preferences in settlers of catan". In: *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2018.

[13] Markus Eger, Chris Martens, and Marcela Alfaro Córdoba. "An intentional ai for hanabi". In: *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE. 2017, pp. 68–75.

[14] Christopher P Ferguson and Thomas S Ferguson. "Models for the Game of Liar's Dice". In: *Stochastic Games And Related Topics*. Springer, 1991, pp. 15–28.

[15] GH Freeman. "The tactics of liar dice". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 38.3 (1989), pp. 507–516.

[16] Quentin Gendre and Tomoyuki Kaneko. "Counterfactual Regret Minimisation for playing the multiplayer bluffing dice game Dudo". In: (2019).

[17] Alex Goldhoorn et al. "Continuous real time POMCP to find-and-follow people by a humanoid service robot". In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 741–747.

[18] Johannes Heinrich, Marc Lanctot, and David Silver. "Fictitious self-play in extensive-form games". In: *International conference on machine learning*. PMLR. 2015, pp. 805–813.

[19] Johannes Heinrich and David Silver. "Self-play Monte-Carlo tree search in computer poker". In: *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*. 2014.

[20] Johannes Heinrich and David Silver. "Deep reinforcement learning from self-play in imperfect-information games". In: *arXiv preprint arXiv:1603.01121* (2016).

[21] Levente Kocsis and Csaba Szepesvári. "Bandit based monte-carlo planning". In: *European conference on machine learning*. Springer. 2006, pp. 282–293.

[22] Marc Lanctot. *Monte Carlo sampling and regret minimization for equilibrium computation and decision-making in large extensive form games*. University of Alberta (Canada), 2013.

[23] Marc Lanctot et al. "Monte Carlo sampling for regret minimization in extensive games". In: *Advances in neural information processing systems* 22 (2009).

[24] Viliam Lis, Marc Lanctot, and Michael H Bowling. "Online Monte Carlo Counterfactual Regret Minimization for Search in Imperfect Information Games." In: *AAMAS*. 2015, pp. 27–36.

[25] Matej Moravčık et al. "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker". In: *Science* 356.6337 (2017), pp. 508–513.

[26] David Silver and Joel Veness. "Monte-Carlo planning in large POMDPs". In: *Advances in neural information processing systems* 23 (2010).

[27] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[28] David Silver et al. "Mastering the game of go without human knowledge". In: *nature* 550.7676 (2017), pp. 354–359.

[29] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[30] Jian Zhang et al. "Multi-Agent Planning under Uncertainty with Monte Carlo Q-Value Function". In: *Applied Sciences* 9.7 (2019), p. 1430.

[31] Li Zhang et al. "A Monte Carlo Neural Fictitious Self-Play approach to approximate Nash Equilibrium in imperfect-information dynamic games". In: *Frontiers of Computer Science* 15.5 (2021), pp. 1–14.

[32] Martin Zinkevich et al. "Regret minimization in games with incomplete information". In: *Advances in neural information processing systems* 20 (2007).