Application of Monte Carlo Tree Search Algorithm in 7 Wonders

BY

LEE JIAN ZHEN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology
(Kampar Campus)

JAN 2023

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: Application of Monte Carlo Tree Search Algorithm in 7 Wonders

Academic Session: JAN 2023

I LEE JIAN ZHEN

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

[illegible]

(Author's signature)

Heng Yew

(Supervisor's signature)

Address:

4080, Jalan Seksyen 4/1

31900 Kampar, Perak

Lee Heng Yew

Supervisor's name

Date: 27/04/2023

Date: **28/04/2023**

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN

Date: 27/04/2023

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that LEE JIAN ZHEN (ID No: 19ACB02281) has completed this final year project entitled "Application of Monte Carlo Tree Search Algorithm in 7 Wonders" under the supervision of Mr. Lee Heng Yew (Supervisor) from the Department of Application of Monte Carlo Tree Search Algorithm in 7 Wonders

I understand that University will upload softcopy of my final year in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

[illegible]

(LEE JIAN ZHEN)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**Application of Monte Carlo Tree Search Algorithm in 7 Wonders**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	[illegible]
Name	:	LEE JIAN ZHEN
Date	:	27 April 2023

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Dr. Lee Heng Yew who has given me this bright opportunity to engage in an algorithm application project. It is my first step to establish a career in algorithm application field. A million thanks to you. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

ABSTRACT

The use of Monte Carlo tree search techniques has been proven effective in enhancing artificial intelligence in various games such as Go and Hex. In this research project, we aim to analyse the effectiveness of this method in the game "7Wonders," which presents unique challenges due to its elements of hidden information and player trades. The game is also challenging to analyse because the value of each card varies greatly depending on the stage of the game and the choices made by other players. Despite its differences from abstract games, "7Wonders" shares similar underlying principles, and previous findings can still be applicable. We will focus on implementing the Monte Carlo Tree Search algorithm into the 3-player version of "7Wonders." To accomplish this, we will conduct research and literature evaluations to gain insight into how to apply algorithmic principles to various board games. By examining existing papers, we hope to develop a clearer understanding of how to implement the Monte Carlo Tree Search algorithm effectively in "7Wonders."

TABLE OF CONTENTS

TITLE PAGE I

DECLARATION OF ORIGINALITY III

ACKNOWLEDGEMENTS V

ABSTRACT VI

TABLE OF CONTENTS VII

LIST OF FIGURES IX

LIST OF ABBEREVIATIONS X

CHAPTER 1: INTRODUCTION 1

1.1 Problem Statement and Motivation 1

1.2 Project Objective 1

1.3 Project Scope 1

1.4 Contributions 2

1.5 Report Organization 2

CHAPTER 2: LITERATURE REVIEW 3

2.1 A Heuristic Monte Carlo Tree Search Method for Surakarta Chess [3] 3

Proposed Improvement 4

2.2 Enhancing Monte Carlo Tree Search for Playing Hearthstone [4] 5

Proposed Improvement 6

2.3 Recommendations 7

CHAPTER 3: SYSTEM METHODOLOGY / APPROACH 8

3.1 Understanding Monte-Carlo Tree Search algorithm 8

3.2 Understanding 7 Wonder 10

3.3 Flow Control of Game 11

3.4 Timeline 12

CHAPTER 4: SYSTEM DESIGN 13

4.1 Game Modal Design 13
Board 13
Player’s Card 14
4.3 Game AI Design 15
MCTS Algorithm 15
CHAPTER 5: SYSTEM IMPLEMENTATION 18
5.1 Hardware Setup 18
5.2 Software Setup 18
Eclipse 2023 18
Java Development Kit 17 19
5.3 Game Logic 20
Game Initialization 20
Legal Move Check 21
MCTS AI Simulation 23
5.4 System Implementation Challenges 25
CHAPTER 6: CONCLUSION 26
BIBLIOGRAPHY 27
FINAL YEAR PROJECT WEEKLY REPORT 28
POSTER 33
PLAGIARISM CHECK RESULT 34

LIST OF FIGURES

Figure 1: Surakarta Chess Boardgame..3
Figure 2: Hearthstone Gameplay Interface 5
Figure 3: MCTS Algorithm Phases...9
Figure 4: Flow Chart...11
Figure 5: Timeline Table 12
Figure 6: 7Wonder Board Interface 13
Figure 7: 7Wonder Played Card Interface 14
Figure 8: MCTS AI Pseudocode...16
Figure 9: Modified MCTS AI Pseudocode 17
Figure 10: Specification of Hardware Used..18
Figure 11: Eclipse Logo/Icon..18
Figure 12: Java Development Kit 17 Icon/Logo...19
Figure 13: Card Reader Method Code 20
Figure 14: Player's Attributes Code 21
Figure 15: Check Legal Move Method Code ~ 1 22
Figure 16: Check Legal Move Method Code ~ 2 23
Figure 17: MCTS Class Code 24

LIST OF ABBEREVIATIONS

AI	Artificial Intelligent
MCTS	Monte Carlo Tree Search
UCB	Upper Confidence Bounds
IDE	Integrated Development Environment
JDK	Java Development Kit
SDK	Software Development Kit
JVM	Java Virtual Machine
JRE	Java Runtime Environment

CHAPTER 1: INTRODUCTION

This chapter provides an overview of our research background and motivation, the contributions we have made to the field, and a summary of the thesis outline.

1.1 Problem Statement and Motivation

The use of Artificial Intelligence (AI) algorithms in game playing has been an active area of research for many years. Several well-known board games such as Chess, Go, and Gomoku have been the focus of much research in this field. However, each game has unique rules and mechanics, such as hidden information and game randomness, which require different approaches or solutions to be solved effectively. The motivation for this project is to propose an efficient algorithm that can solve a specific game and develop an AI system that can be implemented to play the game.

1.2 Project Objective

- To explore the potential of using the Monte-Carlo Tree Search algorithm in the context of the 7 Wonders game.
- To propose an efficient algorithm for the game and develop an AI system that can play the game effectively.
- To design and develop a user-friendly gaming interface for 7 Wonders.

1.3 Project Scope

The scope of this project is to investigate the potential of using Monte-Carlo Tree Search (MCTS) algorithm in the context of the 7 Wonders game. The project will focus on implementing the MCTS algorithm for 3-player gameplay and evaluating its performance against existing algorithms. The project also aims to propose an efficient algorithm that considers the unique rules and mechanics of the game and develop an AI system that can play the game effectively using the proposed algorithm. Additionally, the project will include the design and development of a user-friendly gaming interface for 7 Wonders, which will allow players to interact with the game and AI system.

1.4 Contributions

The contributions of this project can potentially enhance the performance and enjoyment of playing 7 Wonders, as well as contribute to the advancement of game-playing AI research. By exploring the potential of using MCTS algorithm in 7 Wonders, this project contributes to the development of more effective and versatile AI algorithms for a wider range of games. The proposed algorithm and AI system can also be adapted and extended to other card games or strategy games that share similar game mechanics and rules as 7 Wonders, which can have practical applications in various domains, such as gaming, education, and decision-making. Overall, the contributions of this project demonstrate the potential and versatility of MCTS algorithm in game-playing AI and provide insights and tools for future research and development in this area.

1.5 Report Organization

This report consists of six chapters that present the background, methodology, and findings of this project. Chapter 1 introduces the project, including the problem statement, objectives, project scope, and contributions, as well as the organization of the report. Chapter 2 presents a literature review of relevant papers on the algorithms used in this project and the development of the game.

Chapter 3 describes the methodology and approach used in the project, focusing on the Monte-Carlo Tree Search algorithm, and provides details about the game. Chapter 4 discusses the system design, including the architecture and components of the system. Chapter 5 presents the implementation details of the system, including the software and hardware used, and how the system was integrated.

Finally, Chapter 6 provides a conclusion to the project, summarizing the key findings and contributions, discussing the limitations and future directions of the project, and providing recommendations for further research. Overall, this report aims to provide a comprehensive overview of the project, from the problem statement and motivation to the final implementation and conclusions.

CHAPTER 2: LITERATURE REVIEW

The literature review of this research is to study the existing research paper on how Monte Carlo Tree Search algorithm can be used and improved for a boardgame. This literature review will describe and summaries the improvement done for MCTS algorithm for each of the research paper.

2.1 A Heuristic Monte Carlo Tree Search Method for Surakarta Chess [3]

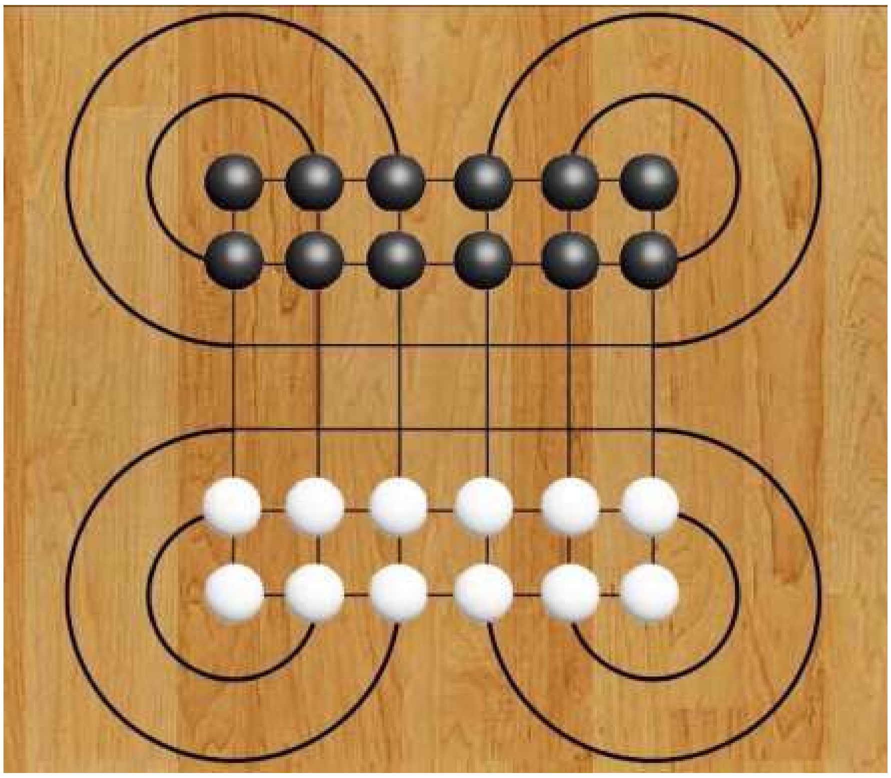


Figure 1: Surakarta Chess Boardgame

This study focuses on the Surakarta board game, a strategic game created by Indonesians, which consists of an 8-arc and 6x6 line board. The research explores the application of Monte Carlo Tree Search (MCTS) and its variations, widely used in computer games, for the Surakarta chess game. Additionally, the study proposes a heuristic search approach to address the issue of high time-complexity of moves in Surakarta chess. The research delves into the motivation behind the use of heuristic tactics and how to combine three of them to efficiently solve the problem. Overall, the study aims to enhance the understanding of Surakarta chess and provide an efficient approach for playing the game using MCTS and heuristics.

Proposed Improvement

To enhance the accuracy of the simulations in Surakarta chess game, the researchers developed an energy function consisting of several terms such as position, attack, and defense. They then modified the selection function of the MCTS to incorporate the heuristic evaluation for better performance. Additionally, they introduced a solution for the repetitive loop problem which frequently occurs in Surakarta chess game by setting a limit of 50 steps after which the simulation returns a draw as a result.

