Othello

Via Artificial Intelligence Techniques

By

Wajahat Ali

Supervised By

Dr. Onaiza Maqbool

Department of Computer Science,

Quaid-i-Azam University, Islamabad, Pakistan.

Session (2015 – 2019)

ACKNOWLEDGEMENT

Firstly, I would like to express my gratitude to Allah Almighty, whose blessings were showered upon me during my time at Quaid-i-Azam University in the form of achievements, and learning. In addition, I am obligated to several people for their sustenance and generousity. This gratitude of mine is especially contributed to my supervisor, Dr. Onaiza Maqbool, for guiding me throughout the process. I couldn't have imagined to garner for having a better advisor and mentor for my project. I am also grateful towards all of my teachers: Dr. Khalid Saleem, Dr. Mudassir Azam Sindhu, Dr. Akmal Khan Khattak, Dr. Rabeeh Ayaz Abbasi, Ms. Memoona Afsheen Malik, Ms. Ifrah Farrukh Khan; for all their teachings and efforts throughout these 4 years, and my beginning career. I want to commemorate my work to my beloved parents, brothers, sister-in-law for all their efforts they had to put up with me during my 4 years here at this university. The people who inspired me in every step along the way, are my my friends: Mohammad Raza Najafi, Dawood Ali, Mohammad Awais, Ali Yar Khan, Muhammad Farwa Noor, Usama Zafar Mughal, Daniyal Shiekh, Hamza Munir Gondal, Meer Hamza Butt, Ibtehsam-ul-Haq, Asad Aslam, Saifullah-uz-Zaman, Haroon Akber (Senior), Huzaifa Sheikh (Senior), Hafiz Muhammad Aon, Hafiz Sikandar Waheed (Senior), Syed Muhammad Shah Tahir (Senior), Danish Ali Cheema (Senior), Muhammad Usman Ali (Senior), Danish Alam (Senior), Wasi Haider, Muhammad Bin Shehzad, Syed Abdul Hannan, Kashif Rasool Wattoo, Aqsa Ameen, Ifza Gul, Wardah Khan Alizai, Urooj Wahid, Maryam Saeed, Nida Waheed, Hafiza Syeda Kainat Gharsheen, Tahira Niazi, Ali Raza Khan, Adil Riaz Gondal and last, but not the least, Hussain Safdar Gondal.

ABSTRACT

The product is a 3D board game project based on the realistic and tangible version played in real life, but with some extensions towards exclusive features. The basic concept of the game includes an 8x8 board where the player with the black discs goes first and the discs are to be placed adjacent or next to their oppositely colored neighboring discs in doing so, the player with most discs on the board wins.

Additionally, the product is equipped with features that includes the option of selecting to play against another player or against the computer itself. He/she can have a choice of selecting different configurations of the disc's before commencing the game, this is just to shift from the traditional black and white discs. It will have 3 different difficulty levels i.e. Easy, Medium and Hard. Player can unlock new custom boards and discs by earning and spending diamonds (incentive and appreciation of playing the game more often) after beating another player or the computer. The player when playing against the player can add comments that would get deleted or fade away shortly. The board game environment is three dimensional giving a realistic simulation.

The product is designed and implemented using the Unity Game Engine's version 2019.1.14f1 (64-bit). The language used for the implementation is C#.

Contents

List of figures

List of tables

Chapter 1: SOFTWARE PROJECT MANAGAMENT PLAN (SPMP)

1. Software Project Management Plan

1.1 Introduction

On March 15th, 2016, the Go-playing computer program known as AlphaGo took the world by surprise as it won its fourth and final game against international Go champion Lee Sedol [1]. To see whether AlphaGo's recent achievement can be recreated for a different board game, this project aims to create a game-playing artificial intelligence program that uses Artificial Intelligence algorithms. The program plays the much simpler board game Othello using a combination of tree search algorithms. This chapter explains project management plan of a Desktop based board game, Othello.

1.1.1 Project Overview

This is a 3D board game project based on the realistic and tangible version played in real life, but with some extensions towards additional features. The basic concept of the game includes an 8x8 board where the player with the black discs goes first and the discs are to be placed adjacent or next to their oppositely colored neighboring discs in doing so, the player with most discs on the board wins. Additional features include the user can have the option of selecting to play against another player or against the computer itself. He/she can have a choice of selecting different configurations of the disc's before commencing the game, this is just to shift from the traditional black and white discs. It will have 3 different difficulty levels i.e. Easy, Medium and Hard. Player can unlock new custom boards and discs by earning and spending diamonds (incentive and appreciation of playing the game more often) after beating another player or the computer. The player when playing against the player can add comments that would get deleted or fade away shortly. The board game environment is three dimensional giving a realistic simulation.

1.1.2 Project Deliverables

Project deliverables for this game are Software requirement Specifications (SRS), Software Project Management Plan (SPMP) and Software Design Description (SDD) till the end of this semester i.e. Spring 2019, and Software Test Documentation (STD) with the implementation of the project till Fall 2019.

1.2 Project Organization

1.2.1 Software Process Model

I will be using the Spiral Model to come up with my project's documentation and development of my project because of the following reason:

- High amount of risk requiring risk analysis, since I have expertise in AI.
- Software is produced early in the software life cycle (this will be possible because of the evolutionary nature of the spiral model, where I will use prototyping during development).
- Project estimates in terms of schedule, cost etc. become more and more realistic as the project moves forward and loops in **spiral** get completed.

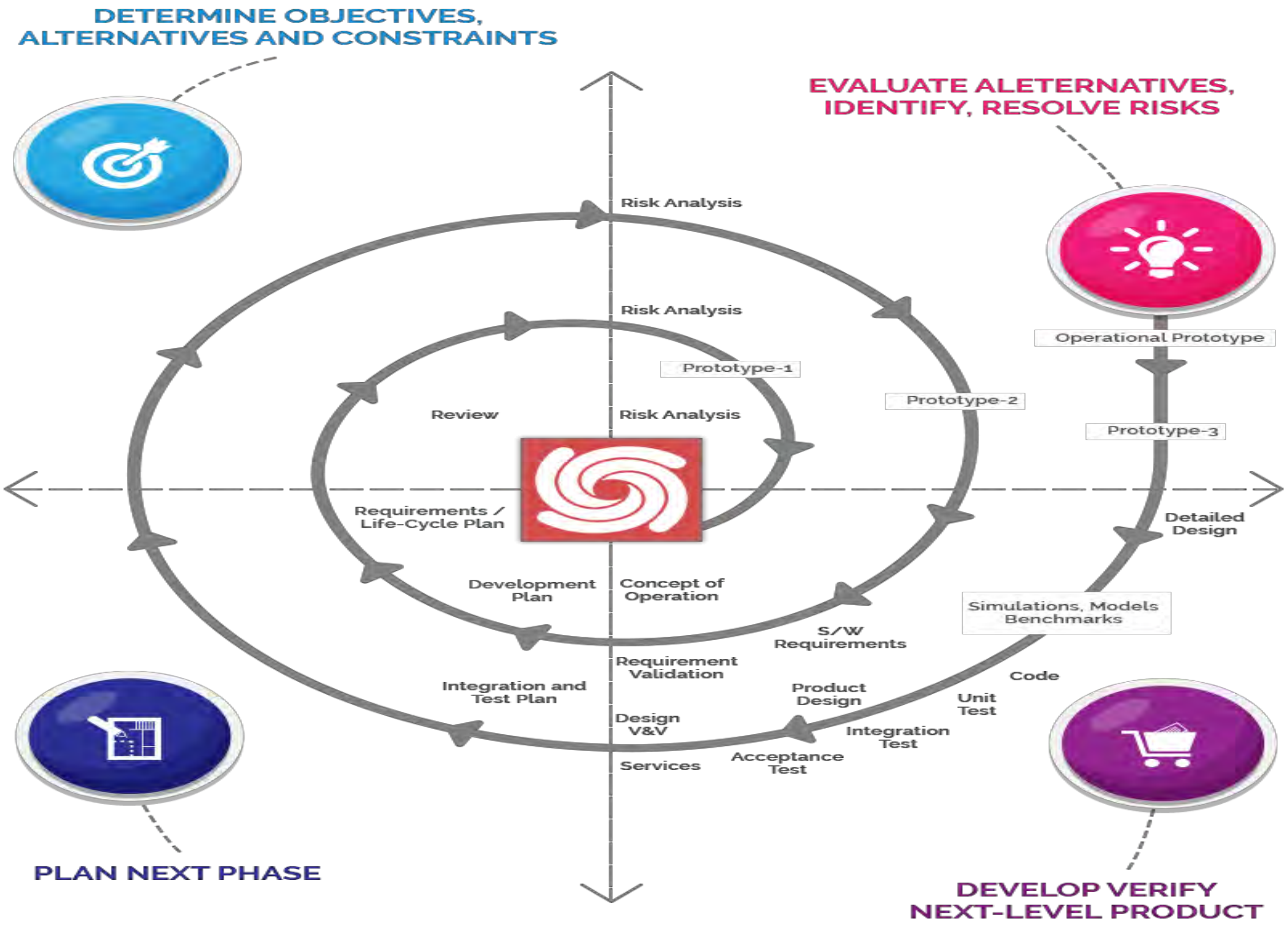


Figure 1.1: Spiral Model [2]

1.2.2 Roles and Responsibilities

I am single developer of this project so there is no division of roles and responsibilities.

1.2.3 Tools and Techniques

Following are the tools used in this project:

Table 1.1: Tools and Techniques

Tools		Techniques
MS-Word		Minimax Algorithm (Alpha-Beta Pruning)
Project Libre	ProjectLibre™	-
ArgoUML		-
Unity Game Engine		-

1.3 Project Management Plan

Following is the description of project management plan for this project (Desktop based board game). It explains how time and resources are managed throughout the life cycle of this game.

1.3.1 Tasks

There are two phases of project plan. First is the requirement and analysis and second is the design phase of this game. In requirements and analysis phase, the major tasks are to identify requirements, define use cases, develop analysis model, develop SRS and review SRS. In the second phase, the major tasks are to develop a design using Object Oriented Approach, design mode of user input, validate input, develop models and evaluate design. Following Figure 1, Figure 2, and Figure 3 shows tasks.

1.3.2 Description

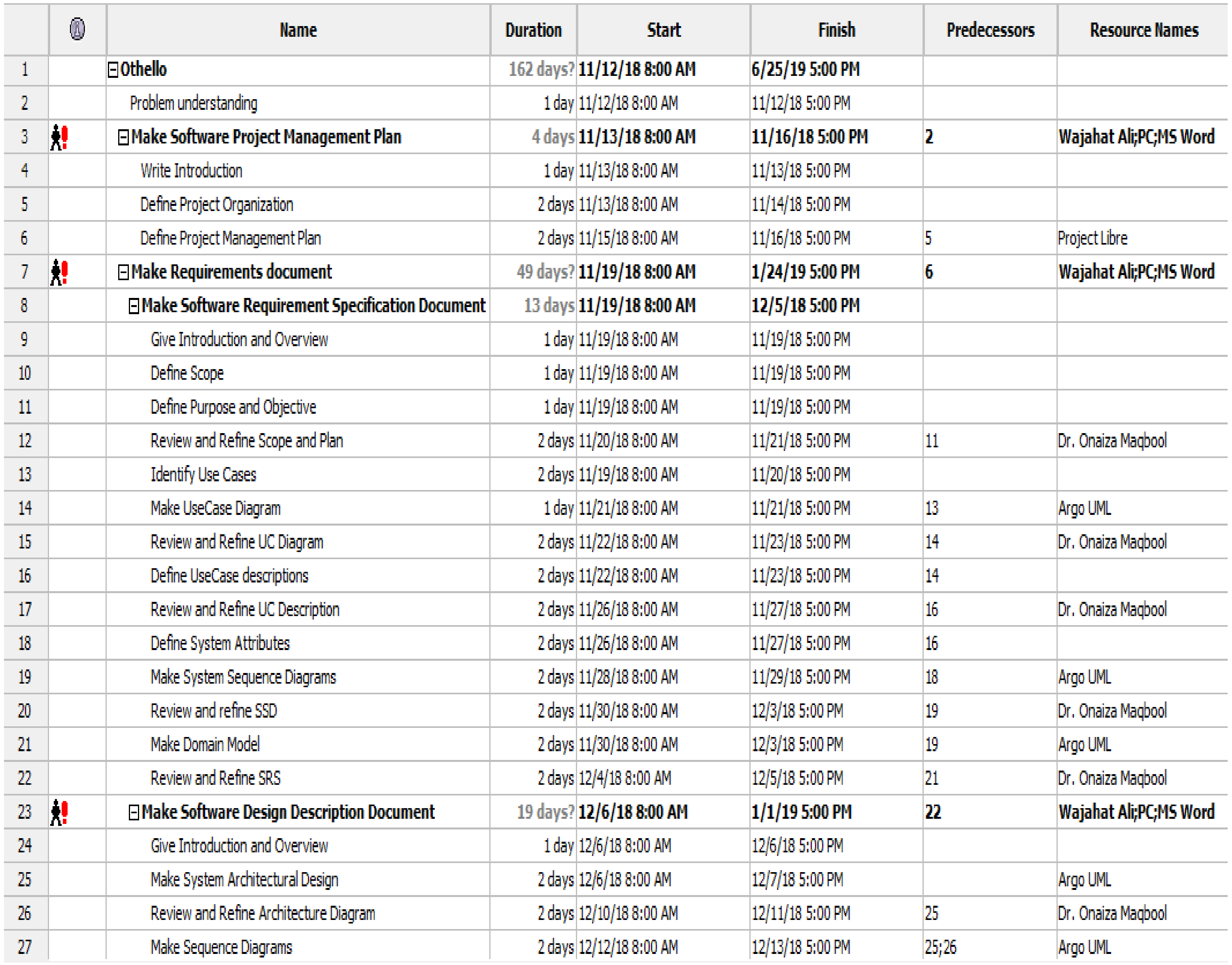
Following is the description of major tasks of both analysis and design phases.

Requirement and Analysis Phase:

- **Identify Requirements:** The main goal is to review case study and define requirements by meeting stakeholders.
- **Define Use Cases:** Define use cases and make a use case diagram, and often analysis models.
- **Develop SRS:** Provide a comprehensive description of the intended purpose and environment for software under development.

