โครงงานเลขที่ วศ.คพ. S003-1/66/2566

เรื่อง

ปัญญาประดิษฐ์สำหรับเกมกระดานรูท

โดย

นายบางกอก วาณิชยานนท์ รหัส 630610746
นายปวเรศ ดิลกวุฒิสิทธิ์ รหัส 630610748

รายงานนี้เป็นส่วนหนึ่งของวิชาสำรวจเพื่อโครงงาน
ตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่
ปีการศึกษา 2566

PROJECT No. CPE S003-1/66/2566

Artificial Intelligence for Root Board Game

Baangkok Vanijyananda 630610746
Pawaret Dilokwuttisit 630610748

A Report Submitted in Partial Fulfillment of Project Survey Course
as Required by 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
(ผศ.ดร. นวดนย์ คุณเลิศกิจ)

Contents

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.

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.

This project aims to create an artificial intelligence (AI) system capable of playing the board game "Root" that is more intelligent than the AIs in the Root video game. The agents will be built using different methods, including Monte Carlo tree search, neural network reinforcement learning, random, human player, etc. 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" and win against the AIs in the Root video game.

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 training 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 [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*.

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. Training and running AI agents built using the following methods:
    - Random decision
    - Monte Carlo tree search (MCTS)
    - Reinforcement learning neural network

1.4 Expected outcomes

- The resulting AIs are able to win against Root video game's corresponding opponent faction AI, i.e.:
    - Resulting Marquise de Cat AI have higher win rate against Root video game's Eyrie Dynasties AI.
    - Resulting Eyrie Dynasties AI have higher win rate against Root video game's Marquise de Cat AI.
    - Root players have a more challenging opponent when playing by themselves.

1.5 Technology and tools

Python is chosen as the language for this project's implementation due to its flexibility of integrating with AI related libraries such as PyTorch and Tensorflow.

1.5.1 Software technology

Programming Languages:

- Python

Important Libraries:

- Pygame (UI)
- PyTorch (RLNN)
- TensorFlow (RLNN)

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										
Agent:**RLNN**: Implementation										
Agent:**RLNN**: Simulation										
Agent:**RLNN**: 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, each step is divided equally, starting from method research to implementation to testing. This allows us to continuously exchange our knowledge about the current

AI agent implementation method, allowing us to create and improve each AI agent to have good performance.

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

This project can work as a foundation for future research on reinforcement learning, asymmetrical games, game AI, and other related topics.

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, and for us to create an AI to compete with Root video game's AI, 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's 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 2-4 player turn-based game where you play as one of the four factions, fighting against other factions to be the ruler of woodland.

The rule is simple: the player who gets 30 victory points or plays and completes the dominance card (the card that changes the endgame winning 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.

2.3 Monte Carlo tree search

[Bradberry, 2015] Monte Carlo tree search (MCTS) 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.

2.3.1 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.

2.3.2 MCTS with imperfect information game and uncertainty

[Liebana et al., 2015] presents that 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. 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.

2.4 Reinforcement learning

[Noparat, 2023] Reinforcement learning (RL) is a popular method used in many branches of applications, including game optimization, industry automation, telecommunications, etc. There are five main elements of reinforcement learning:

1. Agent: the player who makes a decision about what action to take
2. Environment: the world in which an agent takes actions

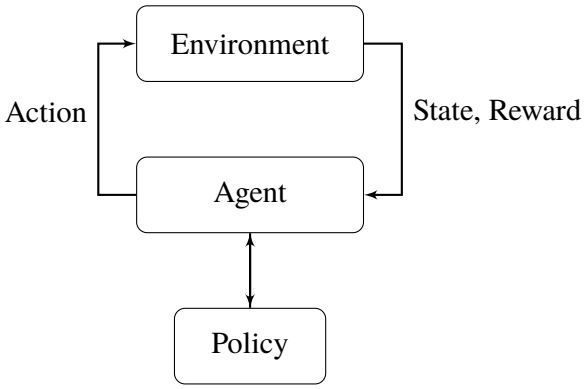

Figure 2.2: Reinforcement Learning Model

3. State: current situation of the agent
4. Reward: feedback from the environment
5. Policy: how agents make decisions

The concept of this method is to use agents to learn in a reward-punishment environment. The reinforcement learning model is described in figure 2.2. The goal of reinforcement learning is to learn a policy that maximizes the reward the agent will receive. At first, the agent will be given the objective they must achieve. At each timestep of learning, the agent will receive the current state and available actions. Then, the agent will decide what actions to take from the policy they have, moving to a new state. By using the reward function, the feedback is returned to the agent to tell how well they behave or how they take actions. The agent will use that feedback to update their policy and then take another action, repeating the process until the agent has reached its objectives.

