โครงงานเลขที่ วศ.คพ. P003-2/66/2566

เรื่อง

ปัญญาประดิษฐ์สำหรับเกมกระดานรูท

โดย

นายบางกอก วาณิชยานนท์ รหัส 630610746
นายปวเรศ ดิลกวุฒิสิทธิ์ รหัส 630610748

โครงงานนี้
เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่
ปีการศึกษา 2566

PROJECT No. CPE P003-2/66/2566

Artificial Intelligence for Root Board Game

Baangkok Vanijyananda 630610746
Pawaret Dilokwuttisit 630610748

A Project Submitted in Partial Fulfillment of Requirements
for the Degree of Bachelor of Engineering
Department of Computer Engineering
Faculty of Engineering
Chiang Mai University
2023

หัวข้อโครงงาน	: ปัญญาประดิษฐ์สำหรับเกมกระดานรูท
	: Artificial Intelligence for Root Board Game
โดย	: นายบางกอก วาณิชยานนท์ รหัส 630610746
	นายปวเรศ ดิลกวุฒิสิทธิ์ รหัส 630610748
ภาควิชา	: วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	: ผศ.ดร. เกษมสิทธิ์ ตียพันธ์
ปริญญา	: วิศวกรรมศาสตรบัณฑิต
สาขา	: วิศวกรรมคอมพิวเตอร์
ปีการศึกษา	: 2566

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่ ได้อนุมัติให้โครงงานนี้เป็นส่วน-หนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต (สาขาวิศวกรรมคอมพิวเตอร์)

..... หัวหน้าภาควิชาวิศวกรรมคอมพิวเตอร์
(รศ.ดร. สันติ พิทักษ์กิจนุกูร)

คณะกรรมการสอบโครงงาน

..... ประธานกรรมการ
(ผศ.ดร. เกษมสิทธิ์ ตียพันธ์)

..... กรรมการ
(ผศ.ดร. กานต์ ปทานุคม)

..... กรรมการ
(อ.ดร. ศศิน จันทร์พวงทอง)

หัวข้อโครงงาน	: ปัญญาประดิษฐ์สำหรับเกมกระดานรูท
	: Artificial Intelligence for Root Board Game
โดย	: นายบางกอก วาณิชยานนท์ รหัส 630610746
	นายปวเรศ ดิลกวุฒิสิทธิ์ รหัส 630610748
ภาควิชา	: วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	: ผศ.ดร. เกษมสิทธิ์ ตียพันธ์
ปริญญา	: วิศวกรรมศาสตรบัณฑิต
สาขา	: วิศวกรรมคอมพิวเตอร์
ปีการศึกษา	: 2566

บทคัดย่อ

เกมกระดาน "รูท" เป็นเกมแนวอสมมาตรที่สังเกตได้บางส่วน โดยผู้เล่นแต่เผ่าจะมีวิธีการเล่นเกมและเป้าหมายที่แตกต่างกัน รูทประกอบไปด้วยหลายเผ่าที่ผู้เล่นสามารถเล่นได้ แต่ละเผ่าจะมีวิธีการบรรลุเงื่อนไขการชนะของเกม ซึ่งก็คือการเก็บวิคทอรีพอยท์ (วีพี) ให้ได้ 30 หน่วย

รูทเวอร์ชันวิดีโอเกมได้ถูกวางหน่ายในปี ค.ศ. 2020 แต่ผลตอบรับจากผู้เล่นและชุมชนออนไลน์ออกมาว่า ปัญญาประดิษฐ์ในวิดีโอเกมรูทนั้นไม่เก่งพอ ด้วยเหตุนี้ พวกเราจึงได้สร้างระบบปัญญาประดิษฐ์ที่สามารถเล่นเกมรูทได้ขึ้นมา ในแต่ละเฟสของเกม ปัญญาประดิษฐ์ของเราจะรับสถานะปัจจุบันของเกมแล้วเลือกแอคชันที่ควรทำอย่างชาญฉลาด ในโครงการนี้ปัญญาประดิษฐ์สำหรับ 2 เผ่าหลักของเกม ได้แก่ "มาร์กีส์" เดอ แคท และ "อีรีย์" ไดนาสตี ปัญญาประดิษฐ์ 1 ตัวต่อ 1 เผ่า ได้ถูกสร้างขึ้น

อัลกอริธึมหลักที่จะในการสร้างปัญญาประดิษฐ์คือ ค้นหาต้นไม้มอนติคาร์โล อัลกอริทึมค้นหาต้นไม้มอนติคาร์โลจะเพิ่มความน่าจะเป็นที่จะชนะให้มากที่สุด โดยการจำลองการเล่นหลายๆ ครั้ง และตัดสินใจเลือกวิธีเล่นที่ดีที่สุด

อัลกอริทึมค้นหาต้นไม้มอนติคาร์โลของโครงการนี้มีพารามิเตอร์ทั้งหมด 6 ตัว ได้แก่ `reward-function`, `expand-count`, `rollout-no`, `time-limit`, `action-count-limit`, และ `best-action-policy` เราสร้างอัลกอริทึมค้นหาต้นไม้มอนติคาร์โลขึ้นมา 108 รูปแบบสำหรับแต่ละเผ่าโดยการเปลี่ยนค่าพารามิเตอร์เหล่านี้

อัลกอริทึมค้นหาต้นไม้มอนติคาร์โลทั้ง 108 รูปแบบถูกนำไปต่อสู้กับเผ่าคู่ต่อสู้ ซึ่งจะใช้อัลกอริทึมค้นหาต้นไม้มอนติคาร์โลรูปแบบพื้นฐาน รูปแบบที่มีอัตราการชนะสูงที่สุด 5 รูปแบบของแต่ละเผ่า จะถูกเลือกเพื่อนำไปต่อสู้กันเองต่อ ท้ายที่สุด จะได้รูปแบบอัลกอริทึมค้นหาต้นไม้มอนติคาร์โลที่ดีที่สุดของแต่ละเผ่ามา

รูปแบบอัลกอริทึมค้นหาต้นไม้มอนติคาร์โลที่ดีที่สุดของเผ่ามาร์กีส์ เดอ แคท มีค่าพารามิเตอร์ตามลำดับที่กล่าวด้านบนดังนี้: `vp-difference-relu`, `200`, `1`, `-1` (ไม่มีลิมิต), `-1` (ไม่มีลิมิต), และ `secure` โดยมีอัตราการชนะเฉลี่ยเมื่อสู้กับเผ่าอีรีย์ ไดนาสตี อยู่ที่ 56.0% รูปแบบอัลกอริทึมค้นหาต้นไม้มอนติคาร์โลที่ดีที่สุดของเผ่าอีรีย์ ไดนาสตี มีค่าพารามิเตอร์ตามลำดับที่กล่าวด้านบนดังนี้: `vp-difference-relu`, `200`, `1`, `-1` (ไม่มีลิมิต), `-1` (ไม่มีลิมิต), และ `secure` โดยมีอัตราการชนะเฉลี่ยเมื่อสู้กับเผ่ามาร์กีส์ เดอ แคท อยู่ที่ 63.4%

Project Title	: Artificial Intelligence for Root Board Game
Name	: Baangkok Vanijyananda 630610746
	Pawaret Dilokwuttisit 630610748
Department	: Computer Engineering
Project Advisor	: Asst. Prof. Kasemsit Teeyapan, Ph.D.
Degree	: Bachelor of Engineering
Program	: Computer Engineering
Academic Year	: 2023

ABSTRACT

The board game called "Root" is a partially observable asymmetric game where each player has different game mechanics and goals. There are many factions within Root, each having its own way to achieve the victory condition of accumulating 30 victory points (VP).