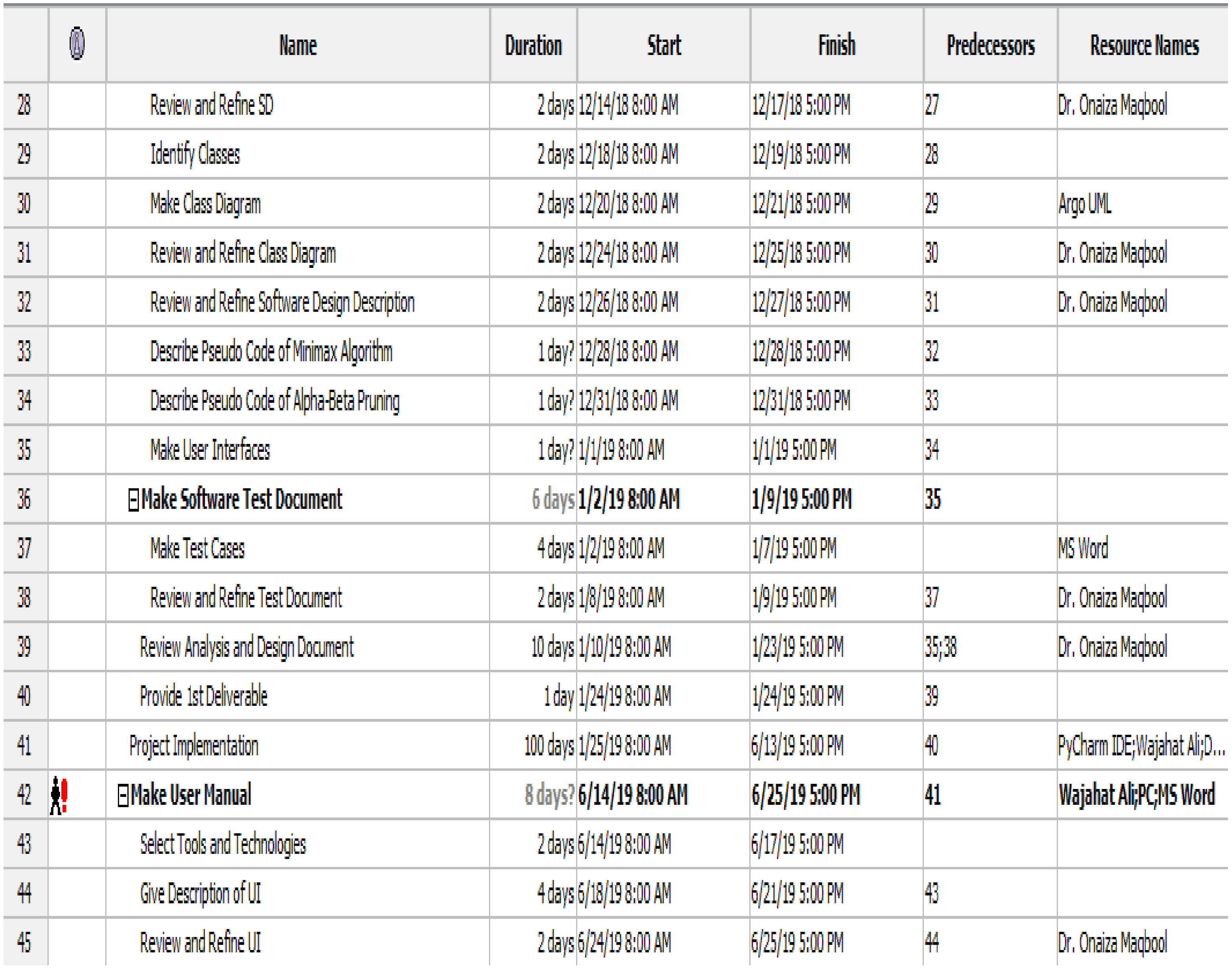
Design Phase:

- **Design Architecture:** Refers to the development of the structure of the project.
- **Make User Interfaces:** Extracts the lively view of the project.
- **Make Class Diagram:** Describe the attributes and operations of each entity associated with the system and the constraints imposed on the system.
- **Make Sequence Diagram:** Document the system's requirements and to flushes out its design by providing the interaction logic between the objects in the system in the time order that the interactions has taken place.



		Name	Duration	Start	Finish	Predecessors	Resource Names
1		**Othello**	162 days?	**11/12/18 8:00 AM**	**6/25/19 5:00 PM**		
2		Problem understanding	1 day	11/12/18 8:00 AM	11/12/18 5:00 PM		
3		**Make Software Project Management Plan**	4 days	**11/13/18 8:00 AM**	**11/16/18 5:00 PM**	**2**	**Wajahat Ali;PC;MS Word**
4		Write Introduction	1 day	11/13/18 8:00 AM	11/13/18 5:00 PM		
5		Define Project Organization	2 days	11/13/18 8:00 AM	11/14/18 5:00 PM		
6		Define Project Management Plan	2 days	11/15/18 8:00 AM	11/16/18 5:00 PM	5	Project Libre
7		**Make Requirements document**	49 days?	**11/19/18 8:00 AM**	**1/24/19 5:00 PM**	**6**	**Wajahat Ali;PC;MS Word**
8		**Make Software Requirement Specification Document**	13 days	**11/19/18 8:00 AM**	**12/5/18 5:00 PM**		
9		Give Introduction and Overview	1 day	11/19/18 8:00 AM	11/19/18 5:00 PM		
10		Define Scope	1 day	11/19/18 8:00 AM	11/19/18 5:00 PM		
11		Define Purpose and Objective	1 day	11/19/18 8:00 AM	11/19/18 5:00 PM		
12		Review and Refine Scope and Plan	2 days	11/20/18 8:00 AM	11/21/18 5:00 PM	11	Dr. Onaiza Maqbool
13		Identify Use Cases	2 days	11/19/18 8:00 AM	11/20/18 5:00 PM		
14		Make UseCase Diagram	1 day	11/21/18 8:00 AM	11/21/18 5:00 PM	13	Argo UML
15		Review and Refine UC Diagram	2 days	11/22/18 8:00 AM	11/23/18 5:00 PM	14	Dr. Onaiza Maqbool
16		Define UseCase descriptions	2 days	11/22/18 8:00 AM	11/23/18 5:00 PM	14	
17		Review and Refine UC Description	2 days	11/26/18 8:00 AM	11/27/18 5:00 PM	16	Dr. Onaiza Maqbool
18		Define System Attributes	2 days	11/26/18 8:00 AM	11/27/18 5:00 PM	16	
19		Make System Sequence Diagrams	2 days	11/28/18 8:00 AM	11/29/18 5:00 PM	18	Argo UML
20		Review and refine SSD	2 days	11/30/18 8:00 AM	12/3/18 5:00 PM	19	Dr. Onaiza Maqbool
21		Make Domain Model	2 days	11/30/18 8:00 AM	12/3/18 5:00 PM	19	Argo UML
22		Review and Refine SRS	2 days	12/4/18 8:00 AM	12/5/18 5:00 PM	21	Dr. Onaiza Maqbool
23		**Make Software Design Description Document**	19 days?	**12/6/18 8:00 AM**	**1/1/19 5:00 PM**	**22**	**Wajahat Ali;PC;MS Word**
24		Give Introduction and Overview	1 day	12/6/18 8:00 AM	12/6/18 5:00 PM		
25		Make System Architectural Design	2 days	12/6/18 8:00 AM	12/7/18 5:00 PM		Argo UML
26		Review and Refine Architecture Diagram	2 days	12/10/18 8:00 AM	12/11/18 5:00 PM	25	Dr. Onaiza Maqbool
27		Make Sequence Diagrams	2 days	12/12/18 8:00 AM	12/13/18 5:00 PM	25;26	Argo UML

Figure 1.2: Time Table 1



		Name	Duration	Start	Finish	Predecessors	Resource Names
28		Review and Refine SD	2 days	12/14/18 8:00 AM	12/17/18 5:00 PM	27	Dr. Onaiza Maqbool
29		Identify Classes	2 days	12/18/18 8:00 AM	12/19/18 5:00 PM	28	
30		Make Class Diagram	2 days	12/20/18 8:00 AM	12/21/18 5:00 PM	29	Argo UML
31		Review and Refine Class Diagram	2 days	12/24/18 8:00 AM	12/25/18 5:00 PM	30	Dr. Onaiza Maqbool
32		Review and Refine Software Design Description	2 days	12/26/18 8:00 AM	12/27/18 5:00 PM	31	Dr. Onaiza Maqbool
33		Describe Pseudo Code of Minimax Algorithm	1 day?	12/28/18 8:00 AM	12/28/18 5:00 PM	32	
34		Describe Pseudo Code of Alpha-Beta Pruning	1 day?	12/31/18 8:00 AM	12/31/18 5:00 PM	33	
35		Make User Interfaces	1 day?	1/1/19 8:00 AM	1/1/19 5:00 PM	34	
36		⊟ **Make Software Test Document**	6 days	**1/2/19 8:00 AM**	**1/9/19 5:00 PM**	**35**	
37		Make Test Cases	4 days	1/2/19 8:00 AM	1/7/19 5:00 PM		MS Word
38		Review and Refine Test Document	2 days	1/8/19 8:00 AM	1/9/19 5:00 PM	37	Dr. Onaiza Maqbool
39		Review Analysis and Design Document	10 days	1/10/19 8:00 AM	1/23/19 5:00 PM	35;38	Dr. Onaiza Maqbool
40		Provide 1st Deliverable	1 day	1/24/19 8:00 AM	1/24/19 5:00 PM	39	
41		Project Implementation	100 days	1/25/19 8:00 AM	6/13/19 5:00 PM	40	PyCharm IDE;Wajahat Ali;D...
42		⊟ **Make User Manual**	8 days?	**6/14/19 8:00 AM**	**6/25/19 5:00 PM**	**41**	**Wajahat Ali;PC;MS Word**
43		Select Tools and Technologies	2 days	6/14/19 8:00 AM	6/17/19 5:00 PM		
44		Give Description of UI	4 days	6/18/19 8:00 AM	6/21/19 5:00 PM	43	
45		Review and Refine UI	2 days	6/24/19 8:00 AM	6/25/19 5:00 PM	44	Dr. Onaiza Maqbool