Furthermore, they implemented the root parallelization strategy to speed up the simulation process. This approach is more efficient than the conventional leaf parallelization strategy since it expands the search space and prevents the algorithm from being stuck in local optimal solutions. By using root parallelization, the researchers were able to significantly improve the performance of the MCTS algorithm in Surakarta chess game.

2.2 Enhancing Monte Carlo Tree Search for Playing Hearthstone [4]

Figure 2: Hearthstone Gameplay Interface

Hearthstone is a popular online collecting card game that presents a unique challenge to players due to the hidden hand of opponents and the uncertainty of drawing cards from their own deck. The game involves two players competing against each other using constructed decks of 30 cards and a chosen hero with a special ability. The ultimate objective is to kill the enemy's hero by using mana crystals to cast spells or summon minions to attack.

In this study, the authors propose using the MCTS algorithm to address three challenges that arise from the game's mechanics. The first challenge is the excess of hidden information and unpredictability, which AI beings must overcome to play the game effectively. The second challenge is the high degree of randomness in Hearthstone compared to other games of similar nature. Finally, the game's ability to mix any type of in-game activity presents a unique challenge for developing AI strategies.

To address these challenges, the authors suggest using the MCTS algorithm to guide the decision-making process of AI beings. The algorithm helps to predict the most likely outcomes of different actions based on simulations, taking into account the uncertainty and randomness inherent in the game. By using this approach, the AI can make more informed decisions that maximize their chances of winning.

Overall, the study suggests that using the MCTS algorithm can be a powerful tool for developing AI strategies in Hearthstone and other similar games with hidden information and randomness.

Proposed Improvement

To address the challenges of playing Hearthstone, researchers proposed a state abstraction approach that incorporates domain knowledge to handle imperfect information in the game. They also introduced a modified Upper Confidence Bound for Trees algorithm and employed a sparse sampling approach to handle randomness and incomplete information in the game.

Furthermore, the researchers developed heuristic approaches that integrate domain-specific knowledge to enhance the performance of MCTS in playing Hearthstone. These approaches were designed to improve the accuracy of simulations and decision-making processes, as well as to reduce the computational complexity of the game.

Overall, the research aimed to provide solutions to the challenges posed by Hearthstone, such as hidden information, randomness, and the ability to mix different types of in-game activities. By implementing state abstraction, modified algorithms, and domain-specific knowledge, the researchers hoped to improve the effectiveness and efficiency of MCTS in playing Hearthstone.

2.3 Recommendations

After reviewing various research papers, it has been found that integrating the MCTS algorithm with domain-specific heuristic evaluations can significantly improve the algorithm's win rate in games. Moreover, the Surakarta Chess game has introduced a rule to address the repetitive loop problem, where the MCTS algorithm stops and returns a draw as a result after 50 steps. Similarly, Hearthstone has implemented sparse sampling to handle the randomness and incomplete information in the game. Building on this previous research, the proposed project aims to modify the MCTS algorithm to increase the win rate in games by integrating domain-specific heuristic evaluations, while also using external rules to enhance the algorithm's efficiency and reduce the time complexity.

CHAPTER 3: SYSTEM METHODOLOGY / APPROACH

To successfully implement the Monte-Carlo Tree Search (MCTS) algorithm in 7 Wonders, it was essential to have a deep understanding of both the algorithm and the game. As a result, extensive research was conducted on both topics. The findings of this research are presented below, providing a comprehensive overview of the key concepts and principles that underpin the MCTS algorithm and the 7 Wonders game.

3.1 Understanding Monte-Carlo Tree Search algorithm.

The Monte Carlo Tree Search (MCTS) algorithm is a decision-making algorithm that has proven to be highly effective in solving complex games like Go, Hex, and Chess. The algorithm is designed to make optimal decisions in situations where there is a large amount of uncertainty and hidden information, such as in games with multiple players or incomplete information.

At its core, the MCTS algorithm works by building a tree-like structure that represents all possible game states and their corresponding moves. The algorithm then performs a series of simulations, called playouts, in which it randomly selects moves from the current game state and plays out the game to a terminal state (i.e., a win, loss, or draw). These playouts are used to estimate the likelihood of winning from each game state, which is then used to guide the algorithm's decision-making process.

As the MCTS algorithm performs more playouts, the tree structure is updated to reflect the new information. Specifically, the algorithm selects a node in the tree based on a selection policy, such as the Upper Confidence Bound (UCB) algorithm, which balances exploration (visiting new nodes) and exploitation (choosing nodes with high win probabilities). Once a node is selected, the algorithm performs a playout from that node to estimate its win probability. The result of the playout is then used to update the node's statistics, such as the number of times it was visited and the number of times it resulted in a win.

Over time, the MCTS algorithm converges to an optimal strategy by using the statistics in the tree to guide its decision-making process. By balancing exploration and exploitation, the algorithm can efficiently search the game tree and make optimal decisions in complex and uncertain game environments.

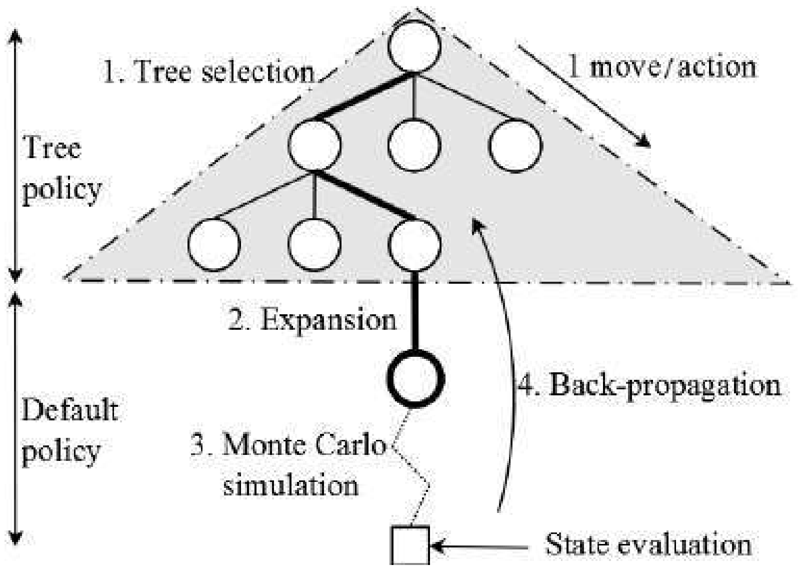


Figure 3: MCTS Algorithm Phases

Selection phase: Starting from the root node, the algorithm traverses the tree by selecting child nodes based on certain criteria until it reaches a leaf node. The selection criteria usually involve a trade-off between exploring new branches and exploiting existing information about the nodes' values.

Expansion phase: Once a leaf node is reached, the algorithm generates one or more child nodes representing possible moves or actions from that state.

Simulation phase: The algorithm then performs a simulation or playout, starting from the newly generated child node(s) until the game reaches a terminal state. The simulation is usually performed using a simple heuristic or random policy to estimate the outcome of the game.

Backpropagation phase: After the simulation, the algorithm updates the statistics of all nodes traversed from the root to the newly expanded node, including the number of visits and the cumulative score. This step involves backpropagating the simulation results up the tree, from the leaf node(s) to the root node.

These four phases are repeated many times until a certain stopping criterion is met, such as a maximum number of iterations or a time limit. The final move or action to be taken by the algorithm is usually chosen based on the statistics of the child nodes of the root node. In the context of a game, this final move represents the best move the algorithm can make based on the information gathered during the search.

3.2 Understanding 7 Wonder

7 Wonders is a board game designed for 3 to 7 players where each player begins with a unique civilization and a game board representing their city. The game is played over three ages, each representing a different historical period. The goal of the game is to accumulate the most victory points at the end of the third age.

During each age, players receive a hand of cards that represent various types of structures, such as resource-producing buildings, military structures, or scientific buildings. Each card requires a certain combination of resources to build, and some cards have special abilities that can provide bonuses or penalties. Players can choose to build the cards in their hand, sell them for money, or use them to build a Wonder stage.

Players obtain victory points through various means such as constructing buildings, achieving military victories, scientific discoveries, and making progress on their Wonders. Each age, players receive a hand of cards that represent different types of structures such as resource-producing buildings, military structures, or scientific buildings.

To construct a card, players must have the necessary combination of resources. Players can also sell cards for money or use them to build a stage of their Wonder. Trading resources with other players is also important, as it allows players to obtain the necessary resources for their constructions. However, since players cannot communicate directly with each other, they must make informed decisions about their opponents' strategies.

At the end of each age, players count their victory points based on their achievements. The player with the most victory points at the end of the third age is declared the winner.

3.3 Flow Control of Game

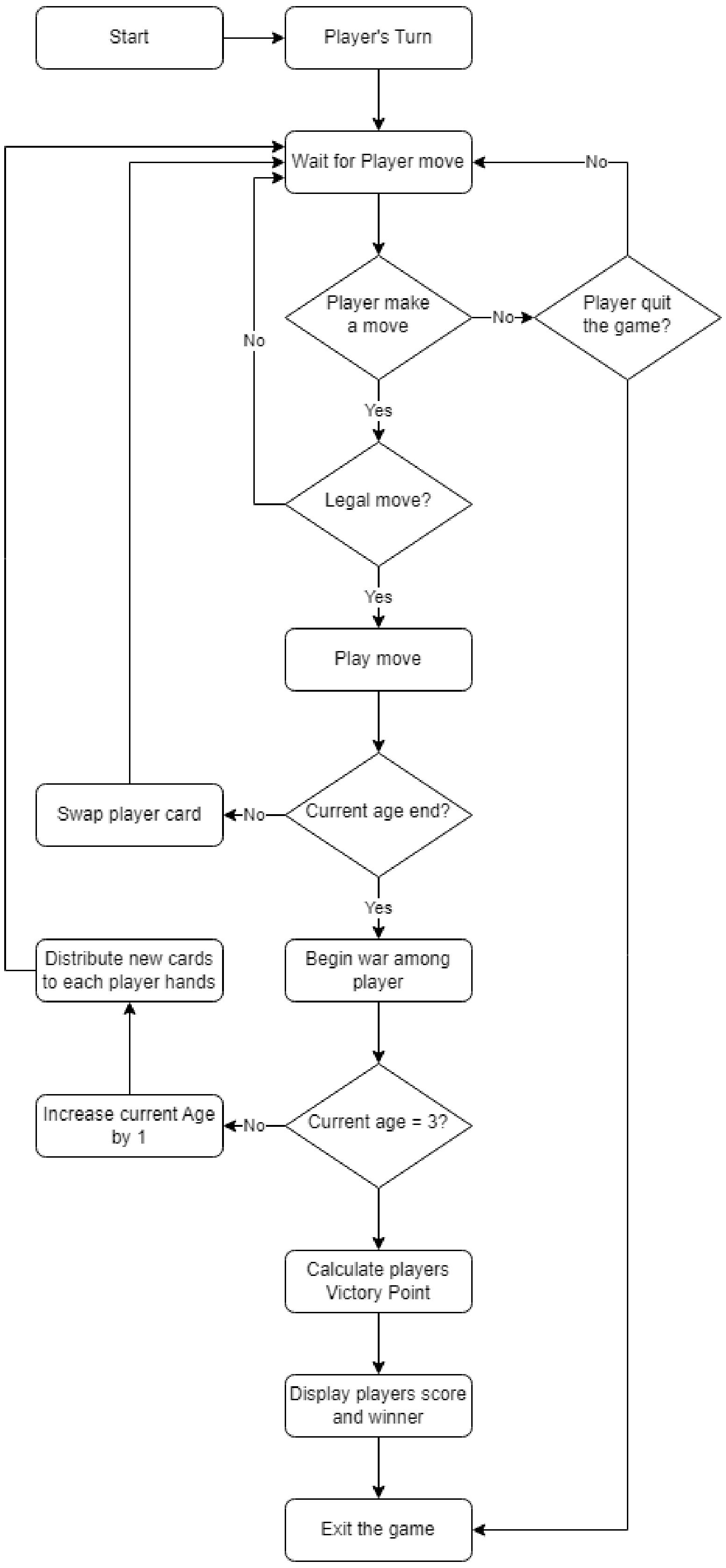
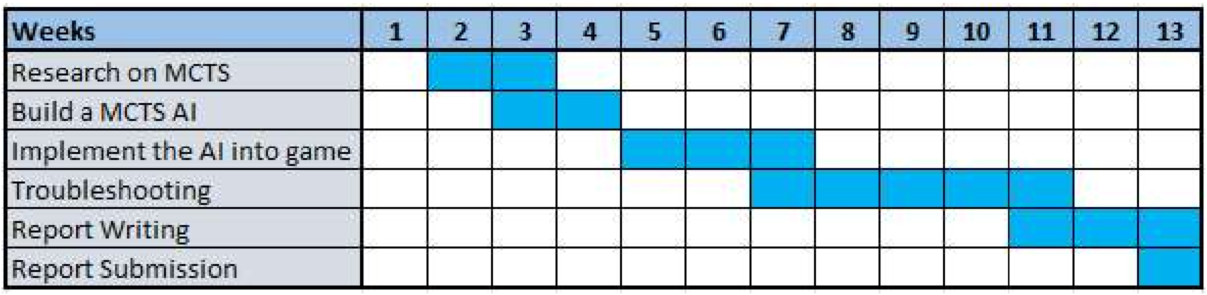