A video game version of Root was released in 2020, but the feedback from the online community stated that the artificial intelligence (AI) in the Root video game is lacking and incompetent. Due to that reason, we developed an AI system capable of playing Root. The AI agents can take game state as input and make intelligent decisions in each phase. An AI agent for each of the 2 base factions: "Marquise" de Cat and the "Eyrie" Dynasties was made in this project.

The main algorithm for implementing the AI is Monte Carlo Tree Search (MCTS). MCTS algorithm maximizes the probability of winning by simulating many rollouts and choosing the best decision.

There are 6 parameters for this project's MCTS algorithm: `reward-function`, `expand-count`, `rollout-no`, `time-limit`, `action-count-limit`, and `best-action-policy`. We created 108 MCTS agent variants for each faction by altering these parameters.

The 108 variants are pitched against the other faction with the base variant. The top 5 variants with the highest win rate for each faction are selected. The top 5 variants from each faction all play against each of the other faction's. The best variant for each faction is then selected.

The best MCTS variant for Marquise de Cat has the following parameters respective to the above parameter list: `vp-difference`, `200`, `1`, `-1 (no limit)`, `100`, and `max`. With average win rate against top 5 Eyrie Dynasties agents of 56.0%. The best MCTS variant for Eyrie Dynasties have the following parameters respective to the above parameter list: `vp-difference`, `200`, `1`, `-1 (no limit)`, `30`, and `robust`. With average win rate against top 5 Marquise de Cat agents of 63.4%.

Acknowledgments

We would like to express our deep appreciation for Asst. Prof. Kasemsit Teeyapan, Ph.D. for his exceptional dedication, expertise, and detailed guidance throughout the project, which served as a constant source of inspiration and helped ensure its successful completion.

Baangkok Vanijyananda
Pawaret Dilokwuttisit
18 February 2024

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Project rationale

Artificial intelligence (AI) has revolutionized the world of video games by giving players friends to play with or foes to play against. The digital adaptation video game of the Root board game is no exception. But there are problems with the AIs. There are multiple reports by Root video game players that the AIs are no longer challenging to play against once you start to know how to play your faction or that the AIs make awful decisions that no humans would ever make in a similar situation [Stetz, 2023] [chandl34, 2020] [CloudBoy117, 2022].

This project focuses on the recommended scenario in a two-player game of Marquise de Cat faction versus Eyrie Dynasties faction since they are both empire factions, i.e., factions with many buildings and warriors on the board, thus making this scenario have a lot of interactions.

Marquise de Cat requires players to balance building types to be able to continuously grow and earn points, while Eyrie Dynasties requires players to carefully build and follow *the Decree*, which can get very powerful, but failing to follow it will result in a setback in the form of losing points. The relevant rules of Root will be explained in this document.

Our goal is to develop an artificial intelligence (AI) system capable of playing the board game "Root". Our objective is to create an intelligent agent that can take game state as input and make intelligent decisions in each phase. The agents will be built using Monte Carlo Tree Search algorithm, and uniform random method. Full details on each AI agent implementation method and their concepts are explained in this document.

1.2 Objectives

- To create an artificial intelligence (AI) that can play the board game "Root".
- To find the best Monte Carlo Tree Search algorithm variant for playing the board game "Root" for each faction.

1.3 Project scope

The project is a software consisting of two main components: 1.) *RootMinimal*, a minimal version of Root video game. 2.) *RootTrainer*, an AI agent framework for RootMinimal. The scope of the project are as follows:

1. Implementation of RootMinimal.
2. Implementation of RootTrainer.

3. Exclude direct integration with Root video game, i.e., no pulling data from a running Root video game, and no injecting decisions into a running Root video game via code.
4. Exclude direct integration with Root board game, i.e., no computer vision will be used to detect the state of the board game.

RootMinimal supports the rules of Root (Section 2.1.1) in the scenario of Marquise de Cat versus Eyrie Dynasties. Game mechanics not used by these two factions are not included. The scope of RootMinimal are as follows:

1. Only Marquise de Cat and Eyrie Dynasties factions.
2. Only "Woodland" map.
3. Exclude *forests* area.
4. Exclude building type *ruins*.
5. Exclude dominance card mechanics.
6. Marquise de Cat starts with their *the Keep* token in the bottom right *clearing*.
7. Marquise de Cat starts with their three starting buildings pre-placed.
8. Eyrie Dynasties starts with a *roost* in the top left clearing.
9. Eyrie Dynasties starts with the *charismatic* leader.

RootTrainer has the following features:

1. Importing and exporting of RootMinimal's game state.
2. Exporting of RootMinimal's log.
3. Interface for human interaction to an instance of RootMinimal.
4. Running AI agents built using the following methods:
    - Random decision
    - Monte Carlo Tree Search (MCTS)

1.4 Expected outcomes

An AI for each faction that excels at playing as that faction is created, i.e.:

- One of the variants for Marquise de Cat AI have higher average win rate against all other Eyrie Dynasties AI variants
- One of the variants for Eyrie Dynasties AI have higher average win rate against all other Marquise de Cat AI variants

1.5 Technology and tools

Python is chosen as the language for this project's implementation due to its flexibility, useful libraries, and portability.

1.5.1 Software technology

Programming Languages:

- Python

Important Libraries:

- Pygame (UI)
- numpy
- scipy
- PyYAML
- pathos

1.6 Project plan

Task	Jun 2023	Jul 2023	Aug 2023	Sep 2023	Oct 2023	Nov 2023	Dec 2023	Jan 2024	Feb 2024	Mar 2024
RootMinimal implementation										
RootTrainer implementation										
Agent:**Random**: Implementation										
Agent:**Random**: Simulation										
Agent:**Random**: Analysis and Documentation										
Agent:**MCTS**: Implementation										
Agent:**MCTS**: Simulation										
Agent:**MCTS**: Analysis and Documentation										
Document Finalization										

1.7 Roles and responsibilities

In RootMinimal, base classes and faction-neutral functions are divided equally among the two of us. Pawaret is responsible for Marquise de Cat-related functionalities. Baangkok is responsible for Eyrie Dynasties-related functionalities.