Figure 1.3: Time Table 2

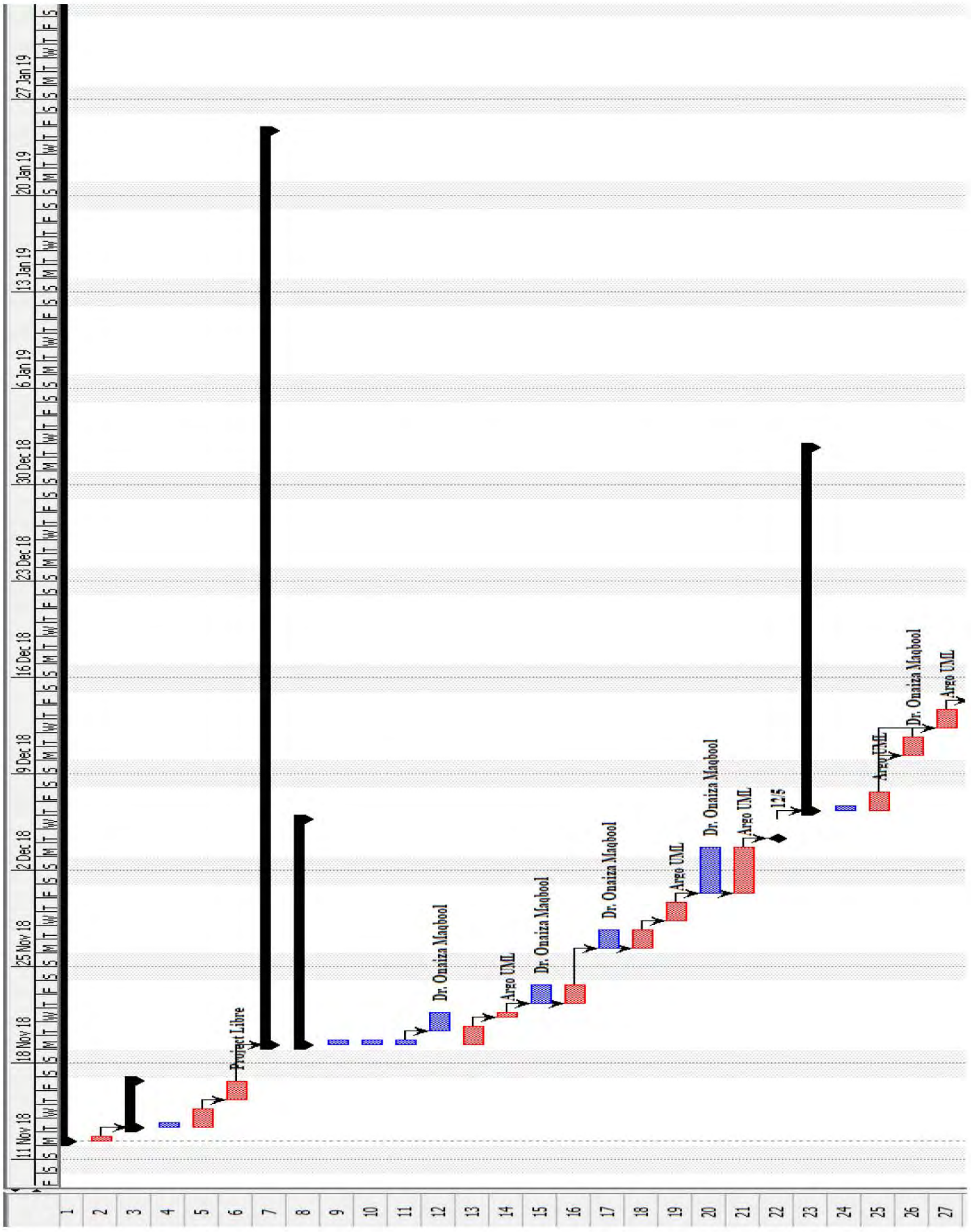


Figure 1.4: Gantt chart 1

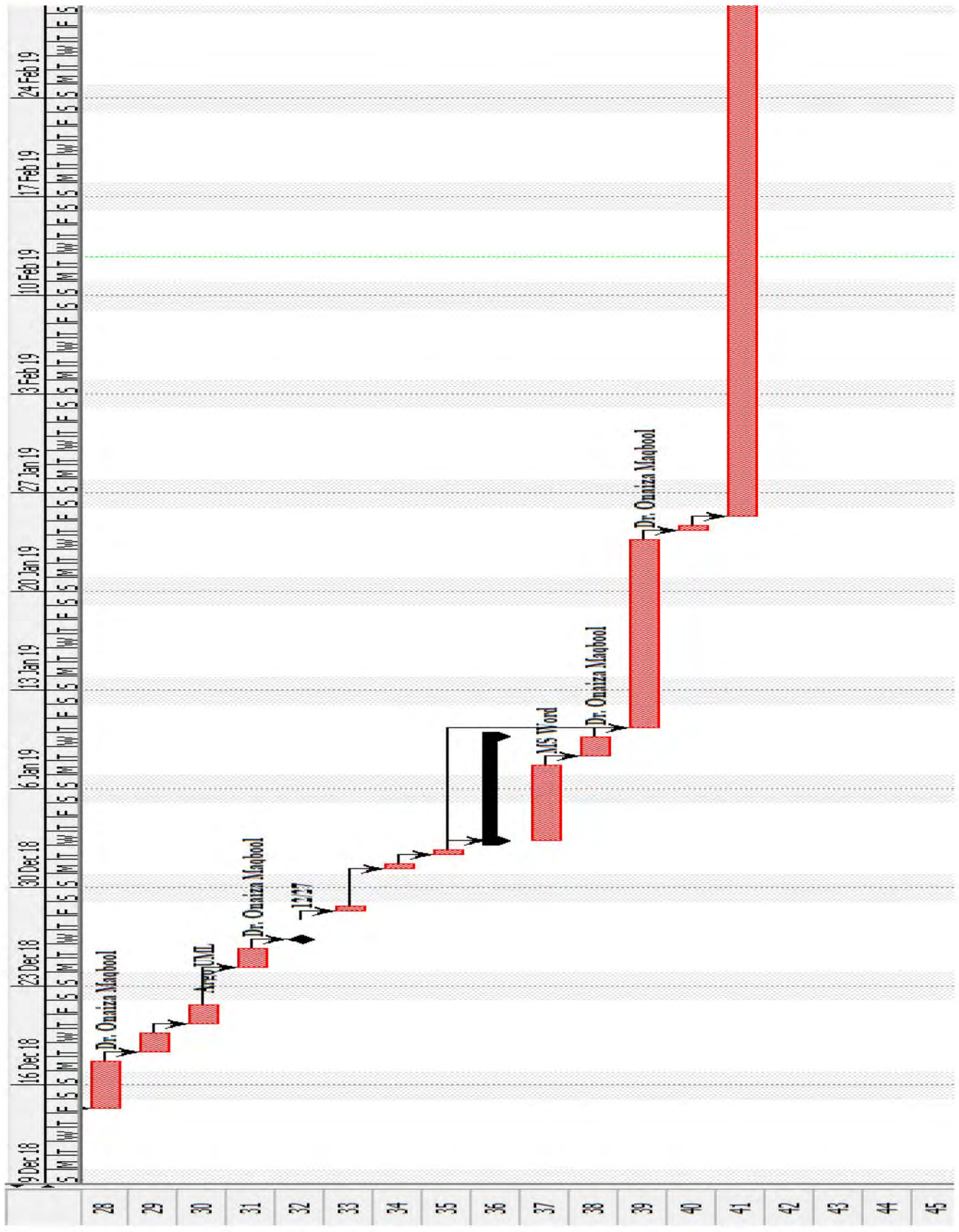


Figure 1.5: Gantt chart 2

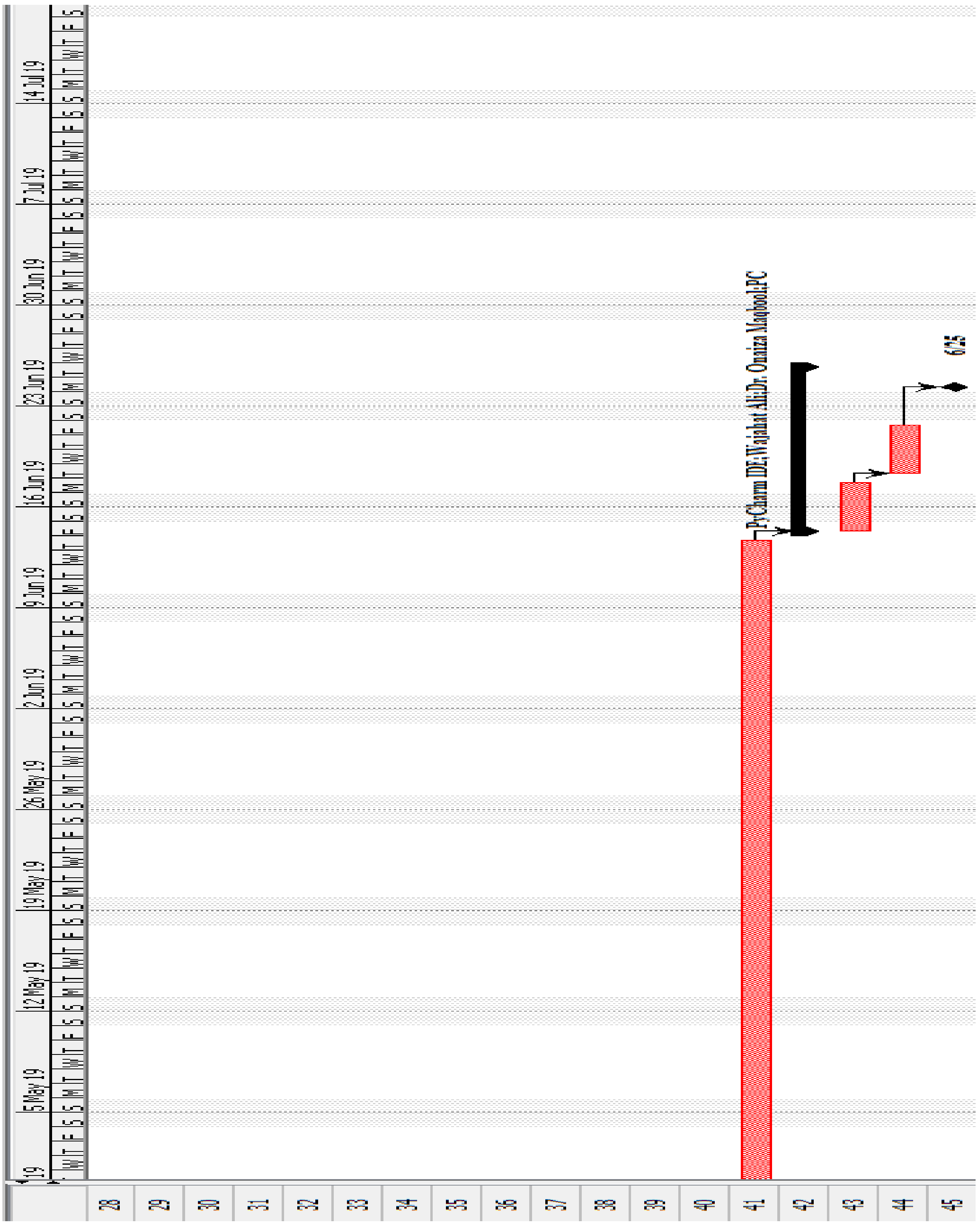


Figure 1.6: Gantt chart 3

Chapter 2: PROJECT BACKGROUND AND UNDERSTANDING

2. Project Background and Understanding

2.1 Game Definition

Othello is a strategy **board game** for two players, played on an 8×8 uncheckered **board**. There are sixty-four identical **game** pieces called disks (often spelled "discs"), which are light on one side and dark on the other. Players take turns placing disks on the **board** with their assigned color facing up.

2.2 Similar Games

Othello is sometimes referred to as **Reversi** [3]. But in some context, both games are separate games from one another.

Difference between Othello and Reversi

There are three main differences between pure Reversi and Othello, which are as follows:

1. Othello consists of a central pool of discs, from where both the players pick and make their move. On the other hand, Reversi has a separate pool of discs for each player.
2. Reversi is initiated with the board being completely empty, whereas in Othello, the central four spaces on the board are reserved with 2 pairs of black and white discs with the opposite colors placed face to face.
3. Reversi happens to be what the game was originally called, while Othello is the copyrighted version of Reversi and it still is.

Do these differences matter?

The differences mentioned earlier are quite minute and simply make no difference to the gameplay at all.

2.3 Game Rules

The rules of the game are simple. Each piece has a black side and a white side. On your turn, you place one piece on the board with your color facing up. You must place the piece so that an opponent's piece, or a row of opponent's pieces, is flanked by your pieces. All the opponent's pieces between your pieces are then turned over to become your color [4]. The game is synched with a clock, in the duration of which the game must end, and the winner will be declared with the player having the most discs.

2.4 Artificial Intelligence Techniques

2.4.1 Minimax Algorithm

Minimax sometimes referred to as MinMax or MM[5], is the most basic and fundamentally used algorithm in games like Chess, Othello, Reversi, Tik-Tak-Toe etc. where one player is supposed to increase his/her winning chances by decreasing that of the others. In other words, it is a decision rule used in **artificial intelligence**, **decision theory**, **game theory**, **statistics** and **philosophy** for minimizing the possible **loss** for a worst case (maximum loss) scenario.

2.4.1.1 Minimax Algorithm Working

Say we want to write a program that can play a turn-based game like chess, one aspect of this is the search algorithm which is what would allow the program to look ahead at possible future positions before deciding what moves it wants to make in the current position.

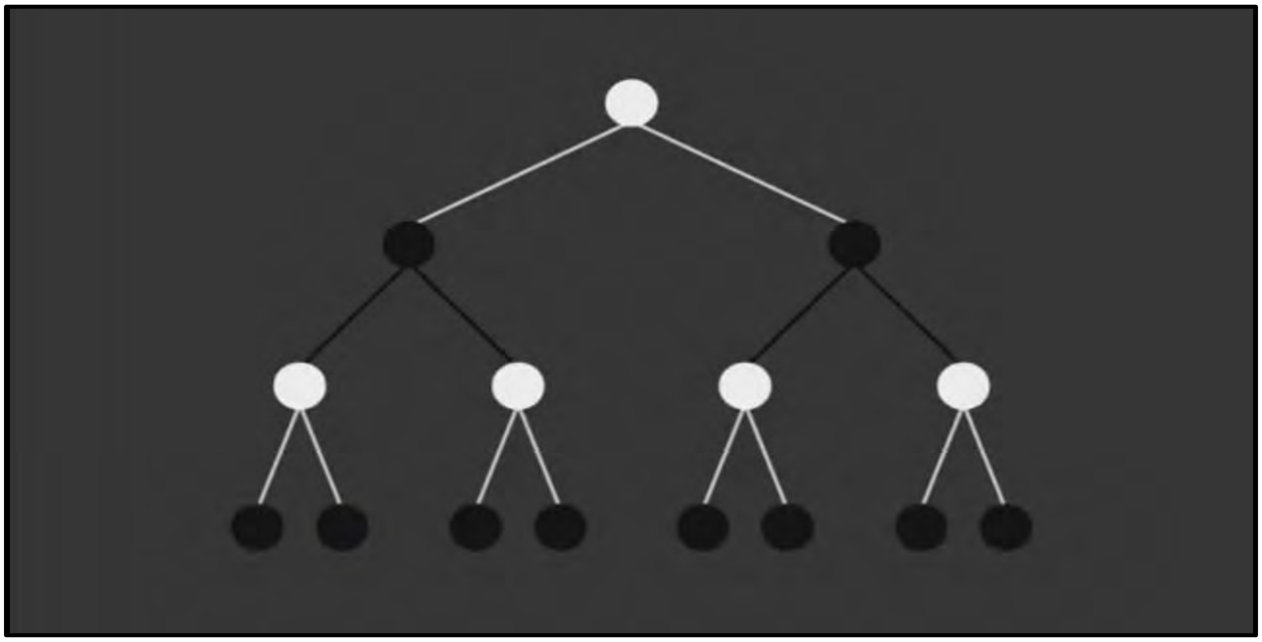


Figure 2.1: Minimax Search Tree Phase 1

The white dot represents some position in our game with the white side to move. To keep things simple, let’s say that in every position there are only two moves to choose between. We can visualize these moves as two separate branches which are two new branches which is now of course, black’s turn to move. We can continue expanding the tree of moves until either we reach the end of the game or we decide to stop because going deeper would take too much time.

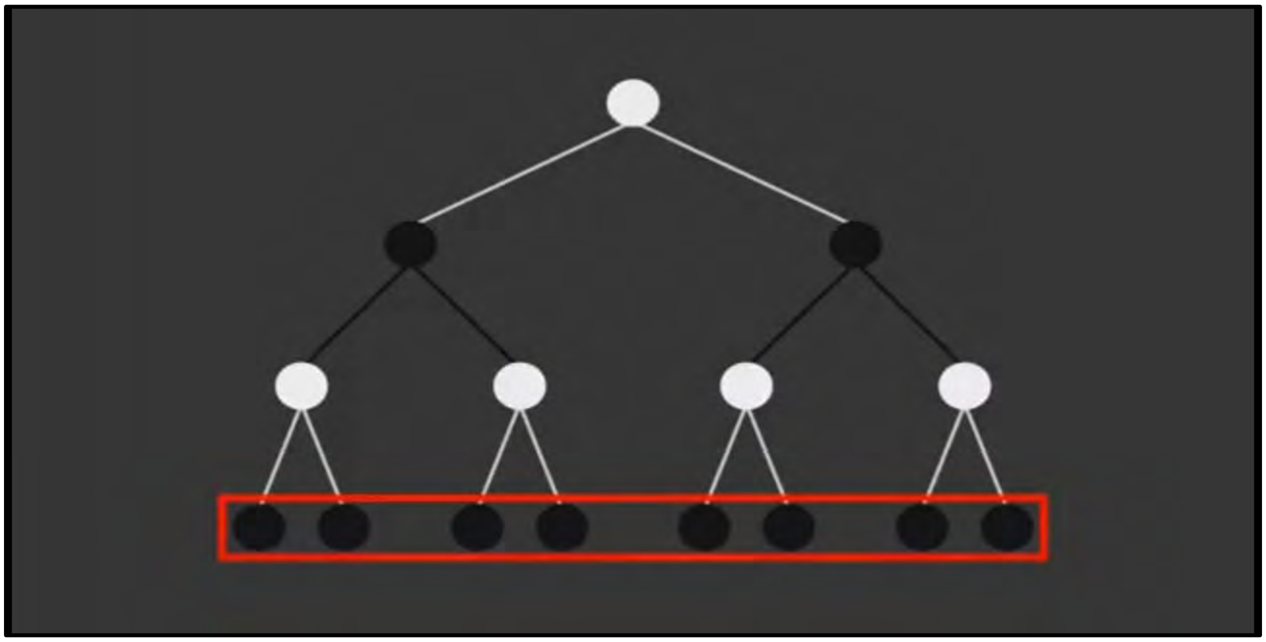


Figure 2.2: Minimax Search Tree Phase 2

Either way at the end of the tree, we will perform a static evaluation on the final positions which happen to be the leaf nodes. A static evaluation just means trying to estimate how good the position is for one side without making any more moves. E.g. a crude approach in chess will add up the values of the remaining white pieces and track the values of the remaining black pieces.

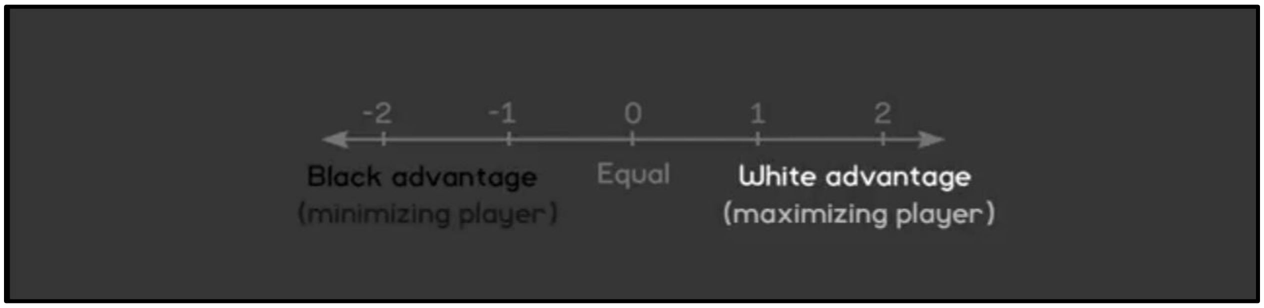


Figure 2.3: Maximizing and Minimizing Range

So, large values will favor white while small values would favor black. For this reason, white is always trying to maximize the evaluation, while black is trying to minimize it.