Reinforcement learning is different from supervised learning in that there is no correct answer in reinforcement learning; instead, the reinforcement agent decides how to carry out the given task. In supervised learning, the training data includes the answer key, so the model is trained with that answer.

Chapter 3
Project Structure

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 for the current player at the current state of the game
- Change state according to the selected action
  - With option for whether the simulation will randomize the non-deterministic actions by itself or whether the player can input a specific outcome value for the non-deterministic actions.

3.2 RootTrainer

RootTrainer is the framework that trains AI agents in RootMinimal instance's environment. Due to a shortage of time and players, we use the multi-agent learning approach, where agents compete against agents rather than actual players.

There are 3 agent implementation methods: Random decision, Monte Carlo tree search, and Reinforcement learning neural network.

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 generate 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.

Because of the nondetermism mechanism in games, we use a specific variant of MCTS, Open-Loop MCTS.

3.2.3 Reinforcement learning neural network (RLNN)

The agent constructed using RLNN will use RNN type model, taking the current state, past states, and past actions as input and outputing the probability of choosing each action from all legal actions in the current state. Then calculate the reward of that action using the reward function, which is yet to be defined. Adjust the neural network according to the reward, and adjust the agent's current state to the new state in the feedback.

1. Agent: the player that's playing as Marquise de Cat faction or Eyrie Dynasties faction
2. Environment: the map, opponent's state and action, draw pile, etc. in RootMinimal
3. State: all game components that belonged to the player
4. Reward: victory points earned, getting a win, or taking hold of clearing with matching suit to the played dominance card, all combined together through the reward function
5. Policy: the neural network

Chapter 4
Experimentation and Results

4.1 Experimental Setup

4.1.1 Introductory experiment

For 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 1000 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) / dominance card (dominance)
- 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
- dominance card of the winning faction: none / bird / fox / rabbit / mouse

4.2 Results

4.2.1 Introductory experiment's results

High dominance card wins

Dominance card wins happen way more often than victory points wins as seen in figure 4.1. Our speculation is that it is the action generation in RootMinimal that makes the "Activate Dominance Card" action have a higher likelihood of being picked. RootMinimal generates actions in a manner that is more suitable for human players, e.g., in the move action, the agent needs to select a start clearing, then a destination clearing, and then how many warriors to move. These are done as three separate actions, while in reality, the action of "Move X warriors from A to B" should be counted as one action.

Marquise de Cat's high dominance card wins

Marquise de Cat have a higher chance to win with dominance card as seen in figure 4.1. Our speculation is that because at the start of the round, Marquise de Cat have control over clearings except the one that Eyrie Dynasties starts in, therefore, if Eyrie Dynasties does

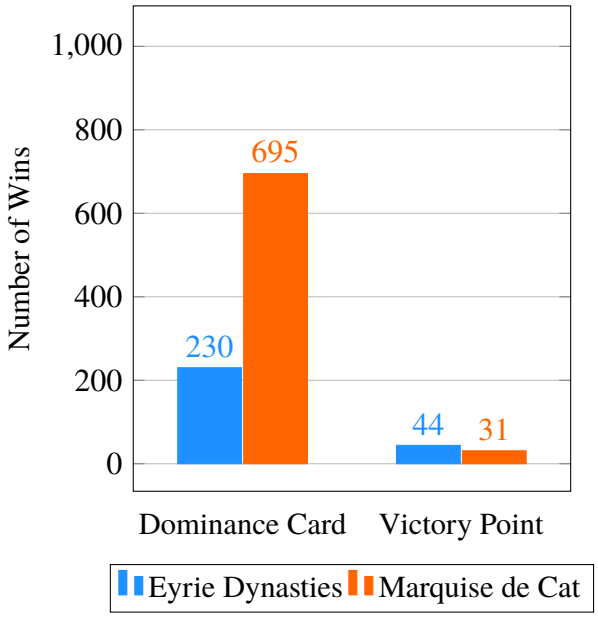

Figure 4.1: The number of wins grouped by winning condition

not play properly to take control of the clearings, Marquise de Cat will likely win upon the activation of a dominance card of any suit.

Additionally, games with Marquise de Cat's dominance card wins take fewer turns, as seen in figure 4.2. Which should be due to Marquise de Cat starting with more clearings under control.

Eyrie Dynasties's high victory points wins

Eyrie Dynasties have a higher chance to win with victory points as seen in figure 4.1. Although there are not many data for rounds with victory points wins, our experience during development where we turned of the dominance card mechanics also have more Eyrie Dynasties victory points win. Our speculation is that this is due to Eyrie Dynasties's passive earning of victory points, all they need to do is protect their roosts and don't turmoil too much. In contrast, Marquise de Cat needs to actively build buildings to earn victory points, which from the issue about action generation in 4.2.1, action of building buildings might not have the correct likelihood of being picked.