In RootTrainer, Baangkok is responsible for implementing the base framework, the Random agent, and the One-Depth MCTS agent. Pawaret is responsible for the core General MCTS agent methods such as Selection, Expansion, Simulation, and Backpropagation. Baangkok also has some contributions to the General MCTS agent such as Parallel rollout and reward functions. Lastly, Pawaret is responsible for running and supervising the simulations.

1.8 Impacts of this project on society, health, safety, legal, and cultural issues

This project can work as an improvement and foundation for future research on game AI and other related topics. Furthermore, this project can be used to implement practical AI and integrate it into the Root video game, enhancing the gameplay experience and making the gameplay more challenging.

Chapter 2
Background Knowledge and Theory

In order to create the AI, we first need to implement RootMinimal. This requires knowledge of game programming fundamentals and extensive knowledge of the rules of Root board game. However, there are minor rule differences between Root video game and Root board game. Since Root video game's AIs are the root cause of our project, we also need to find and implement these minor differences in our RootMinimal.

The next step is to implement RootTrainer, the framework for running RootMinimal and training AI agents. There are two AI agent implementation methods that require additional self-study: Monte Carlo tree search and reinforcement learning neural networks.

2.1 Root

Root is an **asymetric game**: a game where each player starts with different setups, gameplay mechanics, etc. [Shor, 2023]. Although Root's base actions, such as move and battle, are the same for every faction, each faction starts differently, and each has its own unique mechanics and rules. Thus, Root sits closer to the asymmetric side in the symmetric-asymmetric spectrum.
Examples of symmetric games are Chess, Monopoly, Prisoner's Dilemma, etc.
Examples of other asymmetric games are Tapestry, Dune, Cthulhu Wars, etc.

Root is an **imperfect information game**: a game where there is information hidden from the player [Osborne and Rubinstein, 1994]. Most elements of Root, such as warrior placement, building count, etc., are laid out for all players to see. But cards on the player's hand are only visible to that player, i.e., other players' cards in hand are the hidden information in this game.

Root is a **non-deterministic game**: a game where there is some element of randomness involved, such as rolling dice or shuffling cards. Root has both, rolling dice in battle and drawing cards from a shuffled draw pile.

2.1.1 Brief Rules of Root

Root is a turn-based game for 2-4 players in which you fight as one of four factions against other factions to take control of the "Woodland".

The rule is simple: The player who scores 30 victory points or plays the dominance card and fulfills its condition (the card that changes the endgame win condition) first wins. The players play on the map of Woodland (Figure 2.1), which contains 12 clearings, each with different suits and a different number of slots for building. There are also paths that connect those clearings, moving warriors from one clearing to another.

Figure 2.1: Woodland, Root's game board

Table 2.1: Marquise de Cat Flow

Marquise de Cat		
Birdsong	Daylight	Evening
• Place one wood token at each sawmill.	• Craft any cards in your hand using workshops. • Take up to three of the following actions (plus one per bird suit card you spend): • Battle • March: Takes two moves • Recruit: Place one warrior at each recruiter. • Build: Place one building in a clearing you rule by spending wood tokens equal to its cost. • Overwork: Spend a card to place one wood token at one sawmill in a clearing whose suit matches the card spent.	• Draw one card plus one card per draw bonus. (Discard down to five if you have more than five cards.)
The Keep: Only you can place pieces in the clearing with the keep token. **Field Hospital**: Whenever a Marquise warrior is removed, you may spend a card matching its clearing to move the warrior to the clearing with the keep token.		

Each player will take his turn throughout three phases: birdsong, daylight, and evening. The player will be instructed on what he must do and what actions he can take in each phase. Also, each faction will have special abilities that can be used during their turn. Instructions and actions will vary from faction to faction, but there are actions that are shared by all factions, called key actions. There are three key actions:

1. **Move**: Take any number of your warriors from one clearing and move them to one adjacent clearing.
2. **Craft**: Using crafting pieces you have to craft a card from your hand. After crafting, gain benefits or effects on the crafted card.
3. **Battle**: Choose a clearing where you have warriors as the clearing of battle. You are the attacker. Choose another player in the clearing of battle to be the defender. Roll two dice. The attacker deals hits equal to the higher roll, and the defender deals hits equal to the lower roll. Both players deal hits simultaneously and remove pieces equal to the number of hits they receive.

Other than key actions, there will be some specific actions belonging to each faction. We will describe them in detail with an explanation of how each phase of each faction works in Tables 2.1 and 2.2.

Some of the rules will not be explained here, as they are not crucial or essential. For more details on rules, see [Cole Wehrle, 2565].

Table 2.2: Eyrie Dynasties Flow

Eyrie Dynasties		
Birdsong	Daylight	Evening

• If your hand is empty, draw 1 card.
• Add one or two cards to the Decree.
• If you have no roosts, place a roost and 3 warriors in the clearing with the fewest total pieces.
	• Craft using roosts.
	• Resolve the Decree from left column to right, taking one action per card in a matching clearing
	  • Recruit
	  • Move
	  • Battle
	  • Build: Place roost
		• Score victory point of rightmost empty space on the Roosts track.
		• Draw one card plus one card per draw bonus. (Discard down to five if you have more than five cards.)

Lords of the Forest: You rule any clearings where you are tied in presence. **Disdain for Trade**: When crafting items, you score only 1 victory point.		

2.1.2 Rule difference of Root video game and Root board game

In Root board game, Marquise de Cat's Field Hospital ability can be triggered at anytime the Marquise de Cat warriors are removed from the board. But in the Root video game, this ability can only be triggered when in Marquise de Cat's turn.

Also, Marquise de Cat is not able to choose which wood is spent on building. In Root video game, wood is automatically removed from the board by removing the wood that is closest to the Keep first.

2.2 Game tree

A game tree is a directed graph that represents possible game states and decisions within a game. Each node represents a state of the game, and each outgoing edge represents a decision at that state (node), which will lead to another state (node). The size of the game tree can be used to measure the complexity of a game, as it represents all the possible ways a game can pan out.

A complete game tree is a game tree with all states and decisions. It is possible to generate a complete game tree for a perfect information game. Given a complete game tree, it is possible to find a series of decisions from our current state that lead to a winning state.

Root is an imperfect information game, which means generating a complete game tree for it is not possible. Therefore, an algorithm that uses the complete game tree of Root to find a winning strategy is not possible. But even for Chess and Go, which do have a complete game tree, algorithms that play these two games still don't use the complete game tree since generating a complete game tree for a game with such high complexity as Chess and Go requires such a high amount of memory and therefore is not practical for modern computers. Instead, the algorithms that play these games use partial game trees.