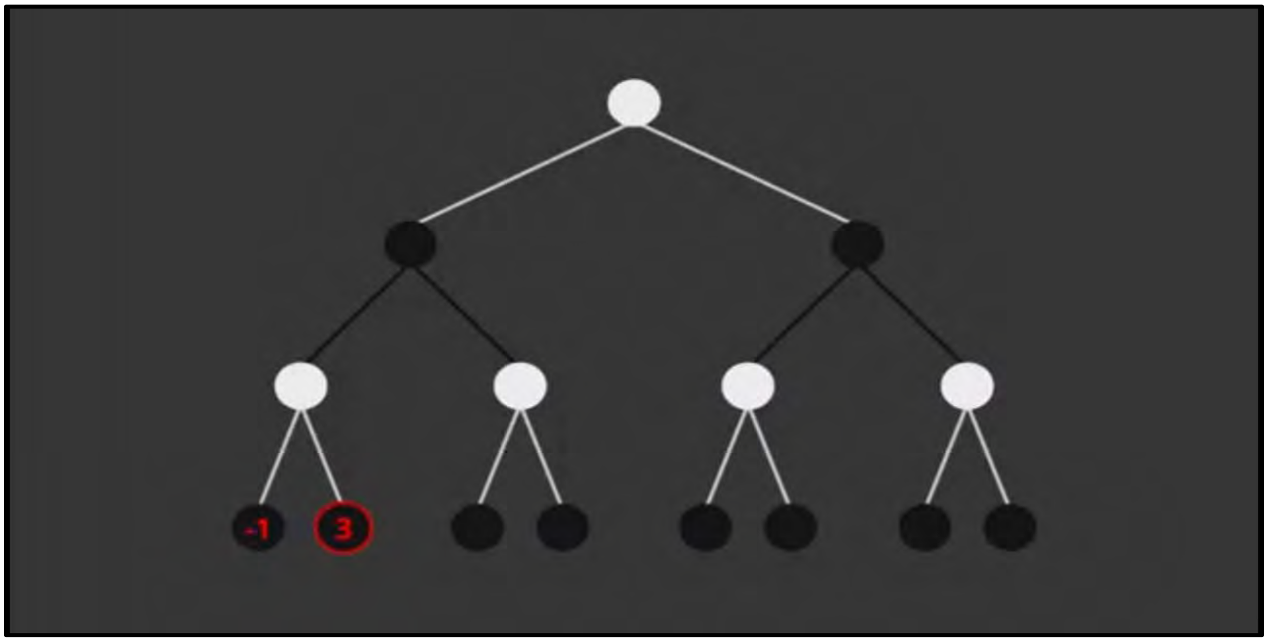


Figure 2.4: Minimax Search Tree Phase 3

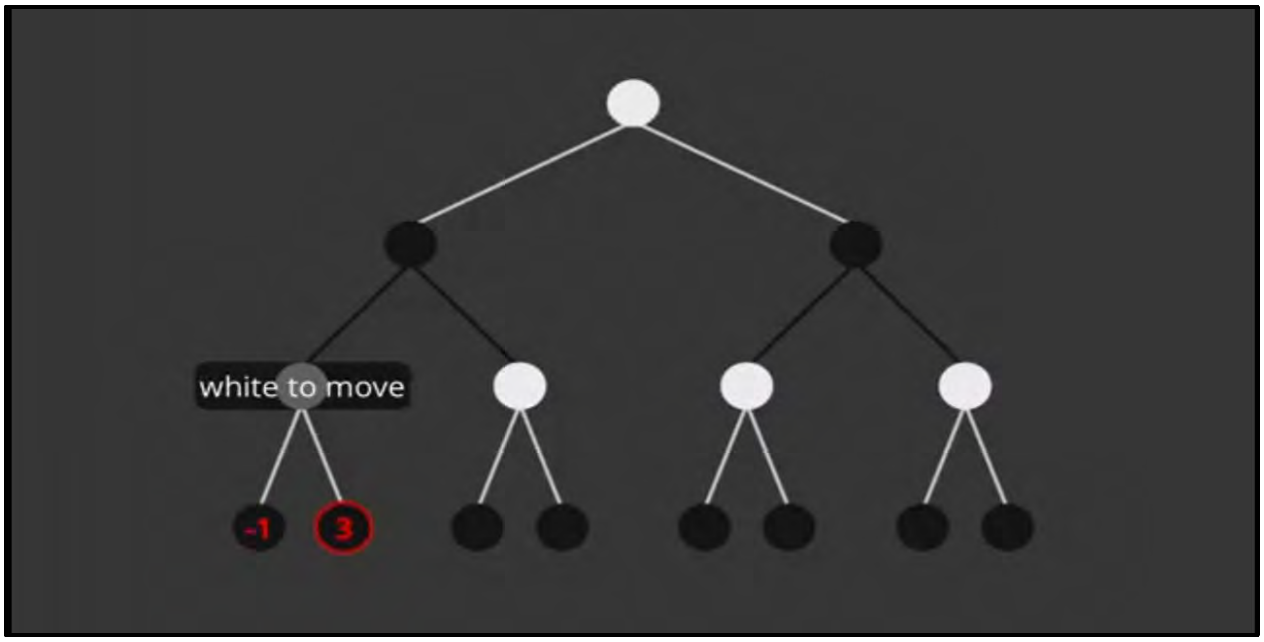


Figure 2.5: Minimax Search Tree Phase 4

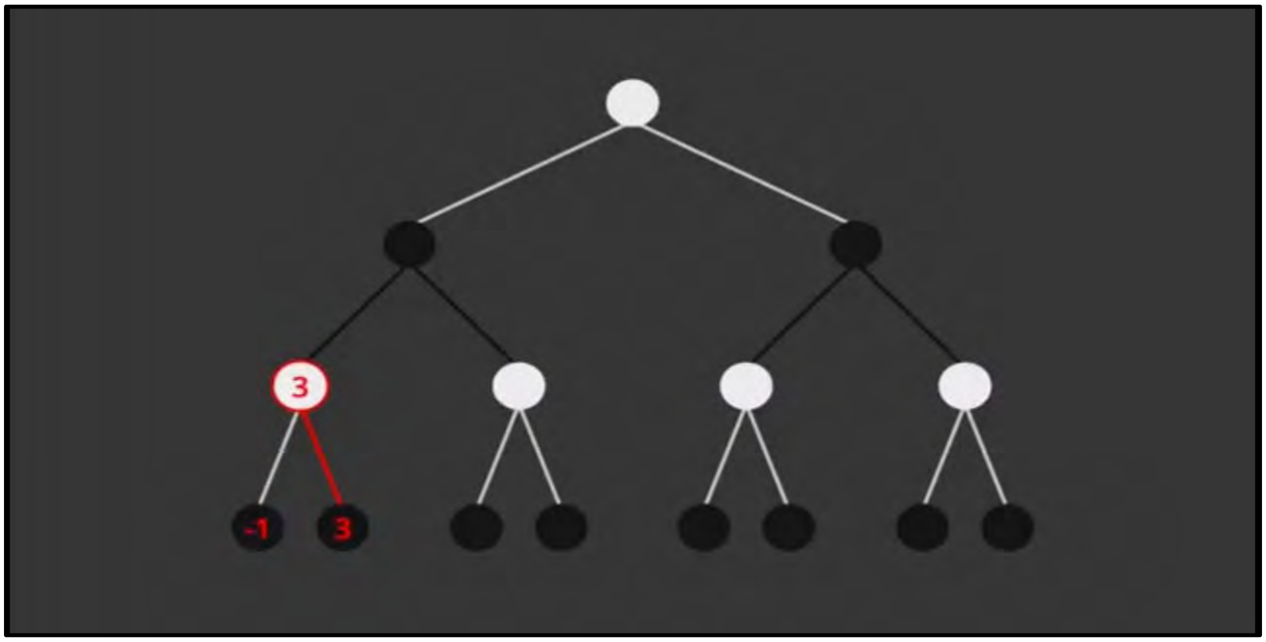


Figure 2.6: Minimax Search Tree Phase 5

Let's start with these positions on the bottom left. Consider after evaluating them, they come out as -1 and 3 respectively. In the previous position, it was white's turn to make the move and since white will choose the move that leads to the highest evaluation, we can assign this position the value of the node with the greater value which in this case is 3.

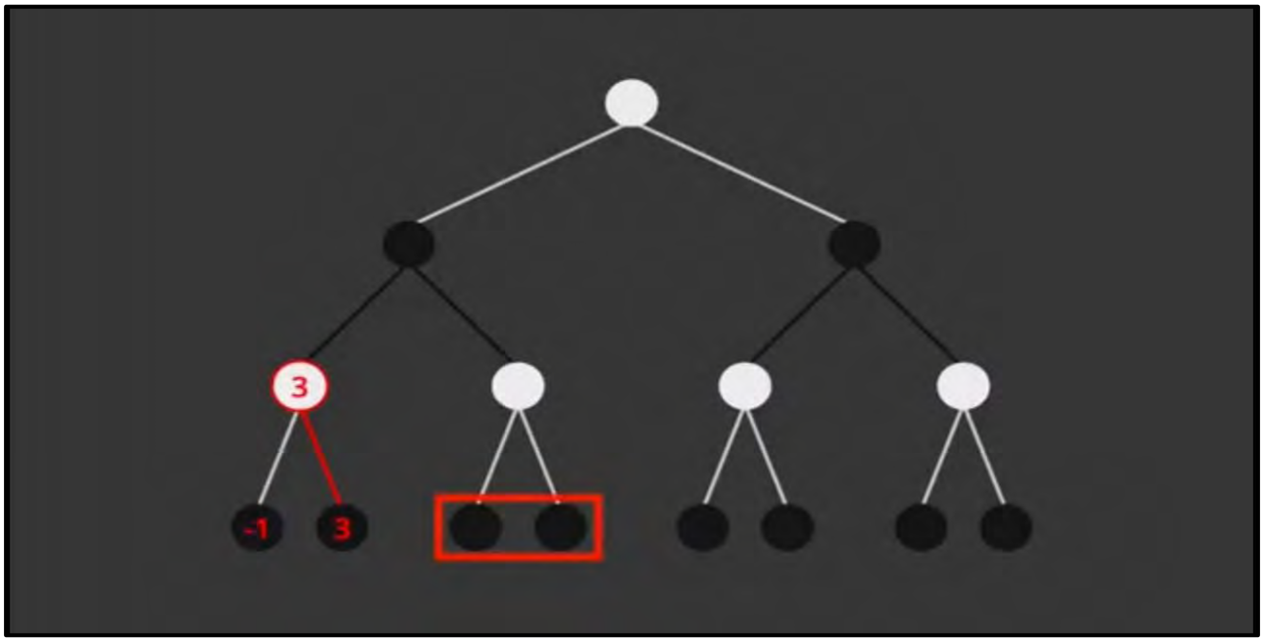


Figure 2.7: Minimax Search Tree Phase 6

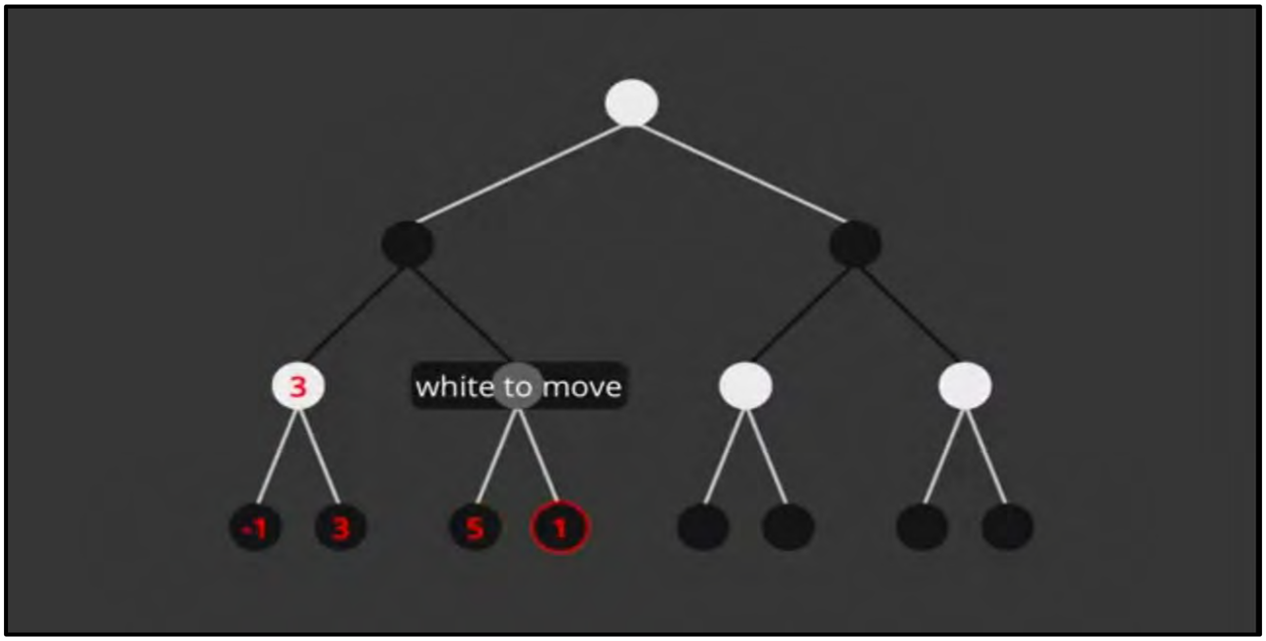


Figure 2.8: Minimax Search Tree Phase 7

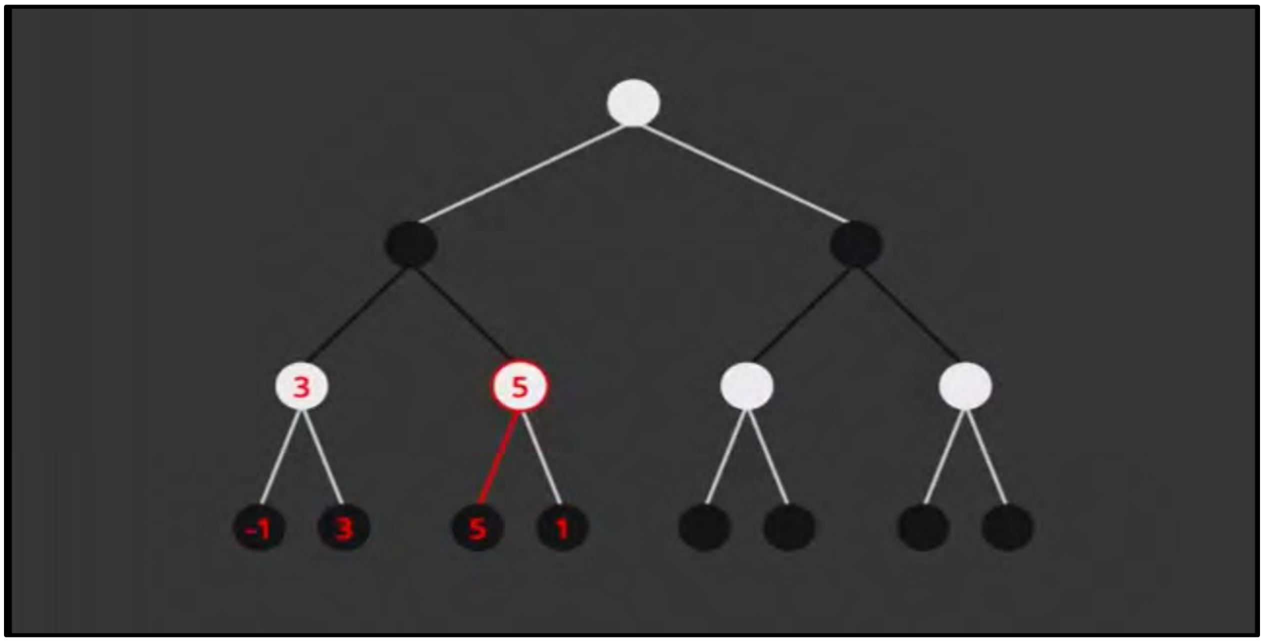


Figure 2.9: Minimax Search Tree Phase 8

Next, let's evaluate the next these positions. Say we get 5 and 1. Once again from the previous position, white would pick the move leading to the highest evaluation and so we can assign the value 5.