Figure 4: Flow Chart

3.4 Timeline



Weeks	1	2	3	4	5	6	7	8	9	10	11	12	13
Research on MCTS		🟦	🟦										
Build a MCTS AI			🟦	🟦									
Implement the AI into game					🟦	🟦	🟦						
Troubleshooting							🟦	🟦	🟦	🟦	🟦		
Report Writing											🟦	🟦	🟦
Report Submission													🟦

Figure 5: Timeline Table

CHAPTER 4: SYSTEM DESIGN

4.1 Game Modal Design

This section will explore the design model or interface of the game, which includes various aspects related to how the game looks and functions. The design model determines the visual and interactive components of the game, such as the layout, graphics, colors, buttons, and menus. A well-designed model or interface can enhance the player's experience, making the game more engaging, intuitive, and enjoyable. Therefore, it's crucial to consider different factors when designing the game, such as the target audience, platform, objectives, constraints, and feedback from users. By analyzing and optimizing the design model, developers can create a compelling and immersive game that meets the players' expectations and preferences.

<u>Board</u>

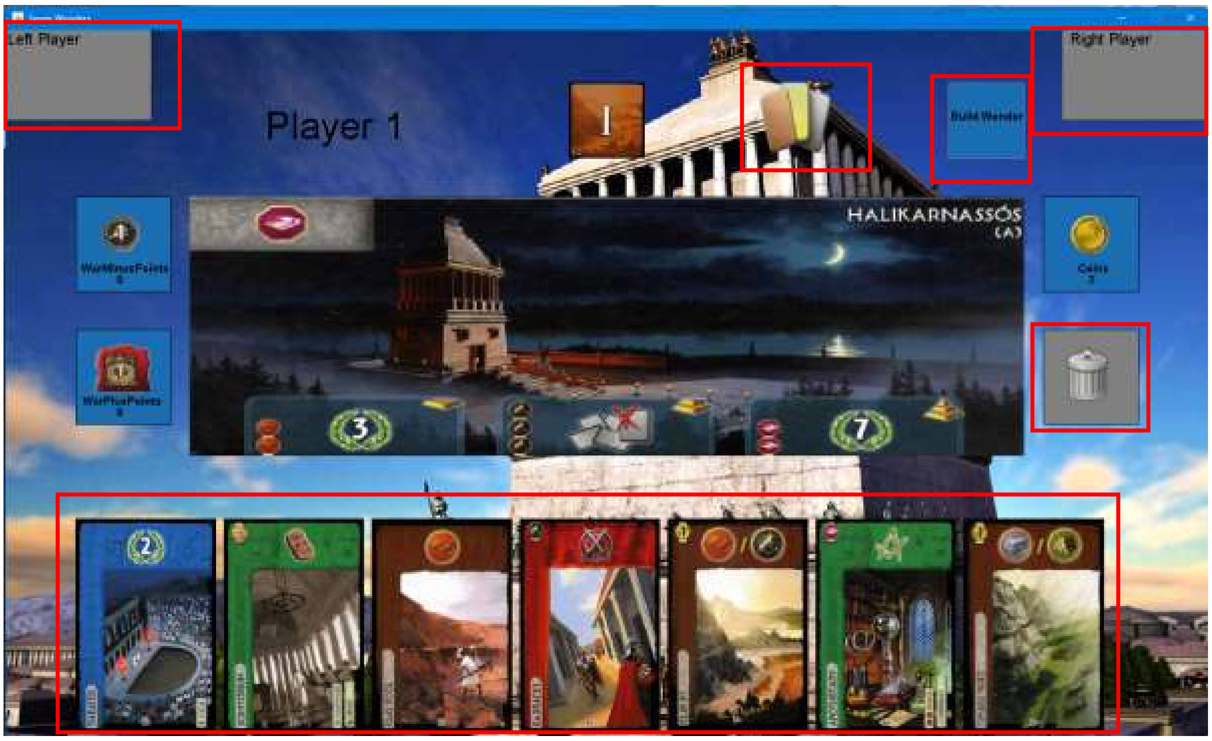


Figure 6: 7Wonder Board Interface

The image provided depicts the game board for each player, which contains all the necessary information required to play the game effectively. This information includes the resources, war points, coins, the player's hand of cards, and their wonder. At the top centre of the board, there is an icon in the shape of a card labelled as "I", which displays the current age of the game. As the game progresses, the age count will increase when an age ends. In addition, the board also features a mechanism that enables the player to view other player's boards by clicking on the appropriate box. The icon "3Card" serves as a button that, when clicked, opens up a separate window displaying the cards that have been played by the other player. This allows the current player to gain insight into the opponent's strategy and adapt their own gameplay accordingly. This function allows the player to gather information on the cards played by other players, their war points, resources, and so on.

Players have multiple options available to them, such as playing a card from their hand, building their wonder, or discarding a card to obtain three coins. These options can be accessed by toggling the "Build Wonder" or the bin figure and selecting the desired card. Overall, the board provides players with a user-friendly interface that allows them to make informed decisions and progress through the game.

Player's Card

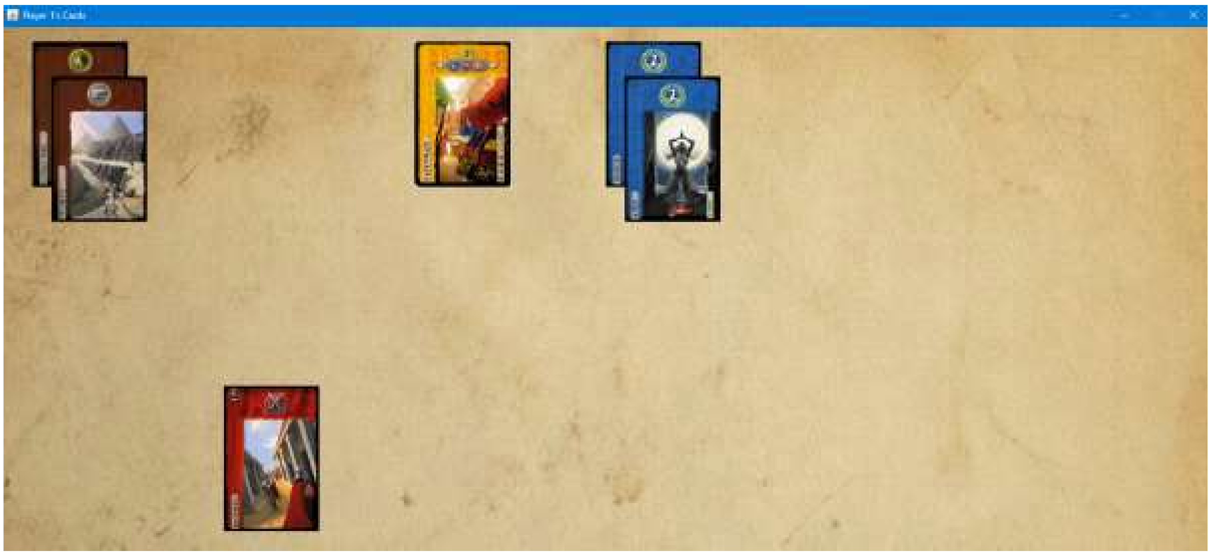


Figure 7: 7Wonder Played Card Interface

The above figure displays the cards played by a specific player, which are organized and categorized according to their respective card types, such as resource, war point, and others. This feature allows players to easily track and manage their played cards during the game, as well as to strategize their future moves based on their existing cards.

4.3 Game AI Design

MCTS Algorithm

In the standard Monte Carlo Tree Search (MCTS) algorithm, the simulation of the entire game is required to determine wins or losses, and the Upper Confidence Bound (UCB) formula is utilized to select the best move. However, in the case of 7 Wonders game, only a single player's simulation is necessary, and the Victory Points are returned when a leaf node is reached. This means that the UCB formula needs to be modified to meet the specific requirements of the game. The modified formula should consider the estimated value of the Victory Points obtained through the simulated game, as well as the exploration and exploitation values. A calculation for this modified UCB can be developed to suit the specific needs of the game.