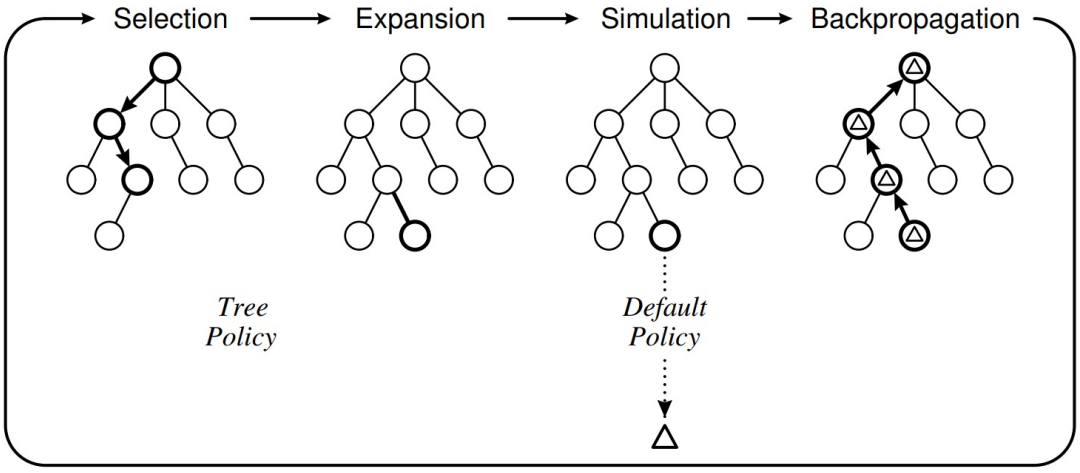

Figure 2.2: MCTS steps, credit to [Browne et al., 2012]

2.3 Monte Carlo tree search

Monte Carlo tree search (MCTS) [Bradberry, 2015] is a search algorithm commonly employed in decision processes, particularly within software designed for playing board games. In such applications, MCTS is used to solve the game tree, i.e., identify the decisions that maximize the probability of leading the player to a winning state. This is done by running many playouts. The results of each playout are then propagated back through the game tree to update the probability of that state being chosen more or less. Each round of MCTS consists of four steps: Selection, Expansion, Simulation, and Backpropagation, as shown in figure 2.2.

2.3.1 MCTS Steps

Selection

Starting at the root node, select the child node by tree policy. Then recursively select child nodes through the tree until an "expandable node" is reached. A node is expandable if it represents a nonterminal state and has unvisited children (i.e. unexpanded).

Expansion

One (or more) child nodes are added to expand the tree, according to the available actions.

Simulation (Rollout)

A simulation is run from the new node(s) according to the default policy to produce an outcome.

Backpropagation

The simulation result is "backed up" (i.e. backpropagated) through the selected nodes to update their statistics.

2.3.2 Exploration and exploitation

Maintaining a balance between exploration and exploitation is an important part of implementing decision-making algorithms such as MCTS. During MCTS processes, all nodes have some chance of being randomly picked, which allows new decisions to be made so that new and better outcomes can be discovered. This is called exploration. At the same time, nodes that lead to a winning state will have a higher chance of being picked again in subsequent playouts. This is called exploitation.

2.3.3 UCB policy

One of the most commonly used tree policies in MCTS is UCB (Upper Confidence Bound) [Russell and Norvig, 2020]. As UCB is commonly used in multi-armed bandit problem, the process in selection phase can be modelled accordingly. A child node j is selected to maximize:

$$\text{UCB} = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{2.1}$$

where n is the number of times a parent node has visited, n_j is the number of times a child node has visited, X_j is the average reward of the parent node, and $C_p > 0$ is a constant.

In (2.1), the two terms are the exploitation term and the exploration term, respectively. If a node has been explored a few times, the exploration term will be high as a denominator n_j. C_p is a constant that balances between exploitation and exploration.

2.3.4 Winning action selection

MCTS continues to run four steps until the endgame condition is met or the computation budget is reached. Then, the action of the root node is selected by some criteria.
Browne says that there are four criteria for selecting winning action, based on their survey [Browne et al., 2012]:

1. **Max**: Select the action that gives the most rewards.
2. **Robust**: Select the action that visited the most.
3. **Max-Robust**: Select the action that has the most rewards and visits. If none exist, continue searching until they are found.
4. **Secure**: Select the action that maximizes the lower confidence bound.

2.3.5 MCTS with imperfect information game and uncertainty

In a deterministic game, MCTS can generate future states in nodes that will be found when performing an action. This is called a Closed Loop MCTS [Liebana et al., 2015]. But when nondeterministic actions come into concern, such as rolling dice or drawing random cards, we can still use a Closed Loop MCTS to generate all available possible outcomes as states and play out each one. However, it should be noted that this approach will make the game tree grows exponentially.

An approximate solution is to use an Open Loop MCTS. Open Loop MCTS only stores the statistics in the nodes, not the states. Instead, the state at a node must be computed by playing actions along the path from the root node to the current node. This way, a node does not represent a state but a series of actions executed starting from the initial state of the game.

To handle imperfect information and uncertainty, a game can be converted into a deterministic game using *determinization*, the process of sampling all possible outcomes of all chance events, making it fully observable [Browne et al., 2012]. For example, rolling dice can be simulated with a predetermined sequence of dice rolls.

Chapter 3
Project Structure and Methodology

The two main components of this project: RootMinimal and RootTrainer.

3.1 RootMinimal

RootMinimal is the game environment that mimics Root video game in a game of Marquise de Cat versus Eyrie Dynasties. It has the following features:

- Simulate the current state of the game
- Generate all possible actions ("legal actions") for the current player at the current state of the game
- Change state according to the selected action

3.2 RootTrainer

RootTrainer is the framework that allows humans and AI agents to interact with RootMinimal instance's environment.

There are 2 agent implementation methods: Random decision and Monte Carlo Tree Search (MCTS).

3.2.1 Random decision

The agent constructed using the random decision method will uniformly select an action from all legal actions at the current state. The list of legal actions has a discrete uniform distribution, with each action having an equal likelihood of being chosen.

3.2.2 Monte Carlo tree search (MCTS)

The agent will perform a tree search at the current state, determine which action has the highest probability of winning the game, and take that action. The winning probability of each action will be calculated using the MCTS approach.

One-Depth MCTS

"One-Depth MCTS", similar to *Flat Monte Carlo* [Browne et al., 2012], is an MCTS agent that we implemented as a proof-of-concept for demonstrating that MCTS algorithm works with Root board game. It is called One-Depth because it only performs the legal action then immediately rollout from that state; not making any expansion down more than the first depth.