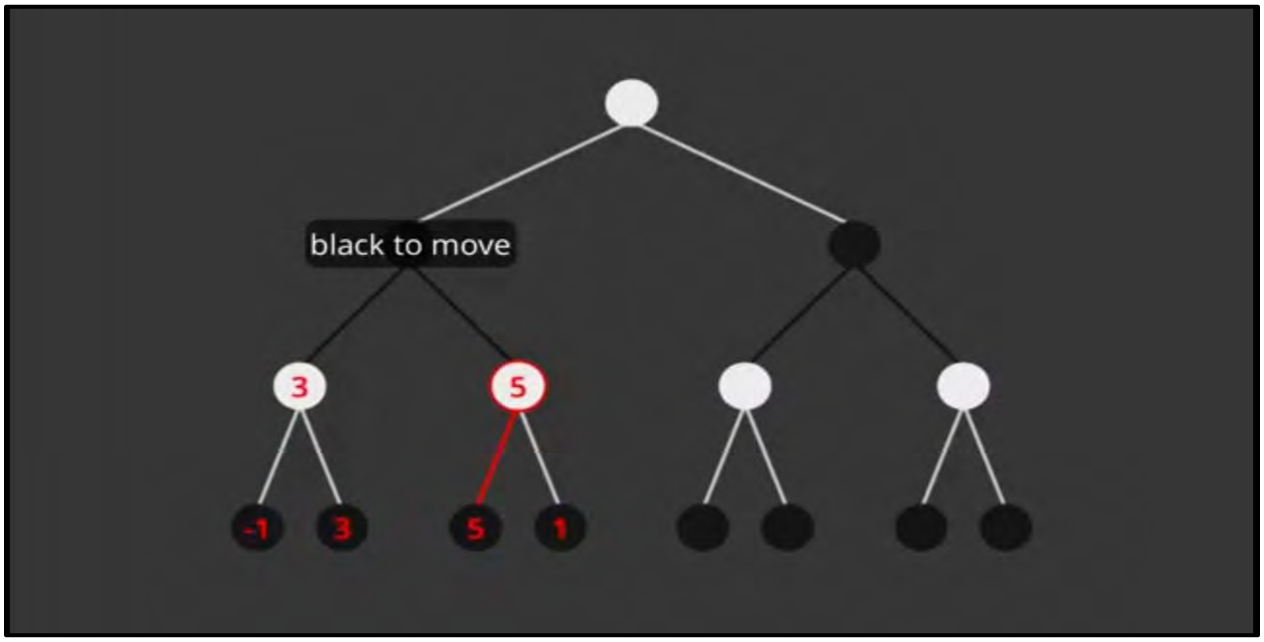


Figure 2.10: Minimax Search Tree Phase 9

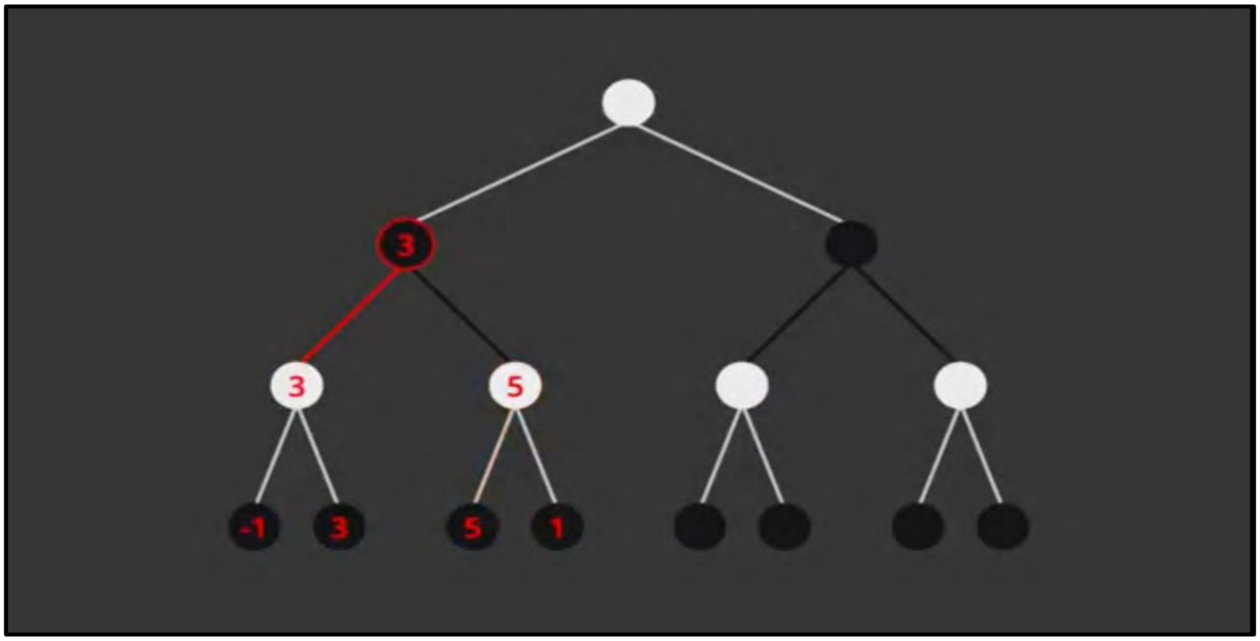


Figure 2.11: Minimax Search Tree Phase 10

We have now evaluated both the stemming from the position where it is black's turn to move. Black will choose the move that will lead to the lowest evaluation. So, we can assign this position a value of 3.

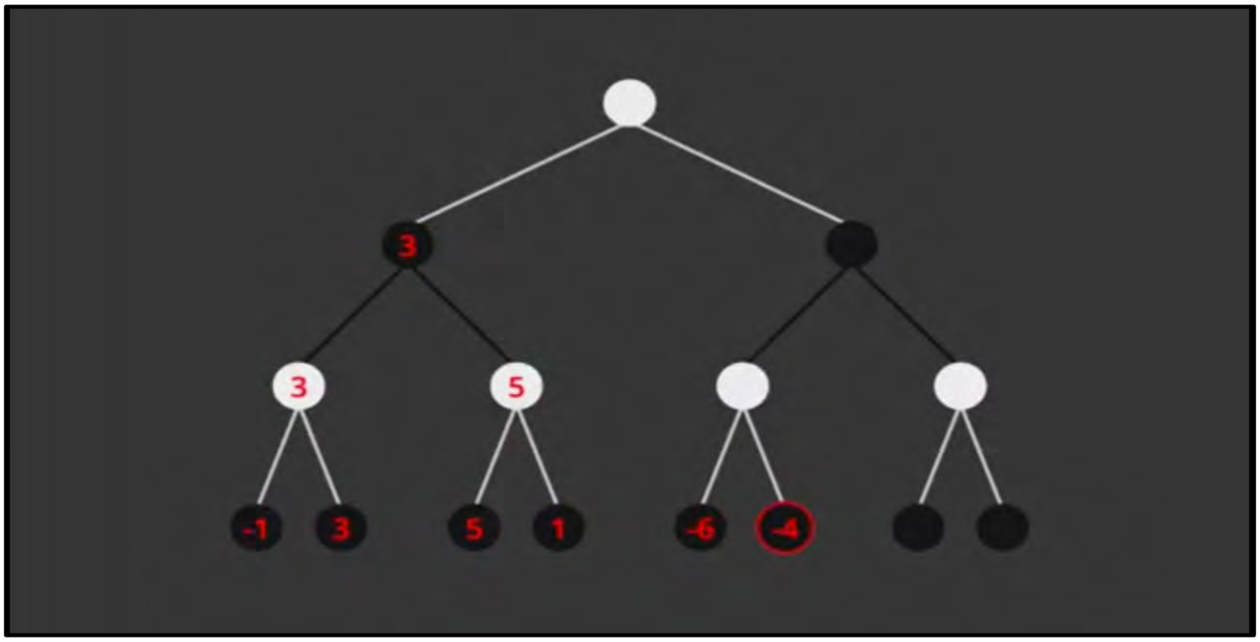


Figure 2.12: Minimax Search Tree Phase 11

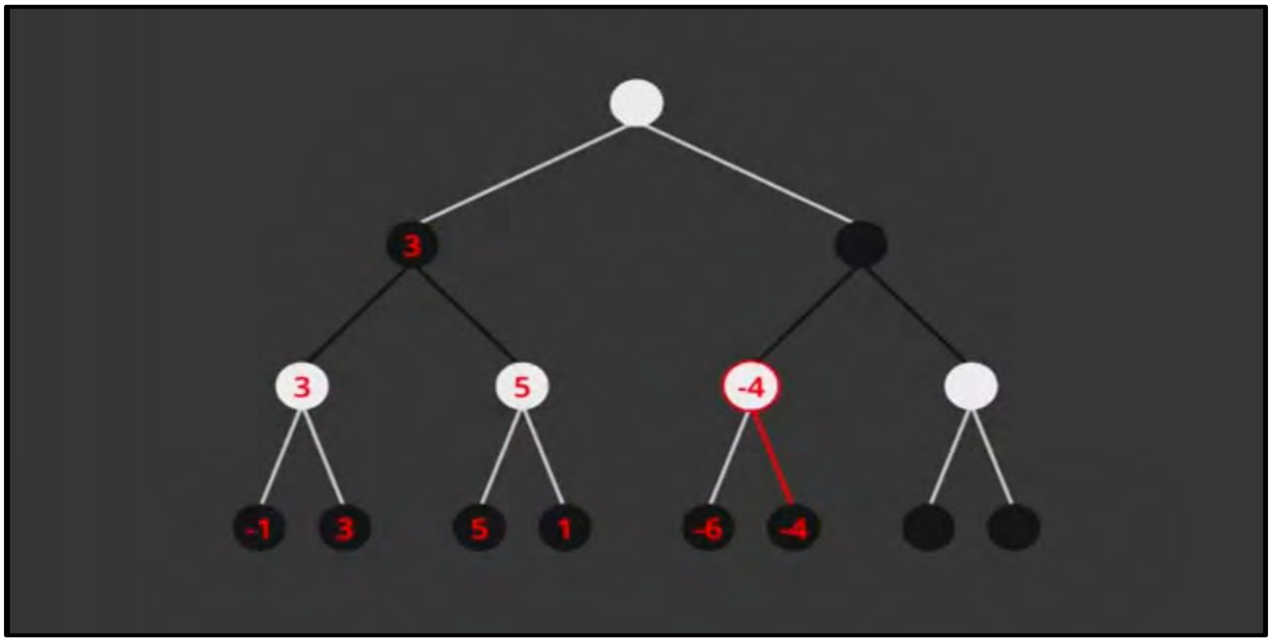


Figure 2.13: Minimax Search Tree Phase 12

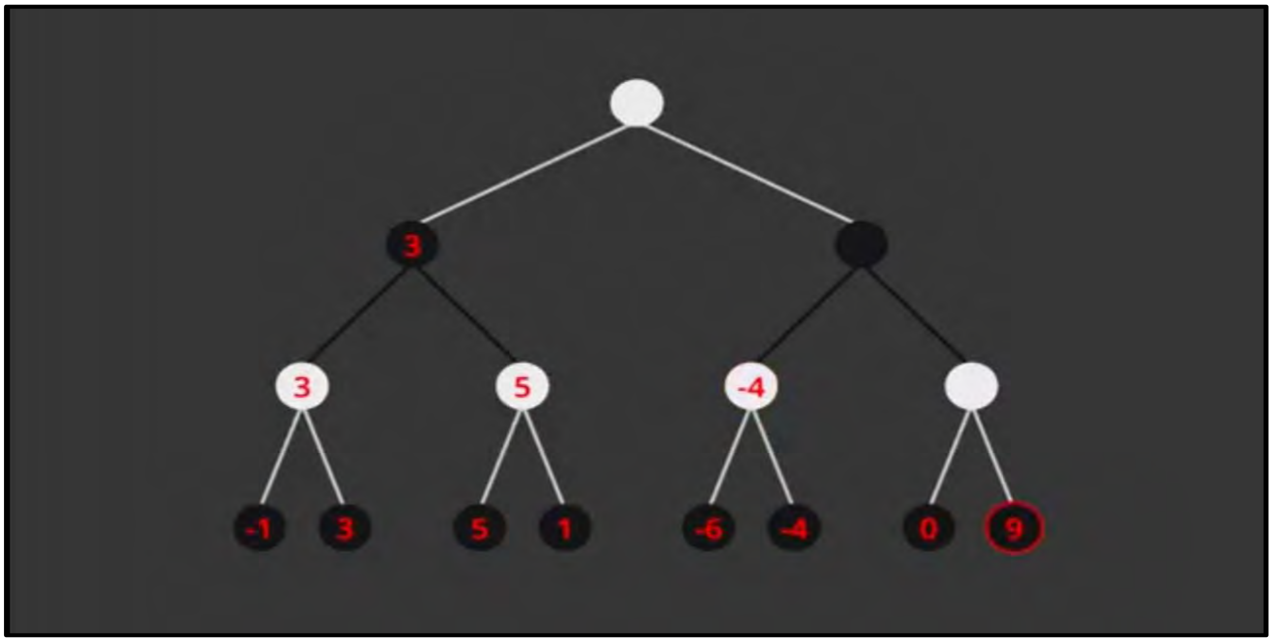


Figure 2.14: Minimax Search Tree Phase 13

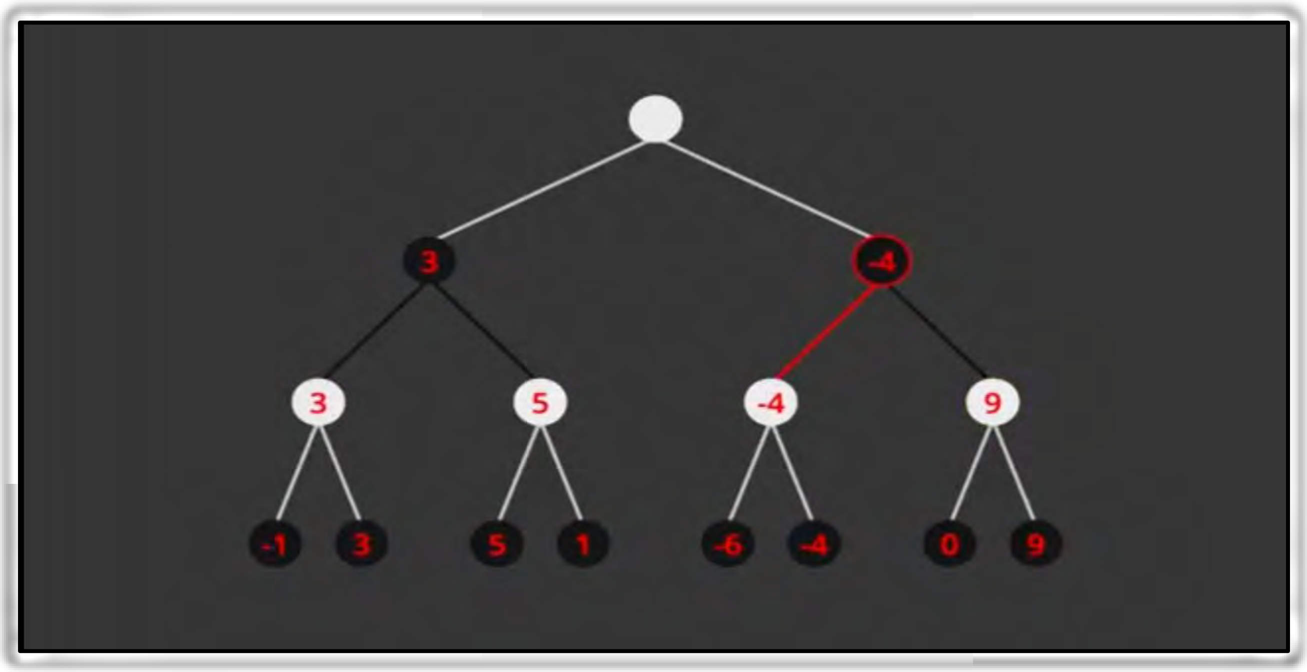


Figure 2.15: Minimax Search Tree Phase 14

Now, say that the remaining leaf nodes come out with values -6 and -4. White will pick the -4. The last leaf nodes are evaluated as 0 and 9, white chooses 9 and between -4 and 9, black will pick -4.