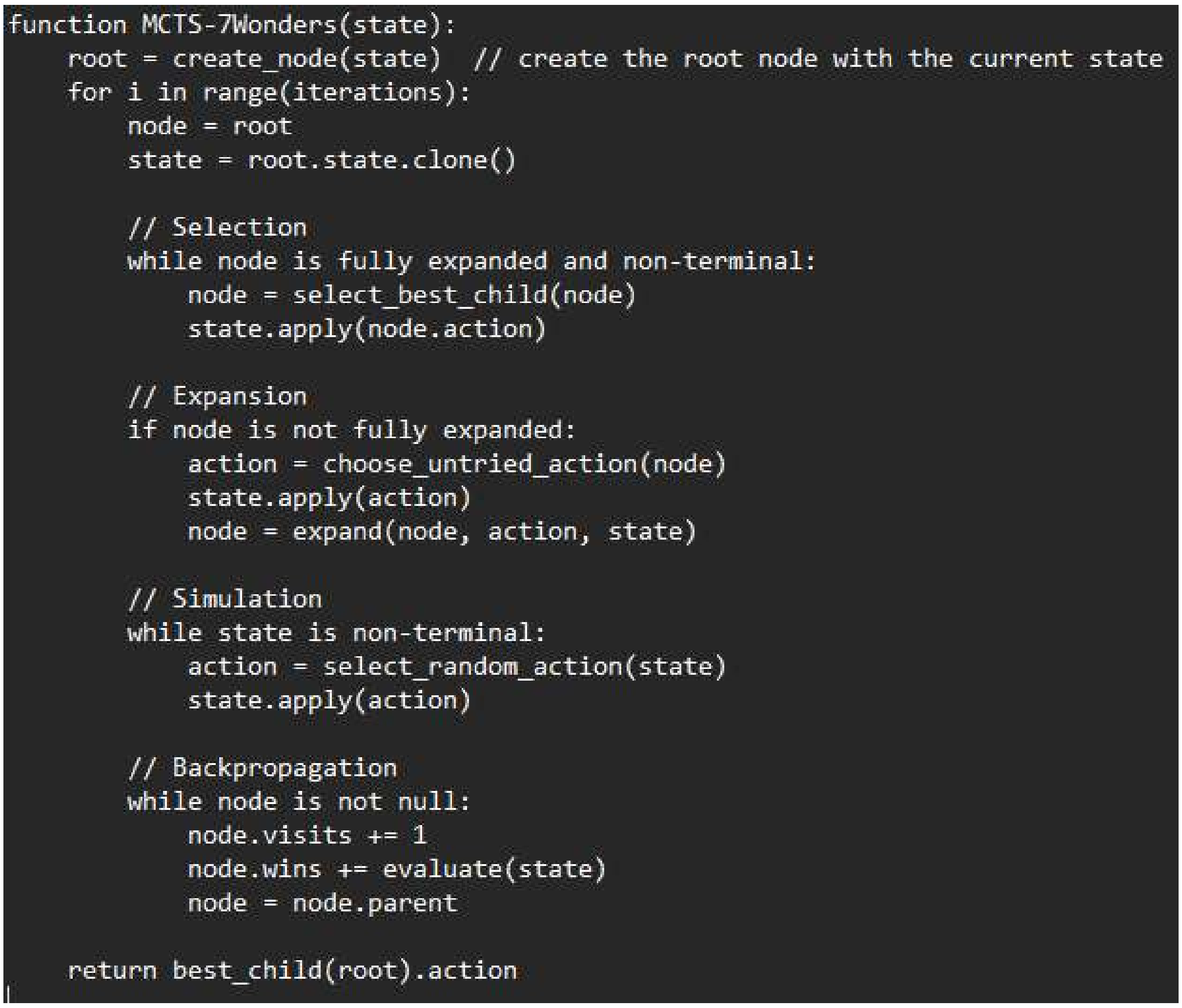
Since the traditional Upper Confidence Bound (UCB) formula is designed for selecting moves that maximize the win rate in a two-player game, it may not be directly applicable to our scenario of simulating a single player to obtain victory points. Instead of maximizing win rate, we want to maximize the expected number of victory points that can be obtained from a given move. Therefore, we modify the UCB formula to take into account the average victory points obtained from a given move, in addition to the exploration term that encourages trying less explored moves.

Hence, here's the modified formula:

$$\boldsymbol{UCB = (total\_victory\_points \,/\, num\_simulations) + exploration\_term}$$

where:

- **total_victory_points**: the total number of victory points obtained from all simulations that involved this move.
- **num_simulations**: the total number of times this move has been simulated.
- **exploration_term**: is a measure of how little this move has been explored, calculated as **c** * sqrt(ln(**total_simulations**) / **num_simulations**), where **c** is a tuning parameter that controls the amount of exploration.

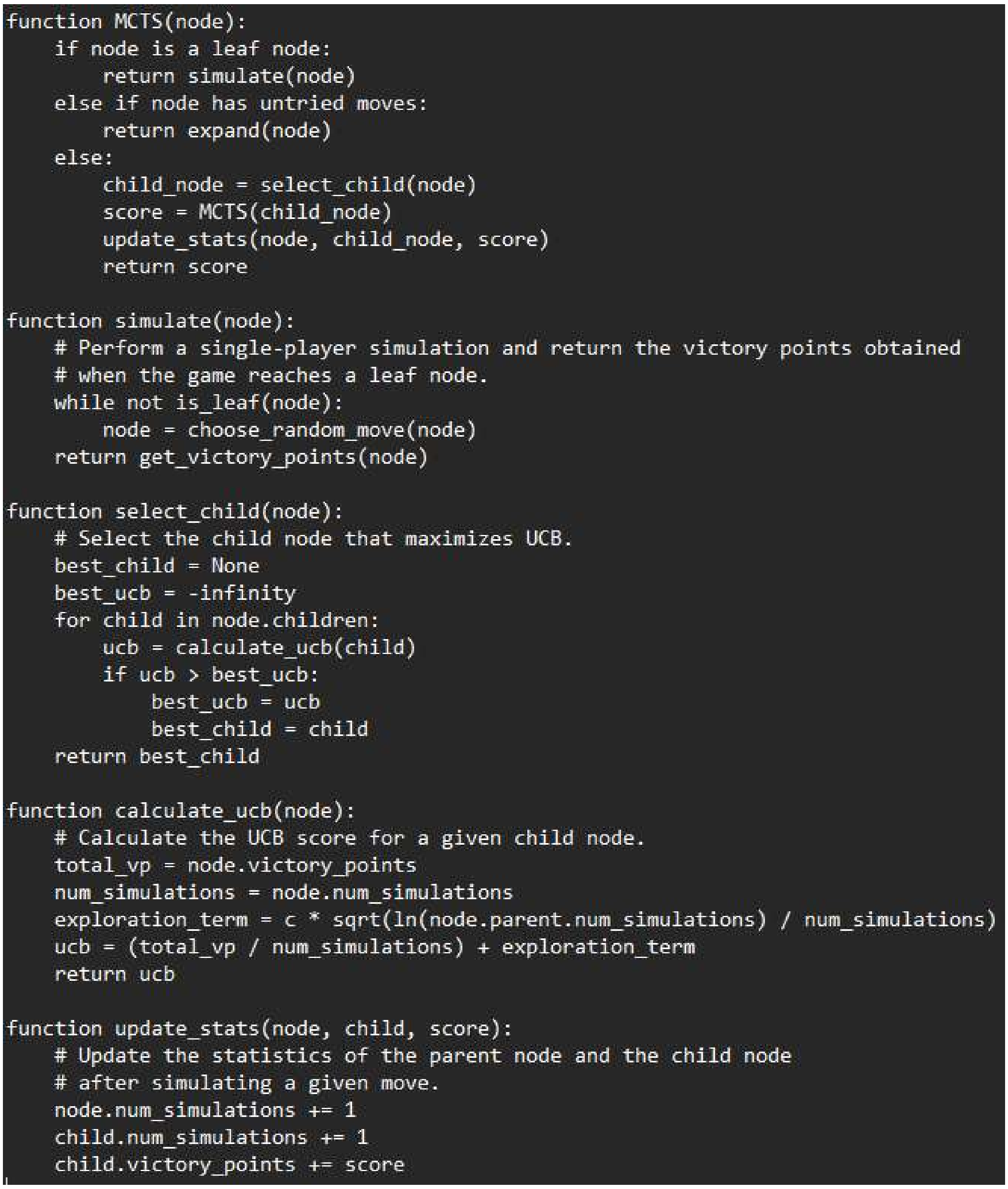


```
function MCTS-7Wonders(state):
    root = create_node(state)  // create the root node with the current state
    for i in range(iterations):
        node = root
        state = root.state.clone()

        // Selection
        while node is fully expanded and non-terminal:
            node = select_best_child(node)
            state.apply(node.action)

        // Expansion
        if node is not fully expanded:
            action = choose_untried_action(node)
            state.apply(action)
            node = expand(node, action, state)

        // Simulation
        while state is non-terminal:
            action = select_random_action(state)
            state.apply(action)

        // Backpropagation
        while node is not null:
            node.visits += 1
            node.wins += evaluate(state)
            node = node.parent

    return best_child(root).action
```

Figure 8: MCTS AI Pseudocode

The above pseudocode represents a basic implementation of the MCTS algorithm for playing 7 Wonders. It starts by creating a root node for the current state of the game and then performs a number of iterations of the MCTS algorithm.

During each iteration, the algorithm performs four steps: selection, expansion, simulation, and backpropagation. In the selection step, the algorithm selects the best child node from the current node until it reaches a non-terminal or fully expanded node. In the expansion step, the algorithm chooses an untried action from the current node and creates a new child node for that action. In the simulation step, the algorithm randomly selects actions until it reaches a terminal state. Finally, in the backpropagation step, the algorithm updates the win and visit counts for each node it traverses from the current node back up to the root. The algorithm then returns the best action from the root node's children. This implementation is a simple pseudocode and can be improved with more advanced techniques as shown below.



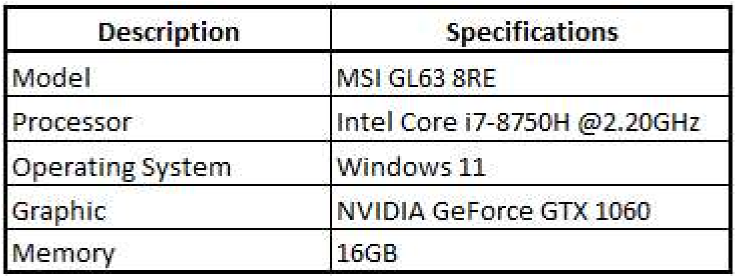
```
function MCTS(node):
    if node is a leaf node:
        return simulate(node)
    else if node has untried moves:
        return expand(node)
    else:
        child_node = select_child(node)
        score = MCTS(child_node)
        update_stats(node, child_node, score)
        return score

function simulate(node):
    # Perform a single-player simulation and return the victory points obtained
    # when the game reaches a leaf node.
    while not is_leaf(node):
        node = choose_random_move(node)
    return get_victory_points(node)

function select_child(node):
    # Select the child node that maximizes UCB.
    best_child = None
    best_ucb = -infinity
    for child in node.children:
        ucb = calculate_ucb(child)
        if ucb > best_ucb:
            best_ucb = ucb
            best_child = child
    return best_child

function calculate_ucb(node):
    # Calculate the UCB score for a given child node.
    total_vp = node.victory_points
    num_simulations = node.num_simulations
    exploration_term = c * sqrt(ln(node.parent.num_simulations) / num_simulations)
    ucb = (total_vp / num_simulations) + exploration_term
    return ucb

function update_stats(node, child, score):
    # Update the statistics of the parent node and the child node
    # after simulating a given move.
    node.num_simulations += 1
    child.num_simulations += 1
    child.victory_points += score
```

Figure 9: Modified MCTS AI Pseudocode

In this modified MCTS algorithm, the **calculate_ucb** function uses the modified UCB formula to calculate the UCB score for a given child node, and the **select_child** function selects the child node that maximizes UCB. The **update_stats** function updates the statistics of the parent node and the selected child node after a simulation is performed, by incrementing the number of simulations and adding the obtained victory points to the child node's statistics.

CHAPTER 5: SYSTEM IMPLEMENTATION

5.1 Hardware Setup



Description	Specifications
Model	MSI GL63 8RE
Processor	Intel Core i7-8750H @2.20GHz
Operating System	Windows 11
Graphic	NVIDIA GeForce GTX 1060
Memory	16GB

Figure 10: Specification of Hardware Used

5.2 Software Setup

Eclipse 2023

Figure 11: Eclipse Logo/Icon

Eclipse is an integrated development environment (IDE) that is commonly used to develop software applications in Java, though it can also support other programming languages. It provides a user-friendly interface and a suite of tools that can aid in software development, such as a code editor, a debugger, and a build automation tool. Eclipse also supports plugins, which can be used to extend its functionality and support additional programming languages, frameworks, and tools.