General MCTS

"General MCTS" is the main MCTS algorithm in our project. We use a specific variant of MCTS called Open-Loop MCTS as a basis of implementation for this agent due to nondeterministic mechanism in this board game and use the UCB policy as a tree policy.

General MCTS has multiple parameters which can be customized to create multiple variants of MCTS. There are 6 parameters that we implemented, including: `reward-function`, `expand-count`, `rollout-no`, `time-limit`, `action-count-limit`, and `best-action-policy`.

1. **`reward-function`**: The MCTS must determine how "good" the state at the end of a rollout is. This is done using the reward-function. There are 4 options for this parameter:
    - `win`: considers whether the player wins; 1 if win, 0 otherwise
    - `vp-difference`: considers the VP differences of the current player and the enemy player; `vp_player` - `vp_enemy`
    - `vp-difference-bin`: considers the VP differences is `d`; 1 if `d` > 0, 0 otherwise
    - `vp-difference-relu`: considers the VP differences is `d`; `d` if `d` > 0, 0 otherwise
2. **`expand-count`**: Whenever the MCTS expands the game tree, a node is added to the tree. This parameter limits the number of expansion. We could expand indefinitely until the entire tree is searched, which should result in a better outcome, but that would take too much computation time. That is the reason that this parameter exists.
3. **`rollout-no`**: Whenever the MCTS expands, it must perform a number rollouts to determine the reward of that node. This parameter controls the amount of rollouts per node. A rollout is bound to come across some nondeterministic actions which could result in varying outcome of that rollout. Multiple rollouts per node can be used to help mitigate the varying outcomes.
4. **`time-limit`**: A rollout may encounter an infinite loop or may take a very long time. This parameter controls the time limit for each rollout. A rollout will immediately stop at its current state if a time limit is exceeded.
5. **`action-count-limit`**: A rollout does not necessarily need to reach a game-ending state to stop. This parameter controls how many actions will be executed before stopping a rollout, i.e., how deep into the future will a rollout look.
6. **`best-action-policy`**: As described in Section 2.3.4, when the MCTS algorithm finishes, it must choose which legal action is the best. This parameter controls how the "best" legal action is picked. There are 4 options for this parameter:

- `max`: picks the legal action with the highest reward
- `robust`: picks the most visited legal action
- `UCB`: picks the legal action that maximises the upper confidence bound
- `secure`: picks the legal action that maximises the lower confidence bound

Chapter 4
Experimentation and Results

4.1 Experiment Setup

4.1.1 Introductory Experiment

For the introductory experiment, we will use random decision agent as it has the most basic implementation. Since we have not implemented RootTrainer, we will conduct experiments by simulating RootMinimal with random actions option turned on, and collect the results.

We will run 100,000 playouts using a random decision agent as each faction and collect the following statistics:

- winning faction: Marquise de Cat / Eyrie Dynasties
- winning condition: 30 victory points (vp)
- number of turns played (the number of birdsong phase played)
- whose turn was it when the game ends: Marquise de Cat / Eyrie Dynasties
- victory points of Marquise de Cat when the game ends
- victory points of Eyrie Dynasties when the game ends

4.1.2 Main Experiment

The main experiment consists of 2 phases. We will make multiple variants of MCTS and have them play against a "base" agent.

Setup

The base agent is an MCTS agent. It was used to play against the Random agent. The results shows that both Marquise de Cat and Eyrie Dynasties, with base agent are able to consistently win most of the rounds against the Random agent. This means that we cannot use the Random agent as base since all MCTS agents would likely have very high win rate.

The base agent's MCTS parameters are as follows:

1. **`reward-function`**: `vp-difference`
2. **`expand-count`**: 100
3. **`rollout-no`**: 1
4. **`time-limit`**: -1
5. **`action-count-limit`**: 20

6. **best-action-policy**: robust

The main experiment's MCTS variants are permutations of these parameters:

1. **reward-function**: win, vp-difference, vp-difference-relu (vp-difference-bin is excluded because it is the same as vp-difference-relu but is unable to separate a "very good" rollout from a "good" rollout.)
2. **expand-count**: 50, 100, 200
3. **rollout-no**: 1 (fixed to 1 because we have, in a way, eliminated nondeterminism in our simulation)
4. **time-limit**: -1 (means no time limit. because the rollout time are so fast that the time limit is ignorable)
5. **action-count-limit**: 20, 100, 200, -1 (-1 means no limit)
6. **best-action-policy**: max, robust, secure

These parameters resulted in 108 variants for each faction.

Phase 1

For the first phase, as there are 108 MCTS variants and 2 factions, there will be 216 battles in this phase. Each battle will be run for 100 rounds to minimize random deviation.

Phase 2

For the second phase, the top variants from the first phase will be selected to continue in this phase. The metrics of selection is "win rate" against the base variant. Unless some variant achieved 100% win rate (win all 100 rounds), it is unlikely that the variants will have overlapping win rates, though if that were to be the case, other metrics will be used such as "average turns to win", etc. The top 5 variants for each faction will be selected. Then they all play against each of the other faction's, i.e., a "Team Round-Robin". There will be 25 battles in this phase. Each battle will be run for 1000 rounds.

Final

Finally, the best performing variant from each faction will be selected to be the "best MCTS variant" for that faction.

4.2 Results

4.2.1 Introductory Experiment's Results

Figure 4.1 shows that the Marquise random agent outperforms the Eyrie random agent. This result proves our assumption that the Marquise faction has a low-complexity, no-penalty gameplay mechanism that makes it easier to score a victory point than the Eyrie faction. It corresponds to the fact that the mean victory point of the Marquise when losing is higher than that of Eyrie, as in Figure 4.3.

Figure 4.2 shows that the average number of turns to win for Marquise and Eyrie is 30.453 and 30.252, respectively. We can establish the baseline performance of our AI model using this statistical data.