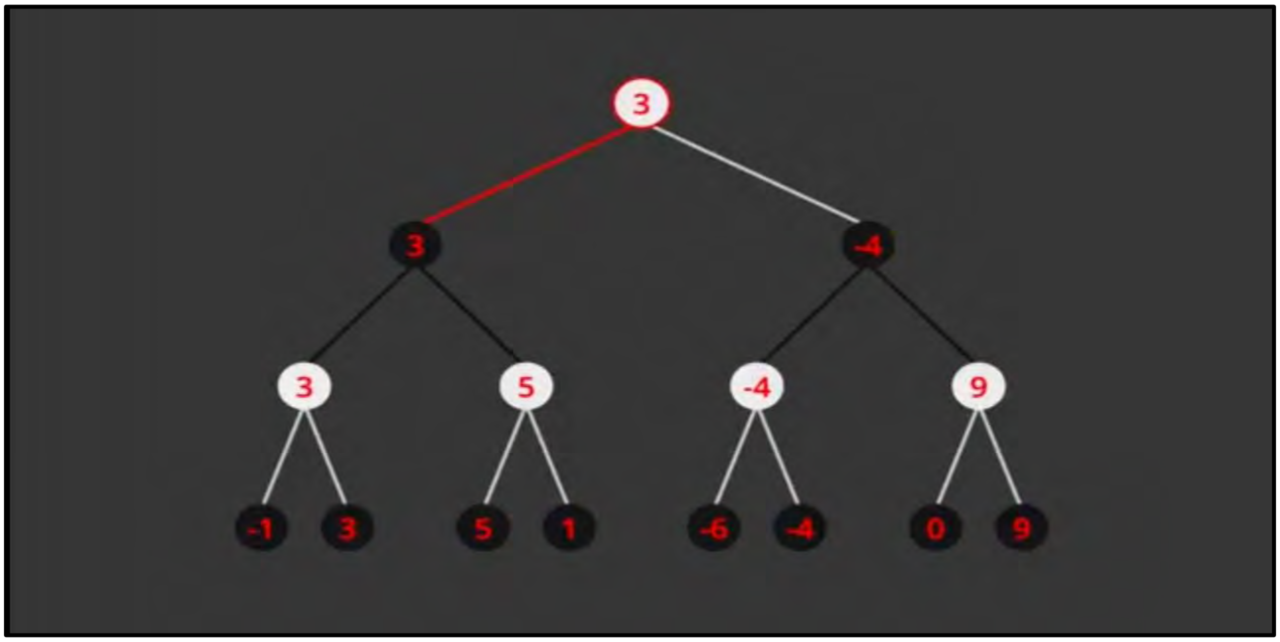


Figure 2.16: Minimax Search Tree Phase 15

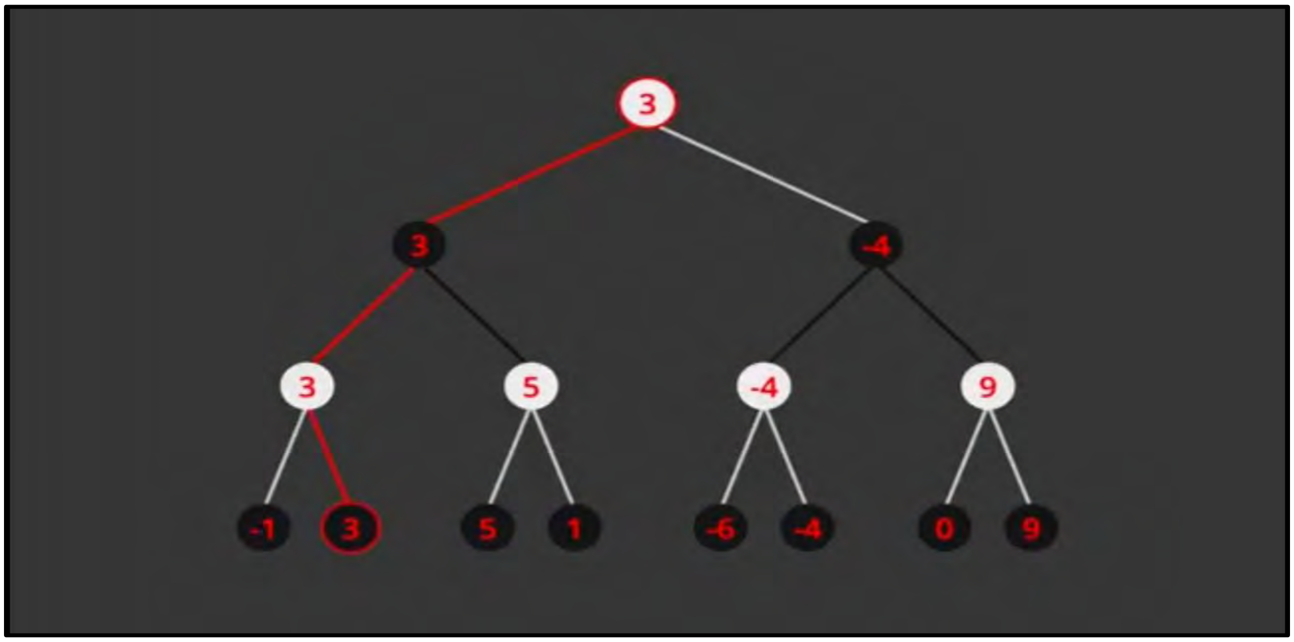


Figure 2.17: Minimax Search Tree Phase 16

At last we have arrived at the top of the tree, where we can see that white should choose the move on the left. Since that way even if black places the best move, white will still get a 3 position.

2.4.2 Alpha-Beta Pruning

The Alpha-Beta Pruning algorithm is equivalent to the minimax algorithm in that they both find the same best move from position and both will assign the same value of expected advantage to it. [6] This algorithm happens to be a modified version of the Minimax Algorithm. We simply don't have to visit all the *look-aheads*, we prune away or simply cut down the sub-tree that would lead us to the result we are in search of.

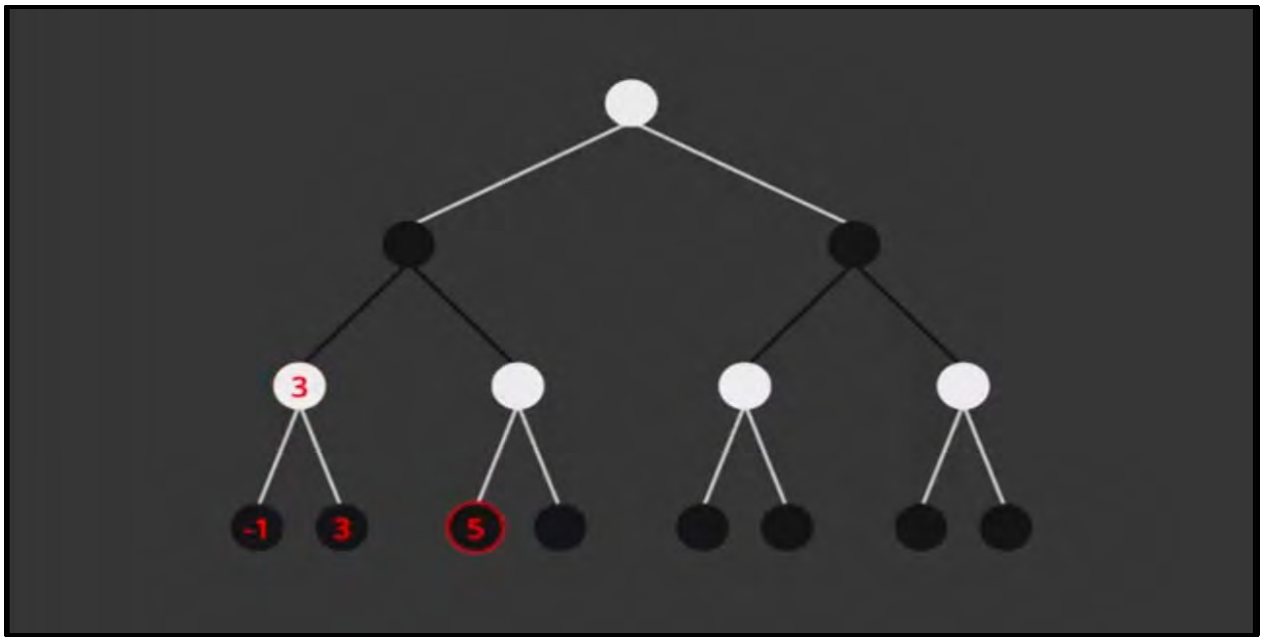


Figure 2.18: Alpha-Beta Pruning Search Tree Phase 1

Considering the same example that was previously used in Section 2.4.1, the first few steps are the

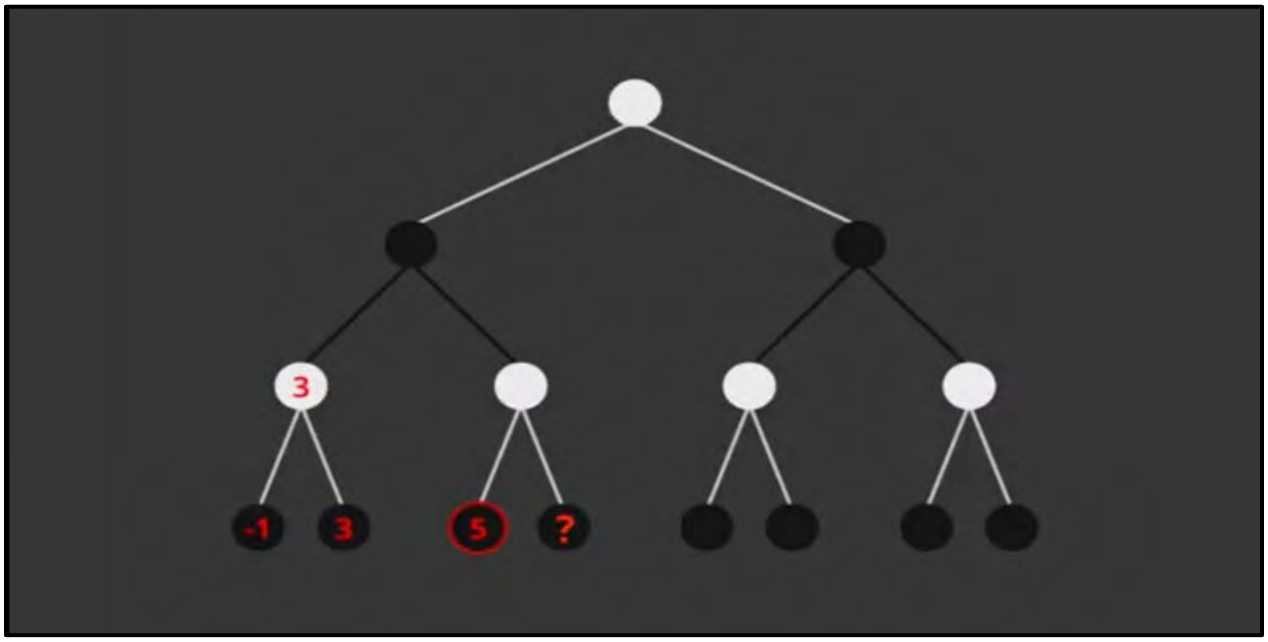


Figure 2.19: Alpha-Beta Pruning Search Tree Phase 2

same as that of the Minimax Algorithm but consider the situation after evaluating the node with the value 5. Without evaluating the other position, we know that white can at least get a 5 from here, so we can mark this position as greater than equal to 5. We can now see that black won't go down this branch because it already has a better option i.e. 3. This observation means that we don't have to waste any computation on evaluating this final position. We can simply pretend that it doesn't exist or in other words we have simply pruned it from the tree.