Java Development Kit 17

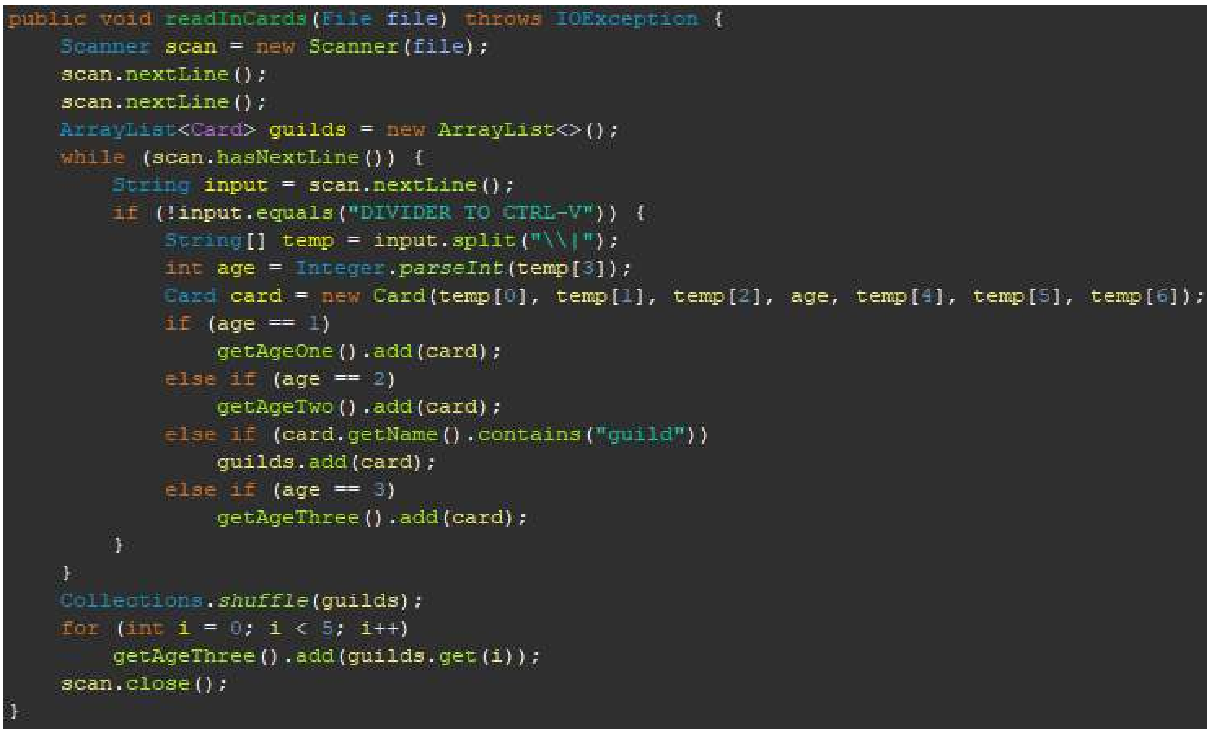
Figure 12: Java Development Kit 17 Icon/Logo

The Java Development Kit (JDK) is a software development kit (SDK) provided by Oracle that includes a set of tools needed to develop Java applications. It includes a Java compiler, which converts Java code into bytecode that can be executed by a Java Virtual Machine (JVM); the Java Runtime Environment (JRE), which provides the environment necessary to run Java applications; and other tools that can aid in Java development, such as a debugger, documentation generator, and JavaFX tools.

5.3 Game Logic

Game Initialization

At the beginning of the game, the program will read a text file containing information about the cards and then separate them into three different ages as shown in figure below. Afterward, the cards will be added to an ArrayList, which is a type of dynamic array in Java that can grow or shrink in size as needed. This process will allow the game to access and use the card information as needed during gameplay.

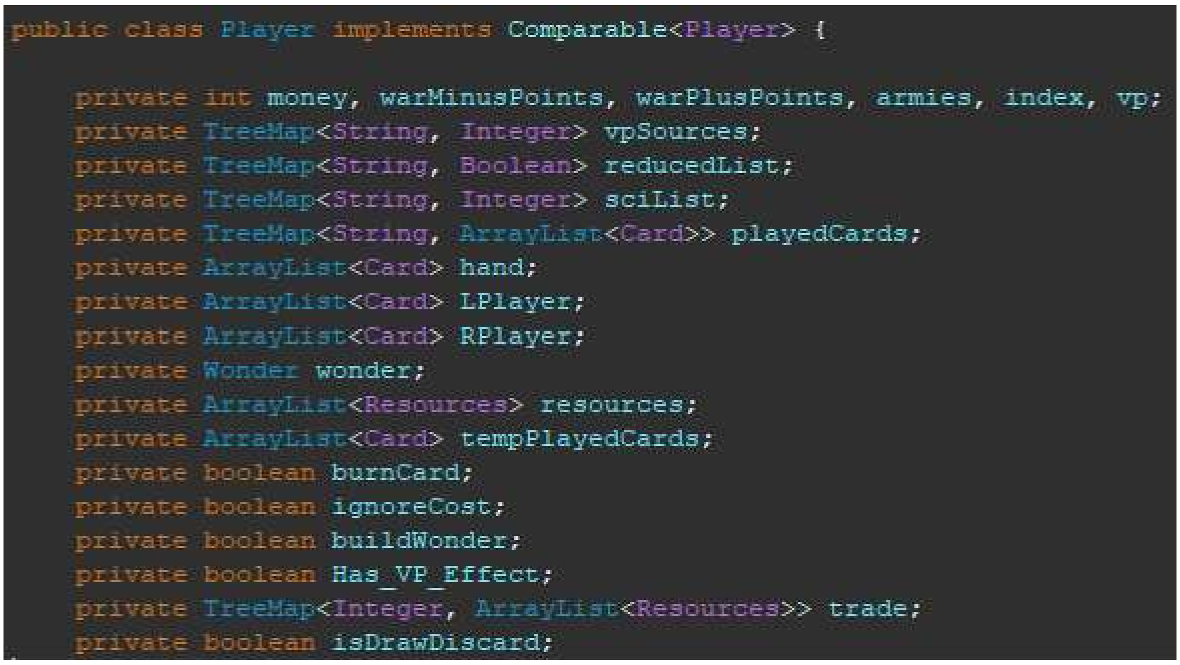


```java
public void readInCards(File file) throws IOException {
    Scanner scan = new Scanner(file);
    scan.nextLine();
    scan.nextLine();
    ArrayList<Card> guilds = new ArrayList<>();
    while (scan.hasNextLine()) {
        String input = scan.nextLine();
        if (!input.equals("DIVIDER TO CTRL-V")) {
            String[] temp = input.split("\\|");
            int age = Integer.parseInt(temp[3]);
            Card card = new Card(temp[0], temp[1], temp[2], age, temp[4], temp[5], temp[6]);
            if (age == 1)
                getAgeOne().add(card);
            else if (age == 2)
                getAgeTwo().add(card);
            else if (card.getName().contains("guild"))
                guilds.add(card);
            else if (age == 3)
                getAgeThree().add(card);
        }
    }
    Collections.shuffle(guilds);
    for (int i = 0; i < 5; i++)
        getAgeThree().add(guilds.get(i));
    scan.close();
}
```

Figure 13: Card Reader Method Code

During the initialization of the game, the system also generates 7 unique wonders along with their associated resources and effects. This information is not read from a text file because there are only 7 wonders in the game, and therefore it does not require a large amount of code to be implemented.

After the system has read the cards and created the 7 unique wonders, players will be assigned a set of these cards and wonders. At this point, players will also be given all the necessary attributes for the game, such as victory points, coins, military strength, and so on as shown below.



```
public class Player implements Comparable<Player> {

    private int money, warMinusPoints, warPlusPoints, armies, index, vp;
    private TreeMap<String, Integer> vpSources;
    private TreeMap<String, Boolean> reducedList;
    private TreeMap<String, Integer> sciList;
    private TreeMap<String, ArrayList<Card>> playedCards;
    private ArrayList<Card> hand;
    private ArrayList<Card> LPlayer;
    private ArrayList<Card> RPlayer;
    private Wonder wonder;
    private ArrayList<Resources> resources;
    private ArrayList<Card> tempPlayedCards;
    private boolean burnCard;
    private boolean ignoreCost;
    private boolean buildWonder;
    private boolean Has_VP_Effect;
    private TreeMap<Integer, ArrayList<Resources>> trade;
    private boolean isDrawDiscard;
```

Figure 14: Player's Attributes Code

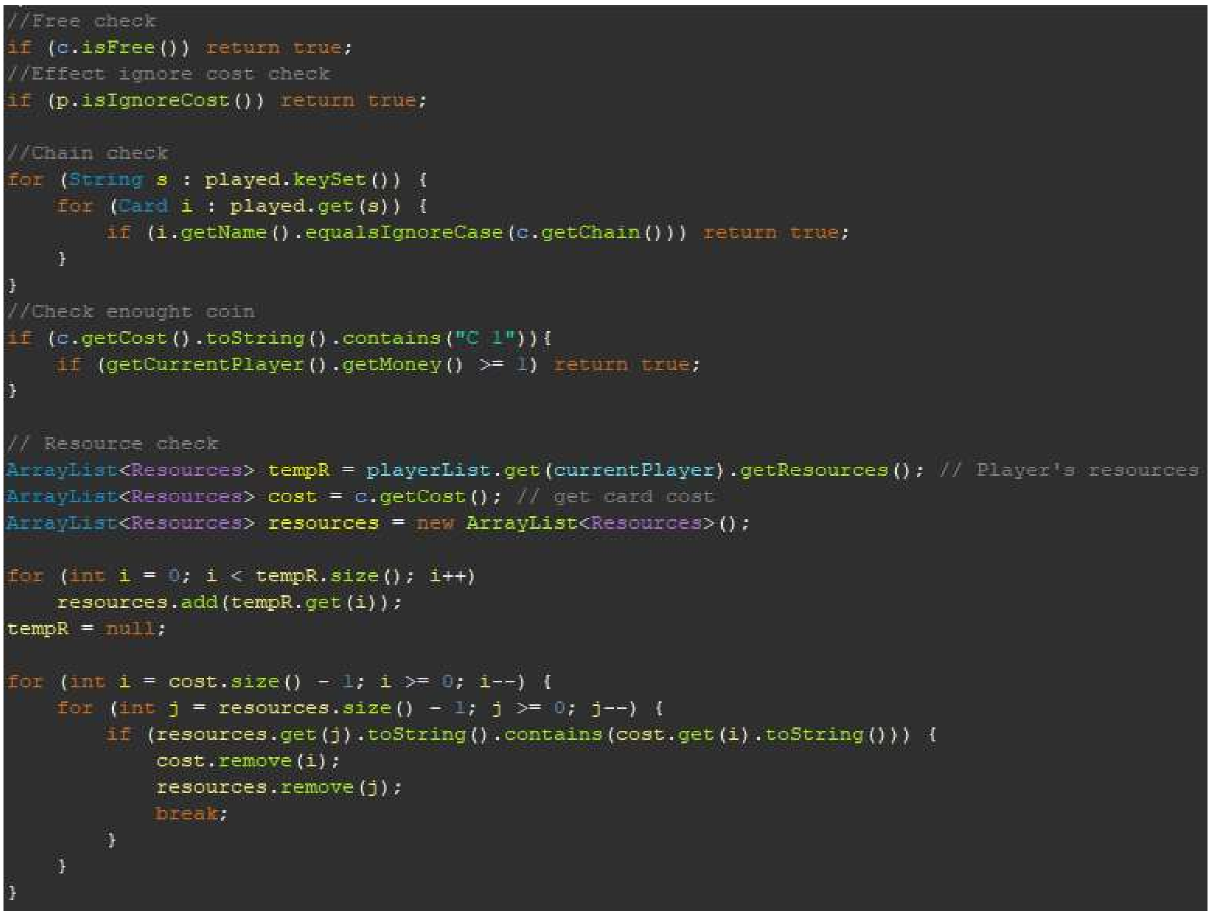
<u>Legal Move Check</u>

This section of the code is responsible for checking whether a player-selected card is playable or not. The code performs a series of checks to determine the card's playability. Firstly, it checks whether the card is free to play or not. Additionally, it checks whether any of the card effects ignore the card's cost.