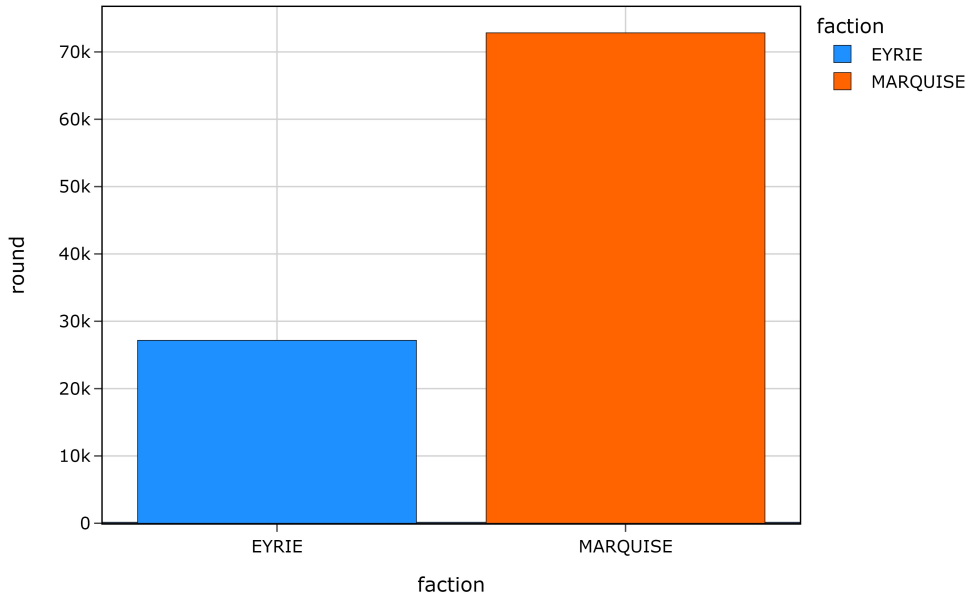

Figure 4.1: The number of wins of each faction

4.2.2 Main Experiment's Result

Phase 1

In this section, parameters `rollout-no` and `time-limit` are not displayed in the tables as they are fixed to 1 and -1 (no-limit), respectively.

The results of top 5 MCTS variants for Marquise de Cat and Eyrie Dynasties versus the base variant are displayed in Table 4.1 and 4.2

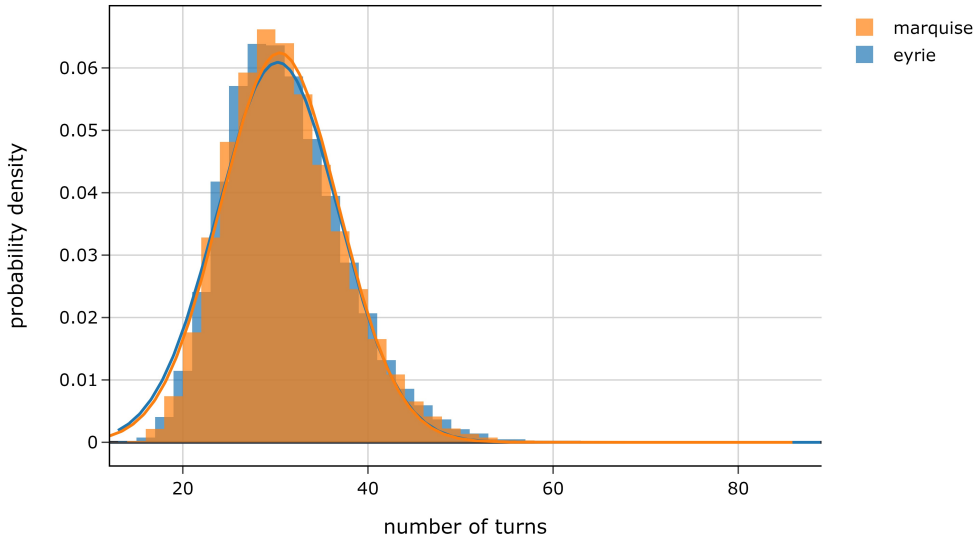

Figure 4.2: The distribution of the number of turns to win for each faction

Table 4.1: Marquise de Cat's top 5 MCTS variants versus the base variant

config no.	**win rate**	**average match length**	**reward function**	**expand count**	**action count limit**	**best action policy**
86	0.61	16.787	`vp-difference`	200	100	secure
76	0.55	15.418	`vp-difference`	200	20	robust
85	0.53	15.547	`vp-difference`	200	100	max
87	0.53	16.245	`vp-difference-relu`	200	100	max
89	0.52	15.327	`vp-difference-relu`	200	100	secure

Marquise de Cat's win rate ranges from 0.52 to 0.61, while Eyrie Dynasties's ranges from 0.71 to 0.81. We speculated that this was due to the difference in "potential" between the two factions. Marquise de Cat's playstyle is straight forward because of the fixed action count per turn, while Eyrie Dynasties has the Decree, which dictates what actions the Eyrie Dynasties player can perform. The Decree, when built and followed properly, can grant the Eyrie Dynasties player over twice the action count per turn when compared to Marquise de Cat, which grants more potential to earn VPs while also punishing with lose of VPs if played incorrectly.

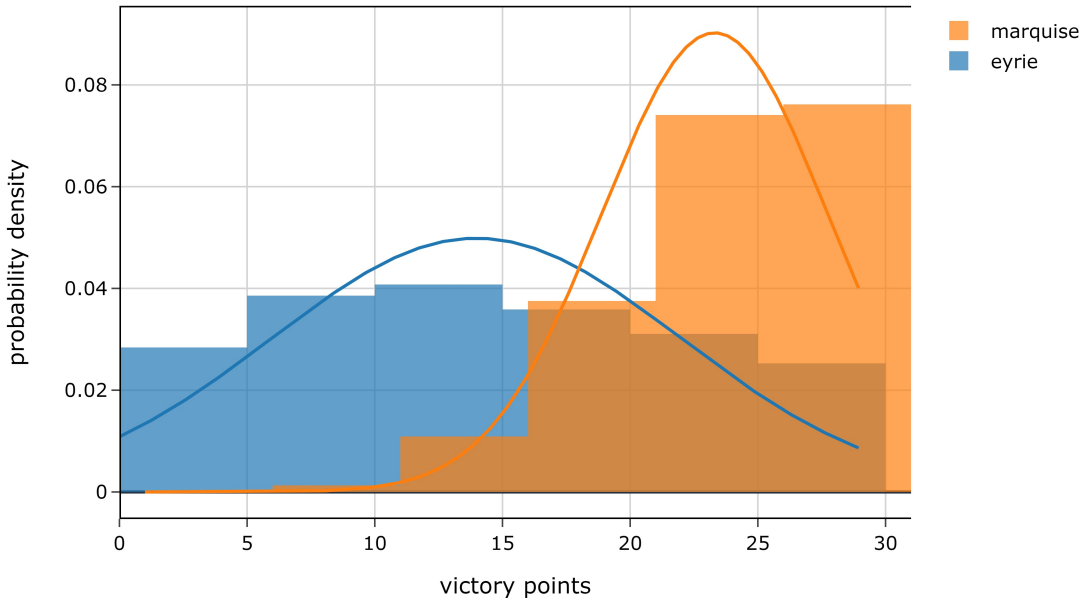

Figure 4.3: The distribution of victory points for each faction when losing

Table 4.2: Eyrie Dynasties's top 5 MCTS variants versus the base variant

config no.	**win rate**	**average match length**	**reward function**	**expand count**	**action count limit**	**best action policy**
184	0.81	14.519	`vp-difference`	200	20	robust
203	0.72	15.722	`vp-difference`	200	200	secure
194	0.71	15.690	`vp-difference`	200	100	secure
212	0.71	16.676	`vp-difference`	200	-1	secure
202	0.71	16.803	`vp-difference`	200	200	robust

Speculation on the impact of each parameters:

- Most variant has `vp-difference`-based reward function. We speculated that this is due to the `win` option only working if the simulation step reaches a game-ending state and `vp-difference-relu` option not being able to differentiate a bad action from a very bad action.
- Every variant has expand count of 200. We speculated that if the MCTS is able to

look forward further into the future and see more possible outcomes, it can then select better paths.