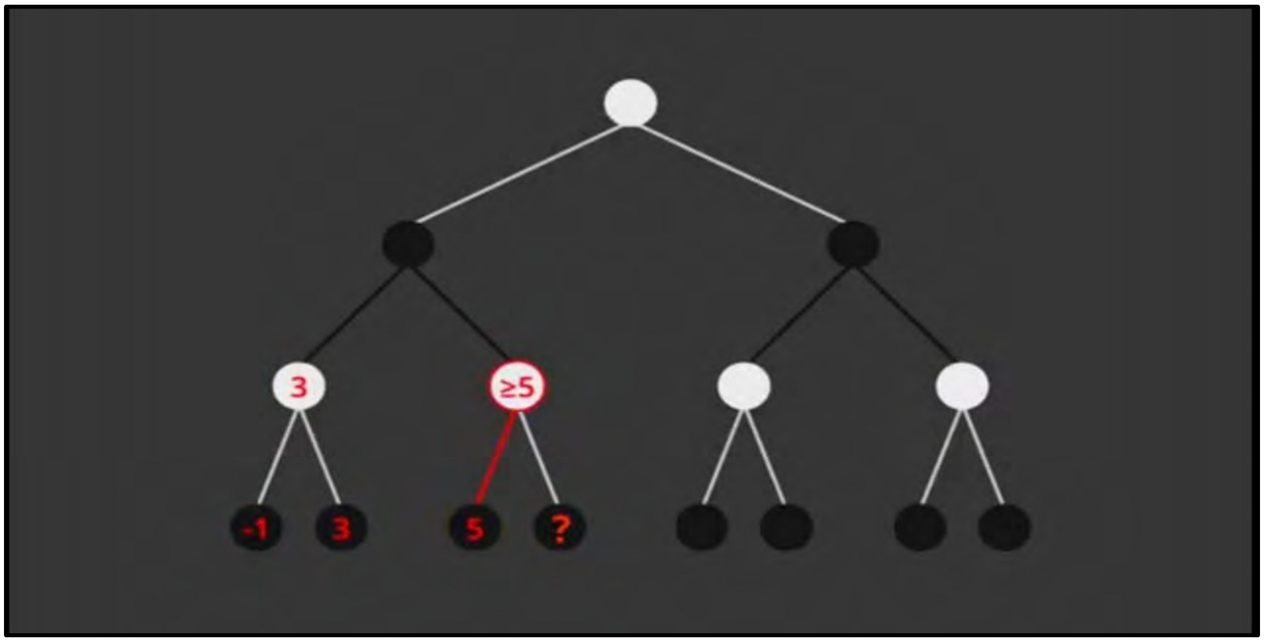


Figure 2.20: Alpha-Beta Pruning Search Tree Phase 3

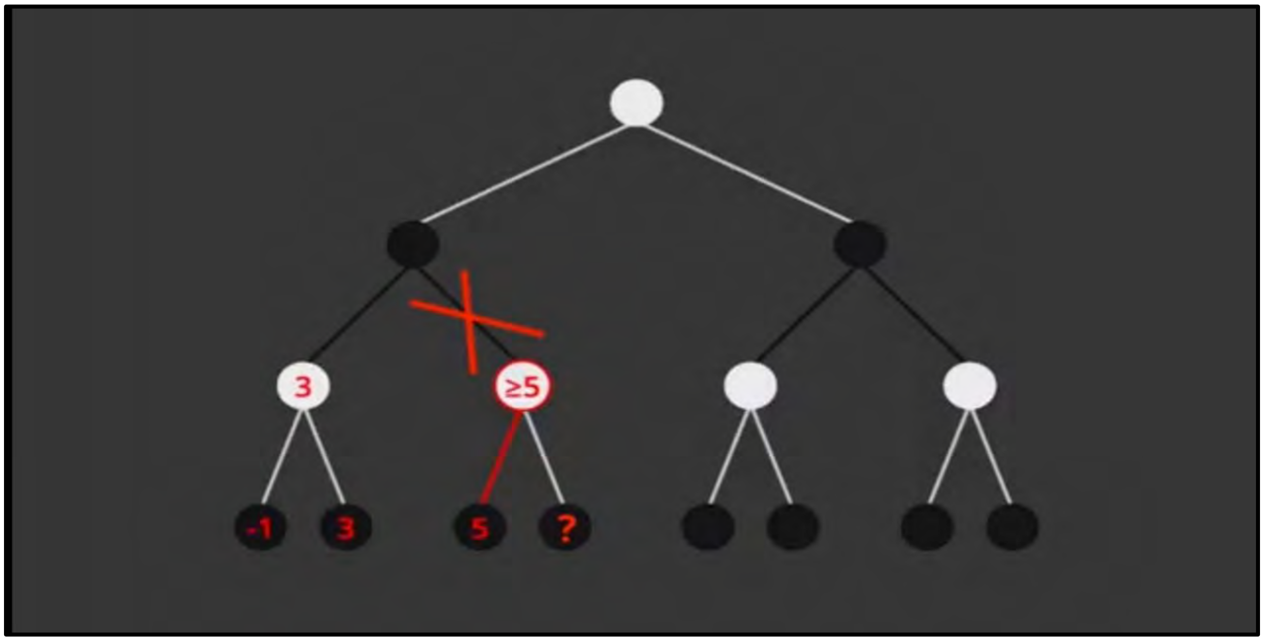


Figure 2.21: Alpha-Beta Pruning Search Tree Phase 4

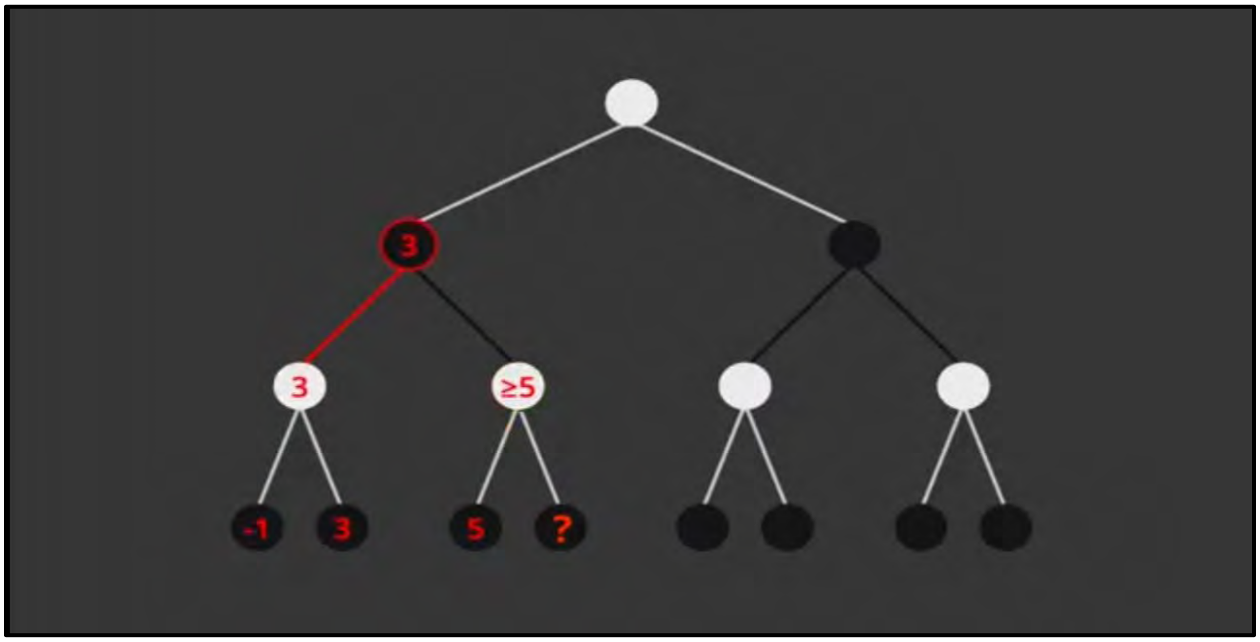


Figure 2.22: Alpha-Beta Pruning Search Tree Phase 5

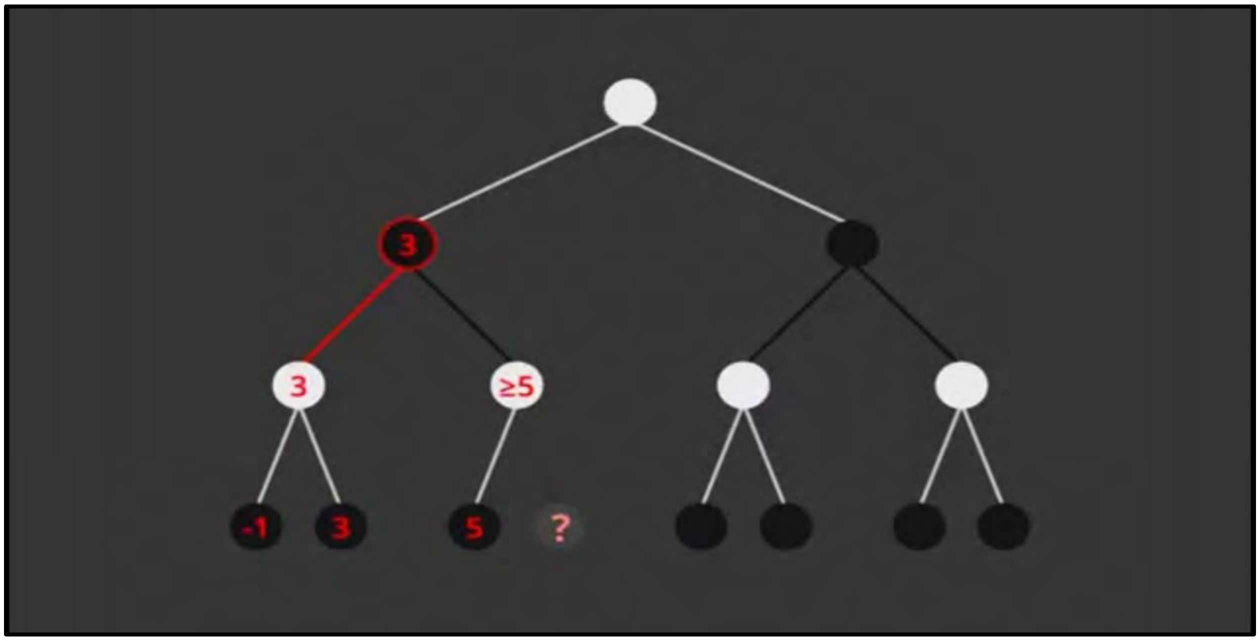


Figure 2.23: Alpha-Beta Pruning Search Tree Phase 6

The things now continue as normal again for a few steps until we get to the position where black decides to make the move. Black should play in this position whichever of the two moves leads to the lowest evaluation.

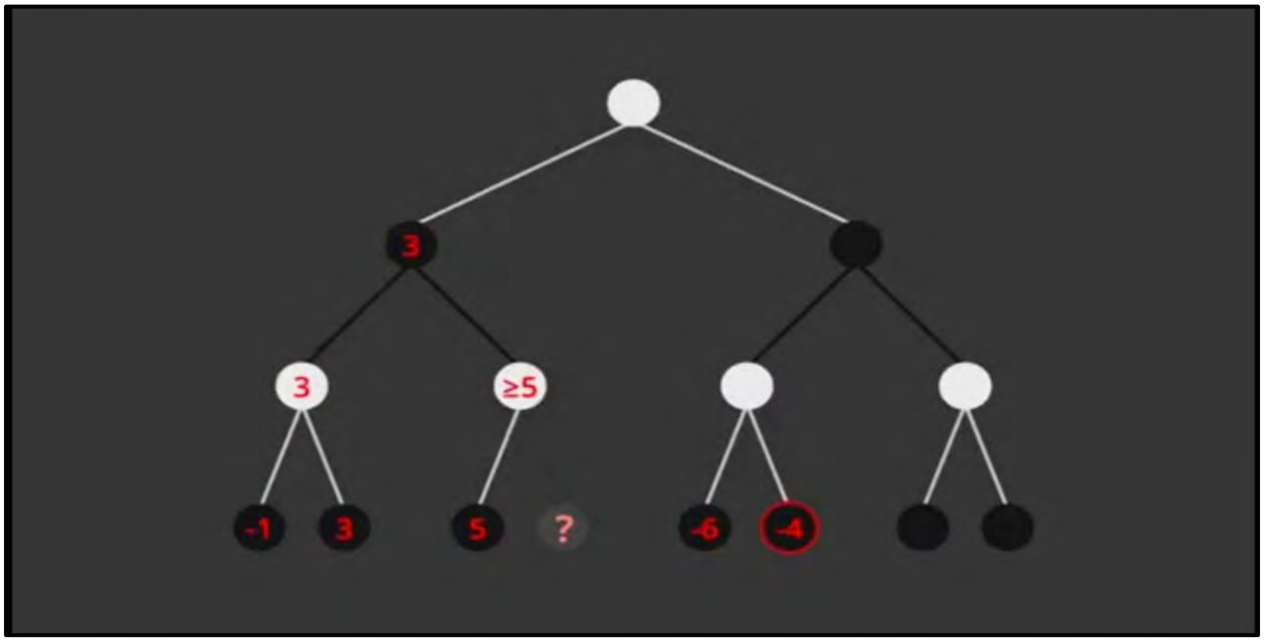


Figure 2.24: Alpha-Beta Pruning Search Tree Phase 7

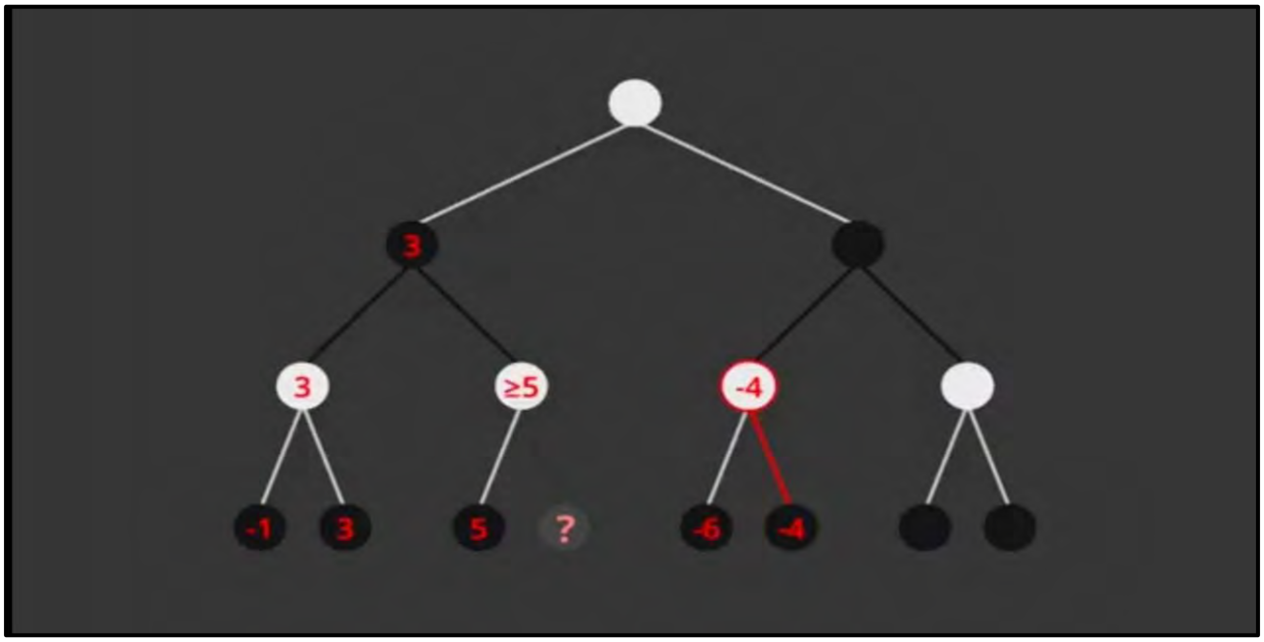


Figure 2.25: Alpha-Beta Pruning Search Tree Phase 8

So, we know the evaluation here is going to be less than or equal to -4. we can now be sure that white won't go down this branch because it has a better option available to it and so we can prune these positions.