Additionally, games with Eyrie Dynasties's victory points wins take more turns than those with Marquise de Cat's, as seen in figure 4.3. Which should be due to Eyrie Dynasties having passive, continuous earning of victory points every turn while Marquise de Cat's earnings is more like a burst of victory points every time they build a building.

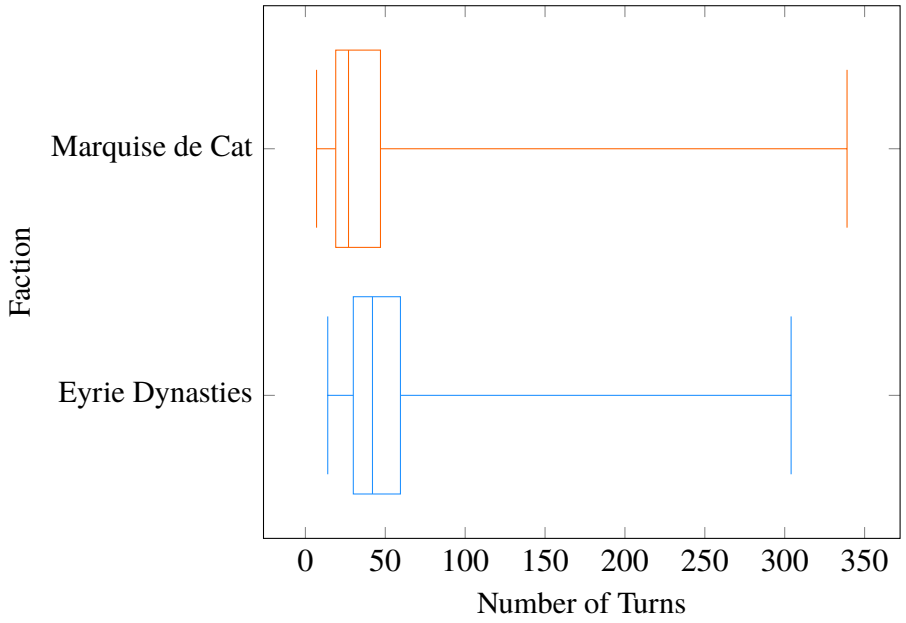

Figure 4.2: Number of turns to achieve a dominance win

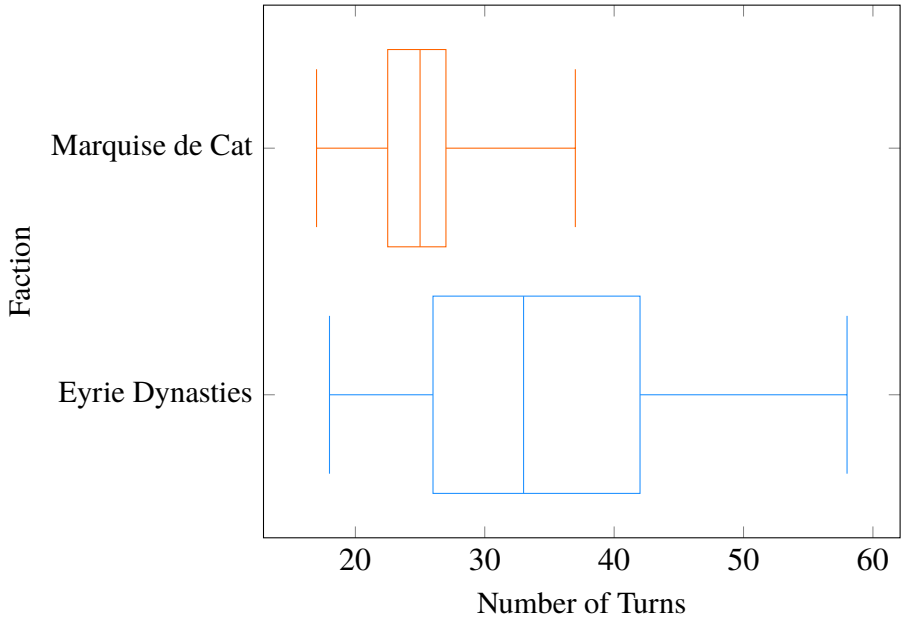

Figure 4.3: Number of turns to achieve a victory points win

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.

[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.

[Noparat, 2023] Noparat, N. (2023). มาทำความรู้จักกับ การเรียนรู้แบบเสริมกำลัง (reinforcement learning). https://bigdata.go.th/big-data-101/introduction-to-reinforcement-learning/. Accessed: 2023-10-06.

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

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