- The action count limit of 100 is the majority for Marquise de Cat while Eyrie Dynasties has a varying action count limit. We speculated that the action count limit does not necessarily need to be high, but it needs to be at the right value that represents the outcome of a series of actions from the root state.
- The best action policy also varies between variants. We speculated that each option for this parameter in itself does not impact much, but if combined with fitting options for other parameters, it can then shine.

Phase 2

In this section, parameters `rollout-no` and `time-limit` are not displayed in the tables as they are fixed to 1 and -1 (no-limit), respectively.

The results of top 5 MCTS variants for Marquise de Cat and Eyrie Dynasties versus the other faction's top 5 MCTS variants are displayed in Table 4.3 and 4.4. The variant with highest win rate in each faction is highlighted in bold.

Table 4.3: Marquise de Cat's top 5 MCTS variants versus Eyrie Dynasties's top 5 MCTS variants

config no.	win rate	average match length	reward function	expand count	action count limit	best action policy
86	0.532	15.363	`vp-difference`	200	100	secure
76	0.390	15.727	`vp-difference`	200	20	robust
85	**0.560**	**15.389**	**`vp-difference`**	**200**	**100**	**max**
87	0.422	15.250	`vp-difference-relu`	200	100	max
89	0.374	15.230	`vp-difference-relu`	200	100	secure

The variant with the highest win rate versus the other faction's top 5 MCTS variants for Marquise de Cat is config number 85 with 0.560 win rate and 15.389 average match length. Eyrie Dynasties's is config number 184 with 0.634 win rate and 13.681 average match length.

With the result of the best variants, we speculated that the `vp-difference` reward function is the best since it can fully represent both negative, positive, and neutral outcomes. The expand count of 200, which is highest amongs our setups, directly impacts the intelligence of the MCTS algorithm, but if it is too high, it will take too long to compute. The action count limit: for Marquise de Cat, certain actions will show significant result after a while,

Table 4.4: Eyrie Dynasties's top 5 MCTS variants versus Marquise de Cat's top 5 MCTS variants

config no.	win rate	average match length	reward function	expand count	action count limit	best action policy
184	**0.634**	**13.681**	**vp-difference**	**200**	**20**	**robust**
203	0.520	15.388	vp-difference	200	200	secure
194	0.580	15.217	vp-difference	200	100	secure
212	0.450	15.735	vp-difference	200	-1	secure
202	0.538	15.517	vp-difference	200	200	robust

such as building sawmills — you will get wood in later turns instead of the turn you build it. But for Eyrie Dynasties, every card added to the Decree will be effective immediately and every action will impact the close-future of that turn (and also far-future); A wrong action will quickly lead to a Turmoil for the Eyrie player. As for best action policies, we still cannot explain why `max` and `robust` are performing best for each faction, or if they has significant impact to the algorithm at all.

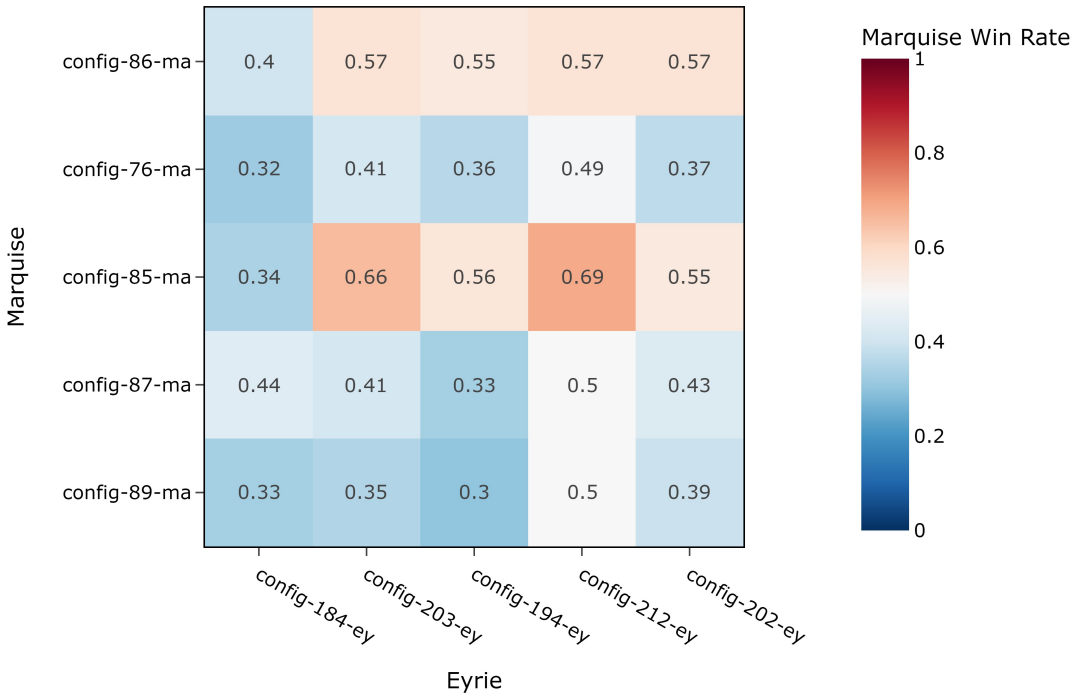

Figure 4.4: Top 5 Marquise variants win rate againts top 5 Eyrie variants: individual battle