The system also verifies whether the selected card is a chain card, which requires a previously played card, and checks whether the player has the necessary card to play it.

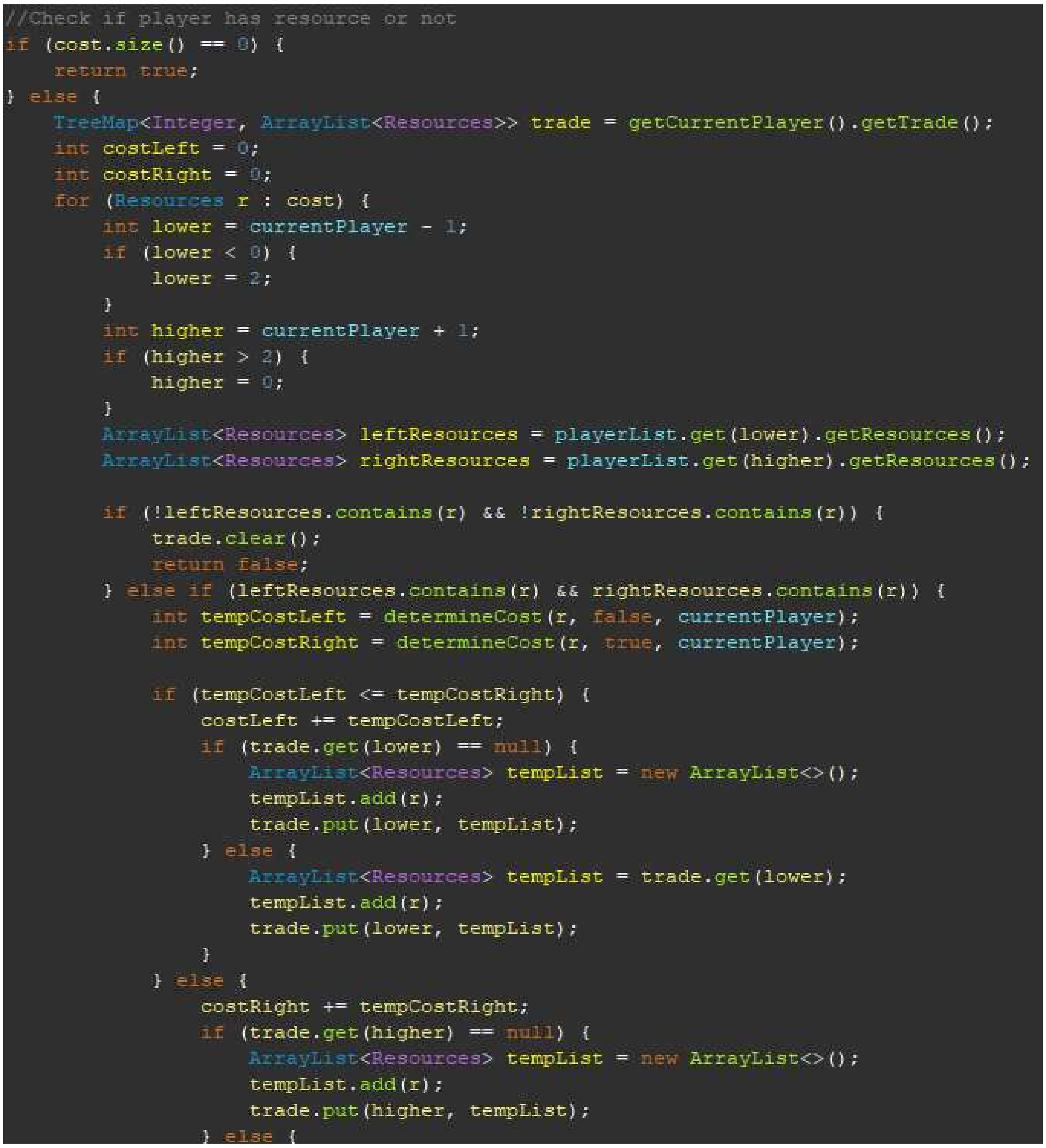
If the card has a cost in coins, the code checks whether the player has enough coins to play it. On the other hand, if the card has a resource cost, the system checks whether the player has the required resources. If not, it then checks whether the left or right adjacent player has the necessary resources, which the current player can use to play the card.

In the end, the code returns false if the card is not playable, and true if it meets one of the above requirements. These checks ensure that players cannot play cards that they are not eligible to play, according to the game's rules.



```
//Free check
if (c.isFree()) return true;
//Effect ignore cost check
if (p.isIgnoreCost()) return true;

//Chain check
for (String s : played.keySet()) {
    for (Card i : played.get(s)) {
        if (i.getName().equalsIgnoreCase(c.getChain())) return true;
    }
}
//Check enought coin
if (c.getCost().toString().contains("C 1")){
    if (getCurrentPlayer().getMoney() >= 1) return true;
}

// Resource check
ArrayList<Resources> tempR = playerList.get(currentPlayer).getResources(); // Player's resources
ArrayList<Resources> cost = c.getCost(); // get card cost
ArrayList<Resources> resources = new ArrayList<Resources>();

for (int i = 0; i < tempR.size(); i++)
    resources.add(tempR.get(i));
tempR = null;

for (int i = cost.size() - 1; i >= 0; i--) {
    for (int j = resources.size() - 1; j >= 0; j--) {
        if (resources.get(j).toString().contains(cost.get(i).toString())) {
            cost.remove(i);
            resources.remove(j);
            break;
        }
    }
}
```

Figure 15: Check Legal Move Method Code ~ 1

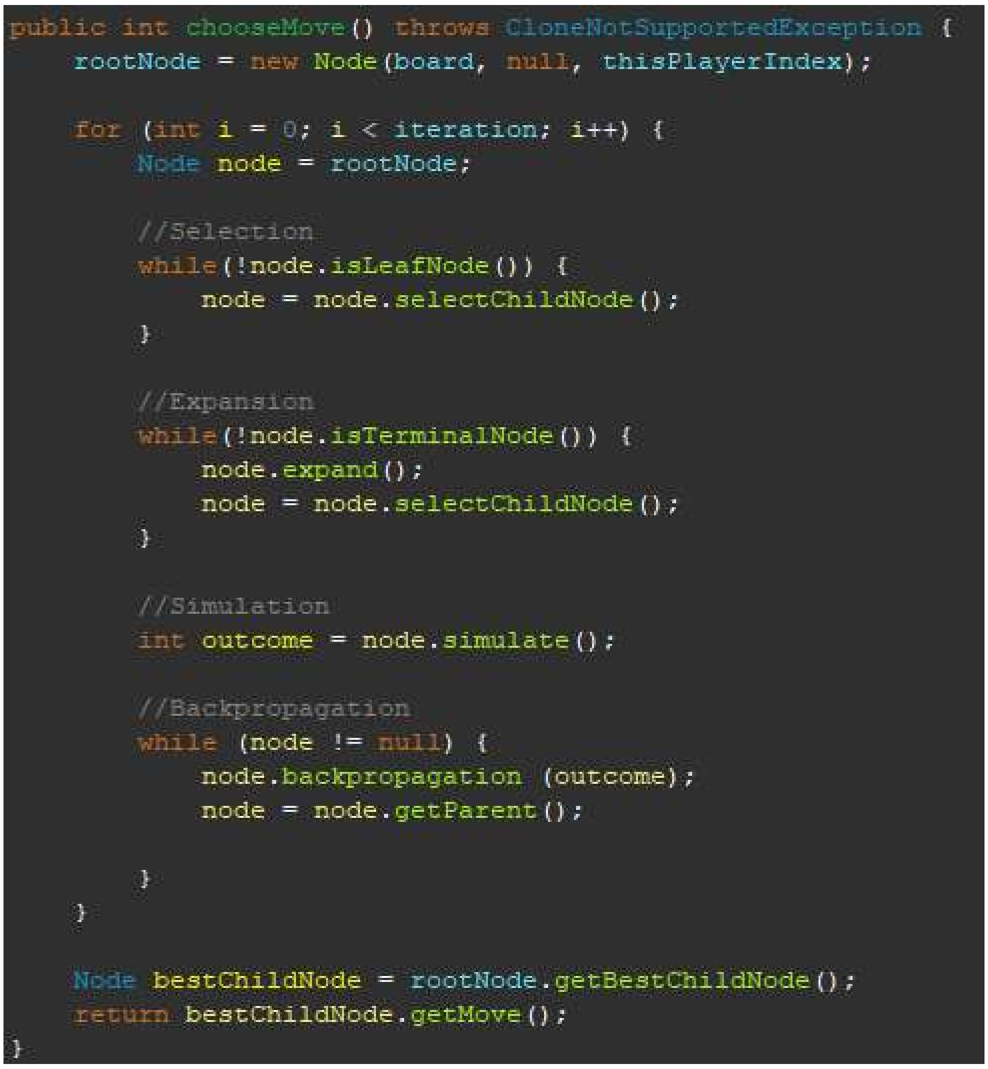


```
//Check if player has resource or not
if (cost.size() == 0) {
    return true;
} else {
    TreeMap<Integer, ArrayList<Resources>> trade = getCurrentPlayer().getTrade();
    int costLeft = 0;
    int costRight = 0;
    for (Resources r : cost) {
        int lower = currentPlayer - 1;
        if (lower < 0) {
            lower = 2;
        }
        int higher = currentPlayer + 1;
        if (higher > 2) {
            higher = 0;
        }
        ArrayList<Resources> leftResources = playerList.get(lower).getResources();
        ArrayList<Resources> rightResources = playerList.get(higher).getResources();

        if (!leftResources.contains(r) && !rightResources.contains(r)) {
            trade.clear();
            return false;
        } else if (leftResources.contains(r) && rightResources.contains(r)) {
            int tempCostLeft = determineCost(r, false, currentPlayer);
            int tempCostRight = determineCost(r, true, currentPlayer);

            if (tempCostLeft <= tempCostRight) {
                costLeft += tempCostLeft;
                if (trade.get(lower) == null) {
                    ArrayList<Resources> tempList = new ArrayList<>();
                    tempList.add(r);
                    trade.put(lower, tempList);
                } else {
                    ArrayList<Resources> tempList = trade.get(lower);
                    tempList.add(r);
                    trade.put(lower, tempList);
                }
            } else {
                costRight += tempCostRight;
                if (trade.get(higher) == null) {
                    ArrayList<Resources> tempList = new ArrayList<>();
                    tempList.add(r);
                    trade.put(higher, tempList);
                } else {
```

Figure 16: Check Legal Move Method Code ~ 2

MCTS AI Simulation

In this implementation, MCTS is the class that contains the main algorithm for the Monte Carlo Tree Search, while Node represents a single node in the tree.



```
public int chooseMove() throws CloneNotSupportedException {
    rootNode = new Node(board, null, thisPlayerIndex);

    for (int i = 0; i < iteration; i++) {
        Node node = rootNode;

        //Selection
        while(!node.isLeafNode()) {
            node = node.selectChildNode();
        }

        //Expansion
        while(!node.isTerminalNode()) {
            node.expand();
            node = node.selectChildNode();
        }

        //Simulation
        int outcome = node.simulate();

        //Backpropagation
        while (node != null) {
            node.backpropagation (outcome);
            node = node.getParent();

        }
    }

    Node bestChildNode = rootNode.getBestChildNode();
    return bestChildNode.getMove();
}
```

Figure 17: MCTS Class Code

The MCTS class has three methods:

- **chooseMove** is the main loop of the algorithm, which repeatedly selects child nodes until it reaches a leaf node, which is then expanded and simulated.
- **selectChildNode** is to select one of the child nodes to simulate next, using the Upper Confidence Bound (UCB) algorithm.
- **expand** add one or more child nodes to the selected node, representing each of the untried moves.
- **simulate** will play the leaf node selected while doing expansion and return the outcome for backpropagation.
- **backpropagation** updates the statistics for a node and all its ancestors, after a given move has been simulated.