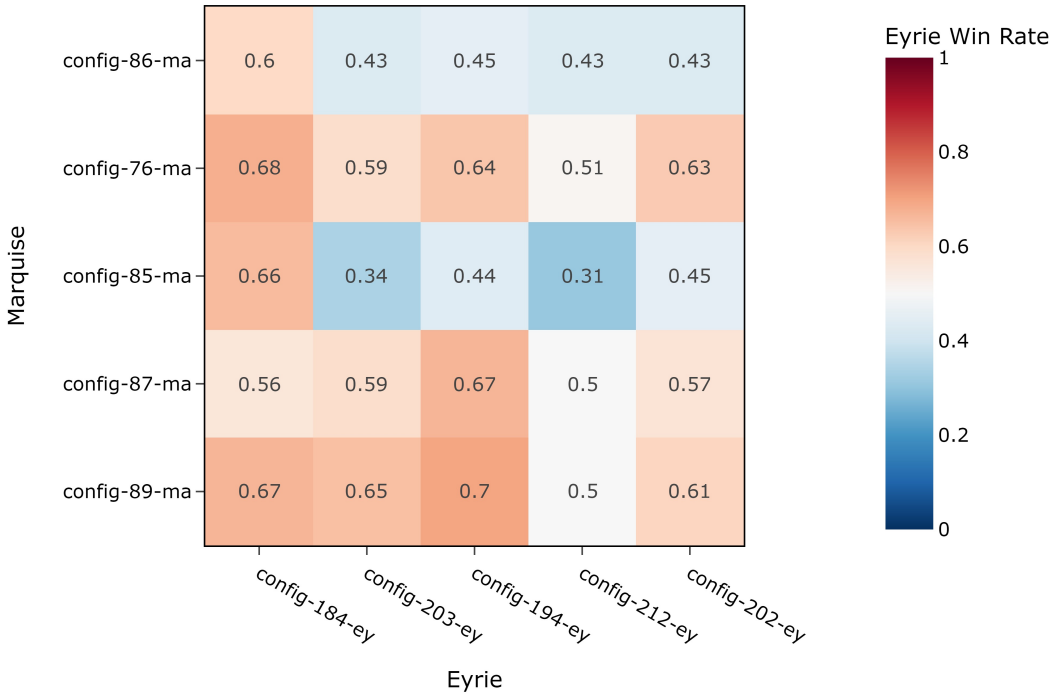

Figure 4.5: Top 5 Eyrie variants win rate againts top 5 Marquise variants: individual battle

The win rate for Marquise de Cat seems low, but if we look at individual battles in Figure 4.4, we can see that config 86 and 85 performed well; having won all Eyrie Dynasties's configs except for config 184. In contrast for Eyrie Dynasties in Figure 4.5, only config 184 won all Marquise de Cat's configs while others lost to config 86 and 85.

Chapter 5
Conclusions and Discussions

5.1 Conclusions

An online forum discussion (and our personal experiences) stated that, a normal game of Root lasts 7-10 rounds [Nebulum, 2020]. For a 2 players scenario, this is around 14-20 turns. Comparing that to our best performing MCTS AI agents which have 15.389 and 13.681 average turns to end (average match length), Marquise de Cat and Eyrie Dynasties, respectively. We have concluded that our MCTS AI agents are able to intelligently play Root board game.

The best MCTS variant for Marquise de Cat is from config number 85. It achieved average win rate against top 5 Eyrie Dynasties agents of 56.0%. It has the following parameters:

- **`reward-function`**: `vp-difference`
- **`expand-count`**: 200
- **`rollout-no`**: 1
- **`time-limit`**: -1 (no-limit)
- **`action-count-limit`**: 100
- **`best-action-policy`**: `max`

The best MCTS variant for Eyrie Dynasties is from config number 184. It achieved average win rate against top 5 Marquise de Cat agents of 63.4%. It has the following parameters:

- **`reward-function`**: `vp-difference`
- **`expand-count`**: 200
- **`rollout-no`**: 1
- **`time-limit`**: -1 (no-limit)
- **`action-count-limit`**: 20
- **`best-action-policy`**: `robust`

5.2 Challenges

- We understimated the complexity of Root board game. We estimated to complete the core implementation of RootMinimal within 3 weeks but it has taken us 3 months. We should give more consideration to carefully estimating the size of the tasks so that we can better plan out the project.

- For the MCTS algorithm to be intelligent, it must be able to look deep into multiple possibilities. To achieve this level of intelligence in an acceptable amount of time requires a high amount of computation budget. The department's server, which has a lot of CPU cores, along with us reducing the number of variants to half, helped us achieve that. Otherwise, we wouldn't have been able to simulate as many battles as we did. This is another lesson in planning for a project that requires processing power; you need a lot of processing power if you don't want to spend a lot of time.
- This project is more of a research-style than a software-development project. We have never done a project in this manner before and this is our first time doing it like this. We gained lots of experience from this.

5.3 Suggestions and further improvements

- To connect RootTrainer to an instance of Root video game and have the AIs play against each other will be the ultimate testing method of whether our AI is better than Root video game's AI.
- AIs for Root board game can be implemented using other methods such as reinforcement learning, artificial neural networks, etc.
- If the user interface is made with something else such as a game engine, human players would be able to play RootMinimal more easily then the current method of arrow keys and spacebar. Though doing that would require more time actually developing it and connecting to the logic code.
- This project does not need to be written in Python. It was written in Python because our original scope includes reinforcement learning with neural network, which we planned to use PyTorch and/or Tensorflow in those parts.
- There could be a better reward function.

References

[Bradberry, 2015] Bradberry, J. (2015). Introduction to monte carlo tree search - jeff bradberry. http://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/. Accessed: 2023-10-06.

[Browne et al., 2012] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

[chandl34, 2020] chandl34 (2020). Root: Steam review: Not recommended - chandl34. Accessed: 2024-03-14.

[CloudBoy117, 2022] CloudBoy117 (2022). Do you think the ai in this game is too easy? Accessed: 2024-03-14.

[Cole Wehrle, 2565] Cole Wehrle, K. F. (2565). Root learning to play. https://cdn.shopify.com/s/files/1/0106/0162/7706/files/Root_Base_Learn_to_Play_web_Oct_15_2020.pdf?v=1603389572.

[Liebana et al., 2015] Liebana, D. P., Dieskau, J., Hunermund, M., Mostaghim, S., and Lucas, S. (2015). Open loop search for general video game playing. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM.

[Nebulum, 2020] Nebulum (2020). Turns to win: Reddit. Accessed: 2024-03-14.

[Osborne and Rubinstein, 1994] Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*. The MIT Press, Cambridge, Massachusetts.

[Russell and Norvig, 2020] Russell, S. J. and Norvig, P. (2020). *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson.

[Shor, 2023] Shor, M. (2023). Symmetric game - game theory .net. Accessed: 2023-10-06.

[Stetz, 2023] Stetz (2023). Root: Steam review: Not recommended - stetz. Accessed: 2024-03-14.

Glossary

complete game tree a game tree with all states and decisions. 8

discrete uniform distribution a symmetric probability distribution wherein a finite number of values are equally likely to be observed; every one of n values has equal probability 1/n. 12

game tree a directed graph that represents possible game states and decisions within a game. 8, 9

multi-armed bandit problem a classic exploration-exploitation tradeoff in decision-making under uncertainty. It involves a scenario where an agent must repeatedly choose between different options (arms), each with unknown reward probabilities, while aiming to maximize cumulative rewards over time. 10

playout the act of playing a game randomly until the game ends. 9, 10, 15