5.4 System Implementation Challenges

One of the major challenges I encountered while working on the project was related to copying objects and simulating them without changing the original object. It took me around 3-5 days to troubleshoot the issue and find a solution to achieve independence by performing deep copies. Another significant challenge I faced was related to the mechanics of the game itself. I needed to implement the effects of each wonder and card into the system, which required writing a specific algorithm for each special effect. To tackle this, I wrote all the card information in a txt file so that I could retrieve it easily using a formula and incorporate the effects of each card into the program.

CHAPTER 6: CONCLUSION

The successful implementation of the MCTS algorithm in the 7Wonder game has been achieved, but there are some limitations to the algorithm. For instance, the AI currently lacks the ability to prioritize building wonders moves over playing card moves, which could affect its overall strategy. Despite this limitation, the AI has managed to win in many instances due to its cognitive complexity and calculative approach, making it challenging for humans to beat.

To improve the AI's performance further, there are suggestions for refactoring the legal move algorithm, as well as recording each hand it receives, which would improve the AI's ability to predict its deck. In 7Wonder, decks are swapped between players in a clockwise or counterclockwise direction, which makes it essential to record the sequence of hands received. Additionally, the AI could record the moves made by other players to help reduce the number of possible cards based on the information available.

The suggested improvements could increase the speed of computation and allow the AI to search deeper, which would lead to a more accurate result. Overall, the successful implementation of the MCTS algorithm in the 7Wonder game has shown that AI can provide a challenging and engaging opponent, even when no human players are available.

BIBLIOGRAPHY

[1] I. Szita, G. Chaslot and P. Spronck, “Monte-Carlo Tree Search in Settlers of Catan,” in *Advances in Computer Games*, Berlin, 2010.

[2] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis and S. Colton, “A Survey of Monte Carlo Tree Search Methods,” *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 4, pp. 1-43, 2012.

[3] G. Zuo and C. Wu, “A heuristic Monte Carlo tree search method for surakarta chess,” in *2016 Chinese Control and Decision Conference (CCDC)*, 2016.

[4] J. S. B. Choe and J.-K. Kim, “Enhancing Monte Carlo Tree Search for Playing Hearthstone,” in *2019 IEEE Conference on Games (CoG)*, 2019.

[5] E. Galván and G. Simpson, “On the Evolution of the MCTS Upper Confidence Bounds for Trees by Means of Evolutionary Algorithms in the Game of Carcassonne,” in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021.

[6] A. Dockhorn, J. Hurtado-Grueso, D. Jeurissen, L. Xu and D. Perez-Liebana, “Game State and Action Abstracting Monte Carlo Tree Search for General Strategy Game-Playing,” in *2021 IEEE Conference on Games (CoG)*, 2021.

[7] R. Bettker, P. Minini, G. Pereira and J. V. C. Assunção, “Towards playing AIs for 7 Wonders: main patterns and strategies for 3-player games,” in *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2021.

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 4**
Student Name & ID: Lee Jian Zhen 19ACB02281	
Supervisor: Dr. Lee Heng Yew	
Project Title: Application of Monte Carlo Tree Search Algorithm in 7 Wonders	

1. WORK DONE
[Please write the details of the work done in the last fortnight.]

Study & Research of MCTS Algorithm

2. WORK TO BE DONE

Create a MCTS Algorithm

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

Learned the data structure of MCTS.

Heng Yew

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 6**
Student Name & ID: Lee Jian Zhen 19ACB02281	
Supervisor: Dr. Lee Heng Yew	
Project Title: Application of Monte Carlo Tree Search Algorithm in 7 Wonders	

1. WORK DONE [Please write the details of the work done in the last fortnight.] Created a MCTS algorithm.
2. WORK TO BE DONE Implement the MCTS algorithm into 7Wonder.
3. PROBLEMS ENCOUNTERED -
4. SELF EVALUATION OF THE PROGRESS Better known of the logic of MCTS algorithm.

Heng Yew

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 8**
Student Name & ID: Lee Jian Zhen 19ACB02281	
Supervisor: Dr. Lee Heng Yew	
Project Title: Application of Monte Carlo Tree Search Algorithm in 7 Wonders	

1. WORK DONE
[Please write the details of the work done in the last fortnight.]

Implement the MCTS algorithm and troubleshooting the errors.

2. WORK TO BE DONE

Successfully implement the MCTS algorithm.
Find a way to modify the algorithm to best fit my 7Wonder program.

3. PROBLEMS ENCOUNTERED

I need to make the MCTS algorithm fit the 7Wonder game.
My 7Wonder code was not good enough and it make some trouble while I am implementing the MCTS algorithm.

4. SELF EVALUATION OF THE PROGRESS

Study the error and solve it.

Heng Yew

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 10**
Student Name & ID: Lee Jian Zhen 19ACB02281	
Supervisor: Dr. Lee Heng Yew	
Project Title: Application of Monte Carlo Tree Search Algorithm in 7 Wonders	

1. WORK DONE
[Please write the details of the work done in the last fortnight.]

Done implement the MCTS algorithm into 7Wonder.

2. WORK TO BE DONE

Write the report.

3. PROBLEMS ENCOUNTERED

Frequently occur errors when implement the MCTS algorithm.

4. SELF EVALUATION OF THE PROGRESS

Study the error and solve it

Heng Yew
Supervisor's signature

[illegible]
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: 12**
Student Name & ID: Lee Jian Zhen 19ACB02281	
Supervisor: Dr. Lee Heng Yew	
Project Title: Application of Monte Carlo Tree Search Algorithm in 7 Wonders	

1. WORK DONE
[Please write the details of the work done in the last fortnight.]

Completed the report.

2. WORK TO BE DONE

Review my project and report for better improvement.
Make my program playable without installing eclipse IDE.

3. PROBLEMS ENCOUNTERED

-

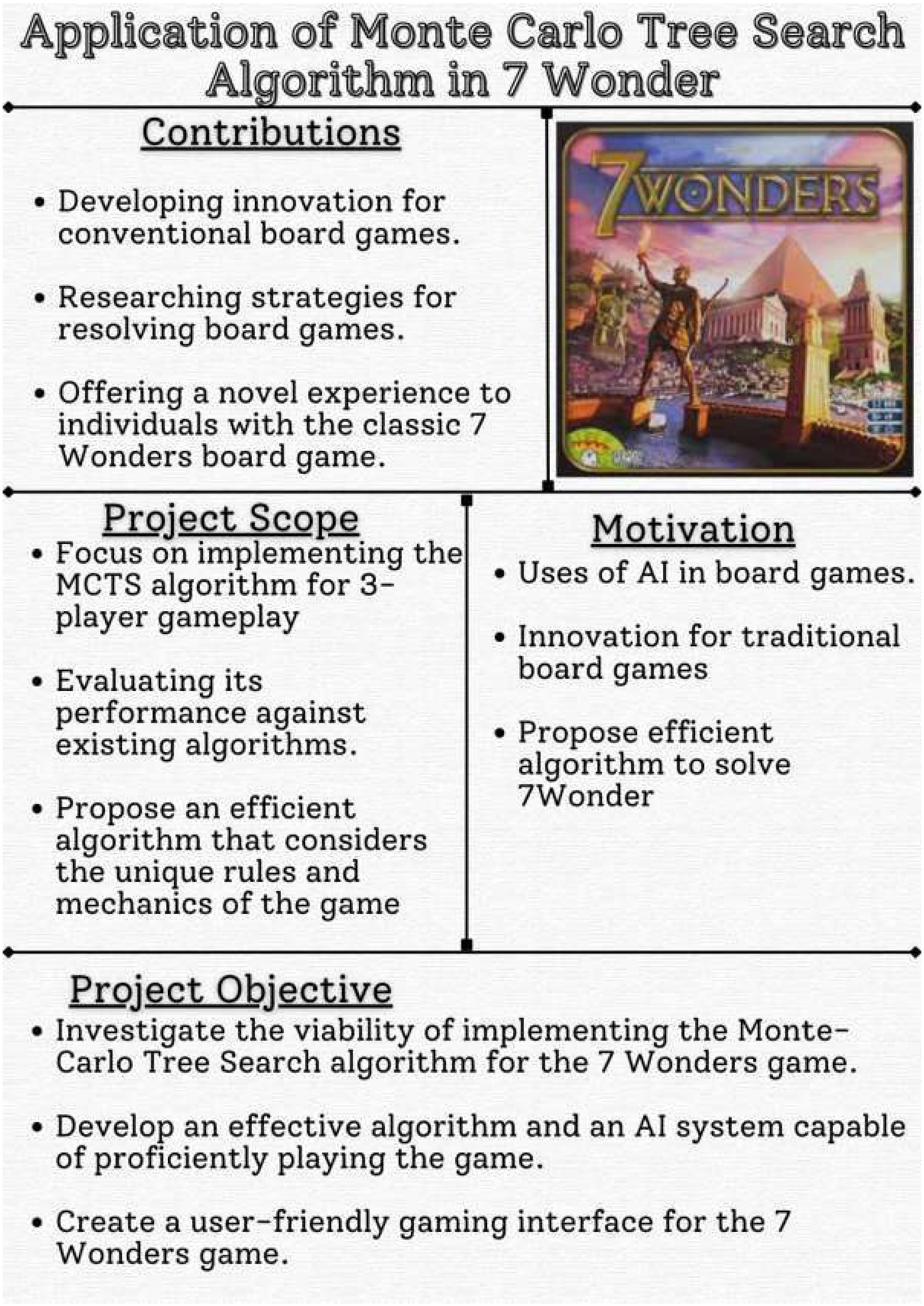
4. SELF EVALUATION OF THE PROGRESS

Refinement of my project.

Heng Yew
Supervisor's signature

[illegible]
Student's signature

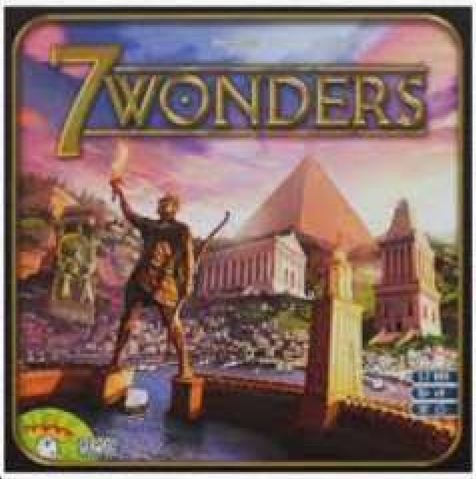
POSTER



Application of Monte Carlo Tree Search Algorithm in 7 Wonder

Contributions

- Developing innovation for conventional board games.
- Researching strategies for resolving board games.
- Offering a novel experience to individuals with the classic 7 Wonders board game.



Project Scope

- Focus on implementing the MCTS algorithm for 3-player gameplay
- Evaluating its performance against existing algorithms.
- Propose an efficient algorithm that considers the unique rules and mechanics of the game

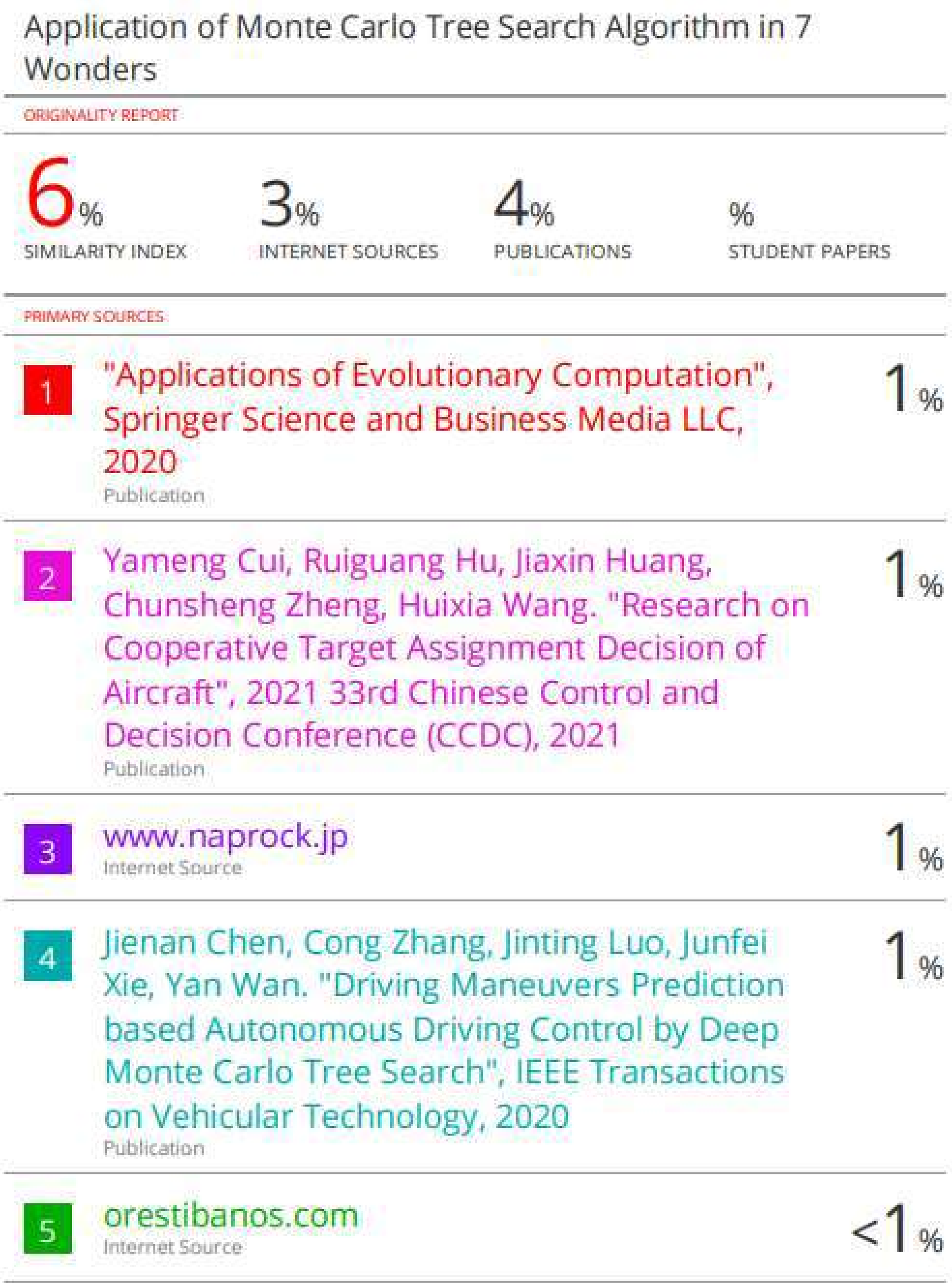
Motivation

- Uses of AI in board games.
- Innovation for traditional board games
- Propose efficient algorithm to solve 7Wonder

Project Objective

- Investigate the viability of implementing the Monte-Carlo Tree Search algorithm for the 7 Wonders game.
- Develop an effective algorithm and an AI system capable of proficiently playing the game.
- Create a user-friendly gaming interface for the 7 Wonders game.

PLAGIARISM CHECK RESULT



Application of Monte Carlo Tree Search Algorithm in 7 Wonders

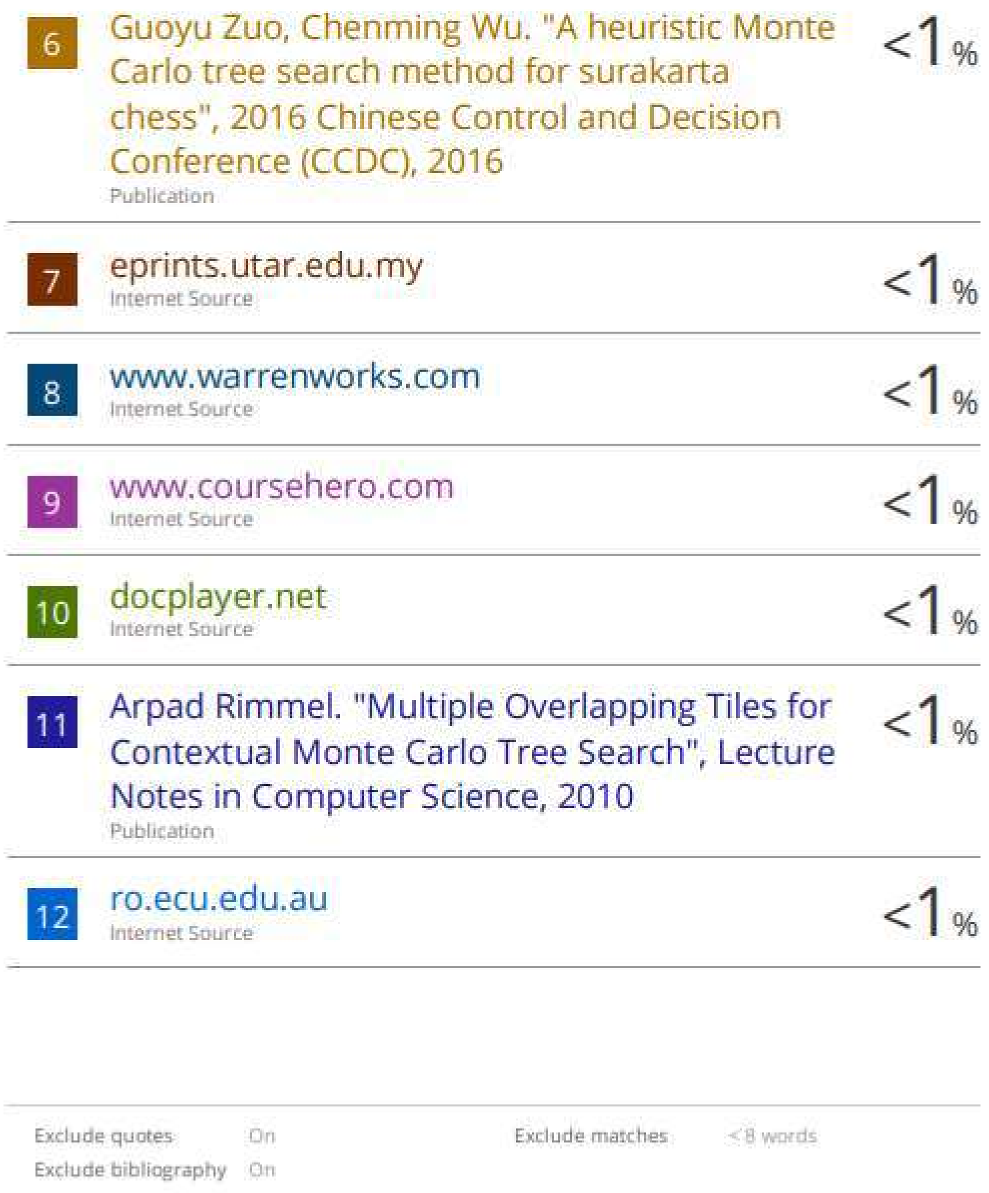
ORIGINALITY REPORT

6%	3%	4%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	"Applications of Evolutionary Computation", Springer Science and Business Media LLC, 2020 Publication	1%
2	Yameng Cui, Ruiguang Hu, Jiaxin Huang, Chunsheng Zheng, Huixia Wang. "Research on Cooperative Target Assignment Decision of Aircraft", 2021 33rd Chinese Control and Decision Conference (CCDC), 2021 Publication	1%
3	www.naprock.jp Internet Source	1%
4	Jienan Chen, Cong Zhang, Jinting Luo, Junfei Xie, Yan Wan. "Driving Maneuvers Prediction based Autonomous Driving Control by Deep Monte Carlo Tree Search", IEEE Transactions on Vehicular Technology, 2020 Publication	1%
5	orestibanos.com Internet Source	<1%

PLAGIARISM CHECK RESULT



6	Guoyu Zuo, Chenming Wu. "A heuristic Monte Carlo tree search method for surakarta chess", 2016 Chinese Control and Decision Conference (CCDC), 2016 Publication	<1 %
7	eprints.utar.edu.my Internet Source	<1 %
8	www.warrenworks.com Internet Source	<1 %
9	www.coursehero.com Internet Source	<1 %
10	docplayer.net Internet Source	<1 %
11	Arpad Rimmel. "Multiple Overlapping Tiles for Contextual Monte Carlo Tree Search", Lecture Notes in Computer Science, 2010 Publication	<1 %
12	ro.ecu.edu.au Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	LEE JIAN ZHEN
ID Number(s)	19ACB02281
Programme / Course	Bachelor of Computer Science (HONOURS)
Title of Final Year Project	Application of Monte Carlo Tree Search Algorithm in 7 Wonders

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 6 **%** **Similarity by source** Internet Sources: 3 % Publications: 4 % Student Papers: 0 %	Meet requirement
Number of individual sources listed of more than 3% similarity: 0	Meet requirement

Parameters of originality required and limits approved by UTAR are as Follows:
- **(i) Overall similarity index is 20% and below, and**
- **(ii) Matching of individual sources listed must be less than 3% each, and**
- **(iii) Matching texts in continuous block must not exceed 8 words**

Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Heng Yew

Signature of Supervisor

Name: **Lee Heng Yew**

Date: **28/04/2023**

Signature of Co-Supervisor

Name:

Date:

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	19ACB02281
Student Name	LEE JIAN ZHEN
Supervisor Name	Mr. LEE HENG YEW

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	Front Plastic Cover (for hardcopy)
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
	List of Tables (if applicable)
	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

[illegible]
____
(Signature of Student)
Date: 27/04/2023