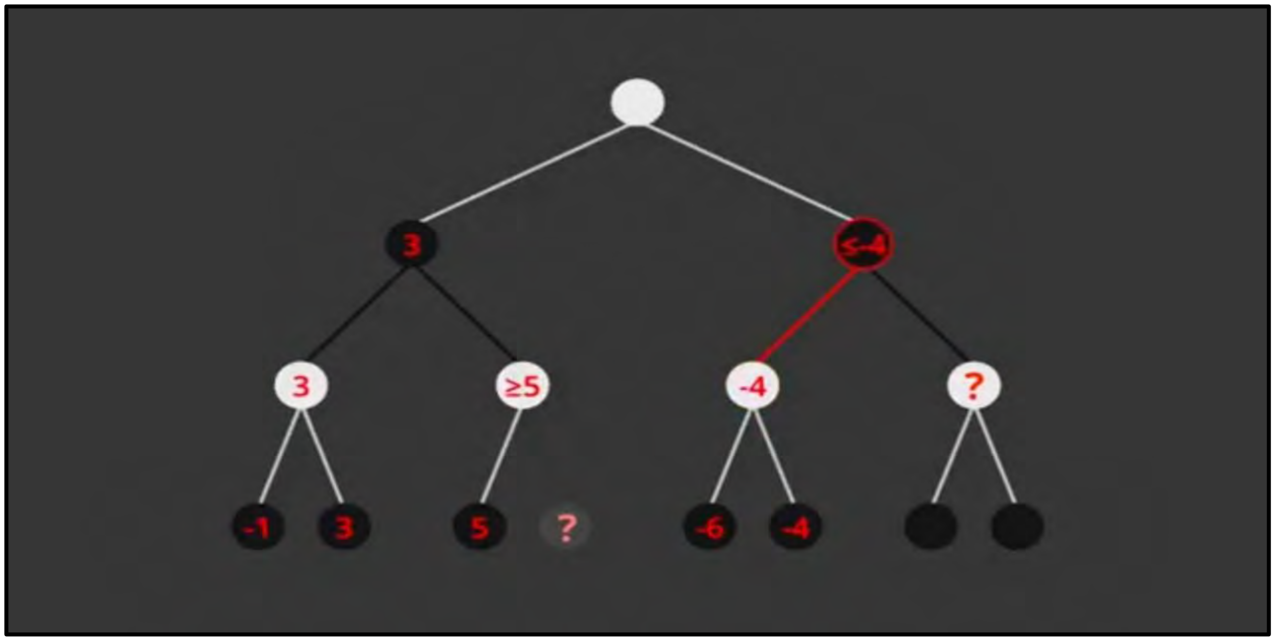


Figure 2.26: Alpha-Beta Pruning Search Tree Phase 9

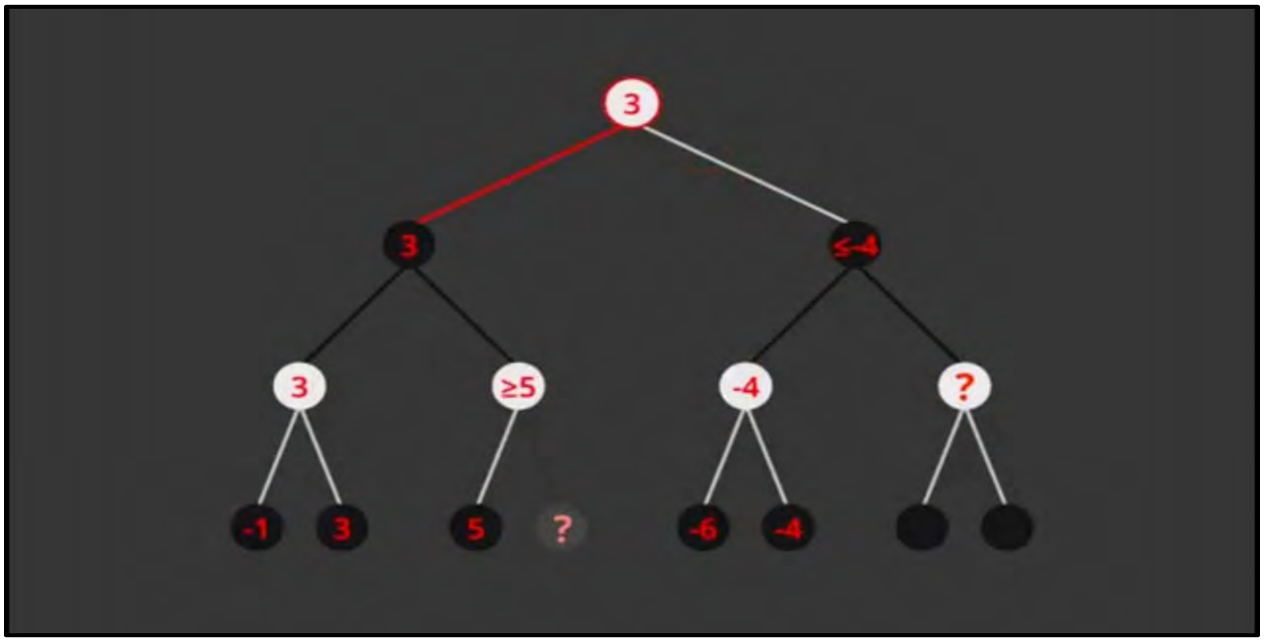


Figure 2.27: Alpha-Beta Pruning Search Tree Phase 10

The result of the search is exactly the same as that of Minimax Algorithm, but with the advantage of saving some time by not considering the positions that can't affect the outcome [6].

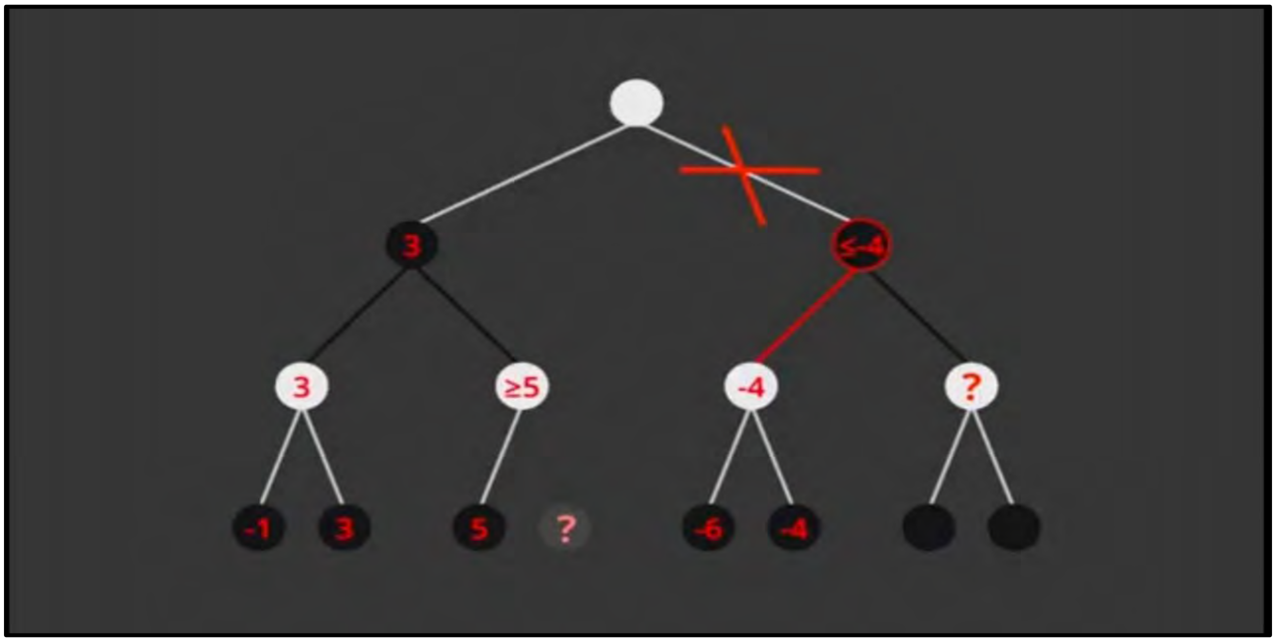


Figure 2.28: Alpha-Beta Pruning Search Tree Phase 11

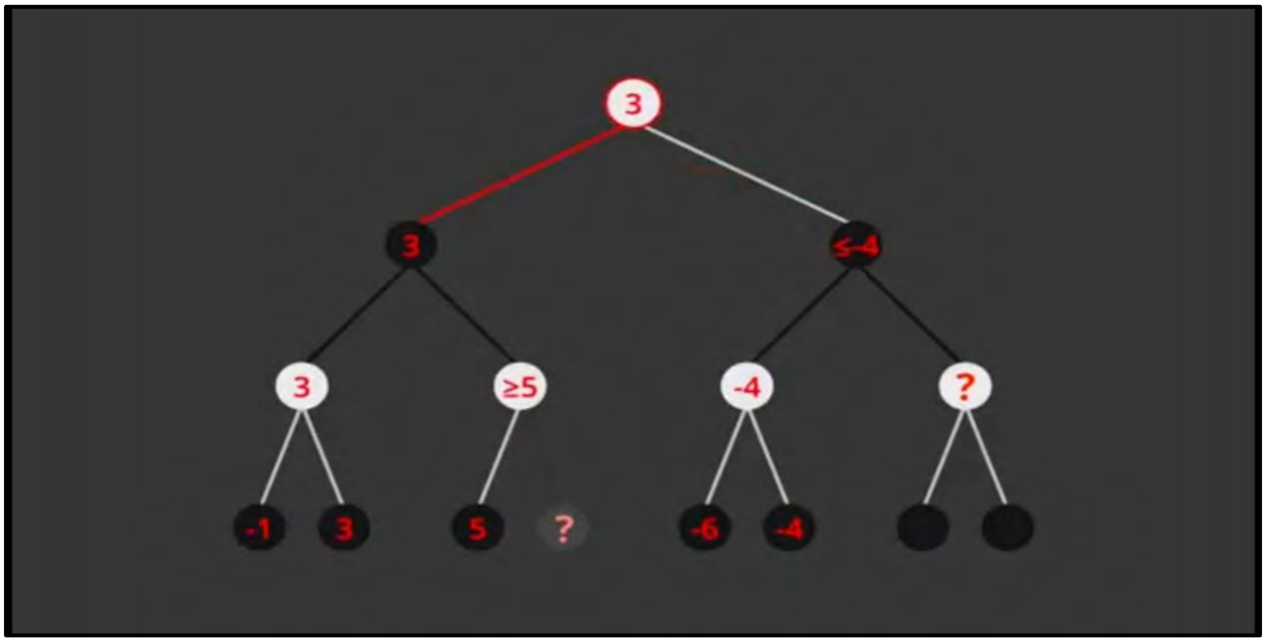


Figure 2.29: Alpha-Beta Pruning Search Tree Phase 12

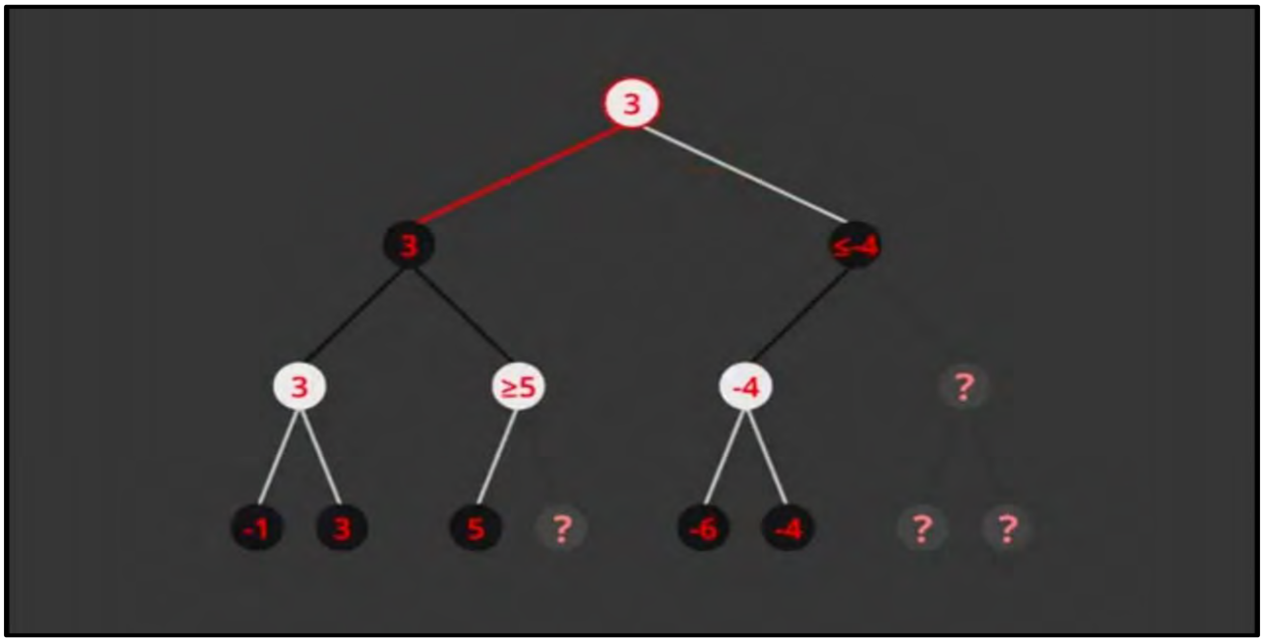


Figure 2.30: Alpha-Beta Pruning Search Tree Phase 13

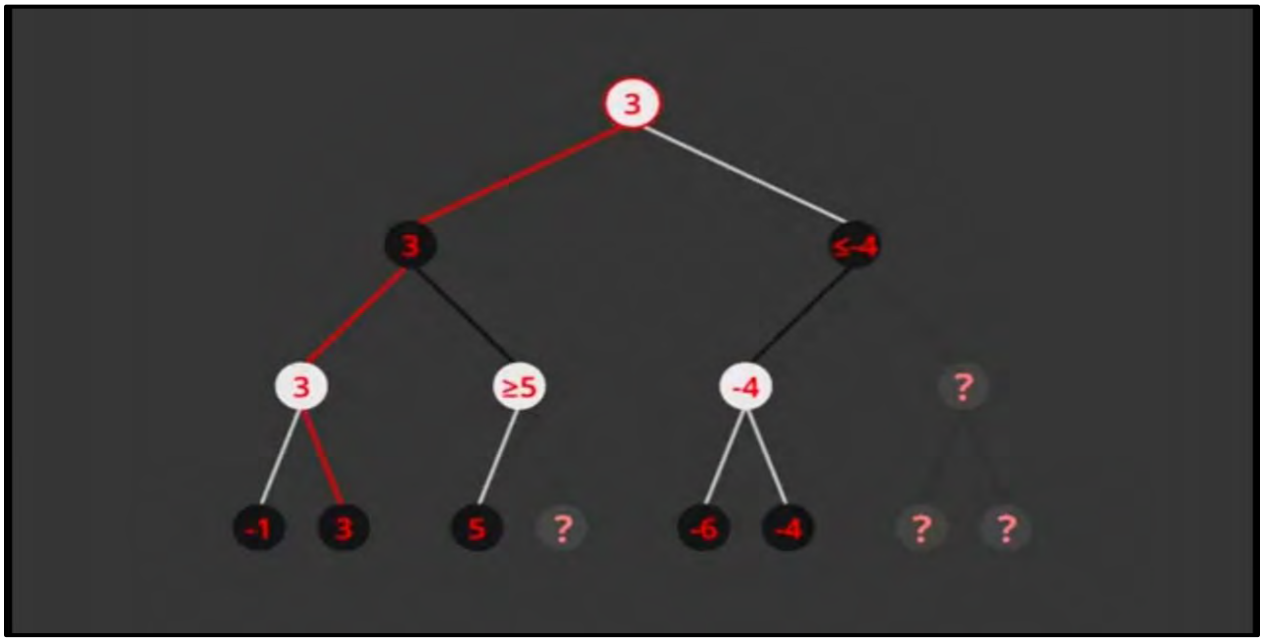


Figure 2.31: Alpha-Beta Pruning Search Tree Phase 14

2.5 Othello Gameplay

The gameplay is demonstrated using minimax and alpha-beta pruning, in order to understand the real time circumstances that would occur using these algorithms:

🟦	White disc placement (good)	🟦	White disc placement (better)
🟩	Black disc placement (good)	🟫	Black disc placement (better)
🟥	Alpha-Beta Pruning (best)		

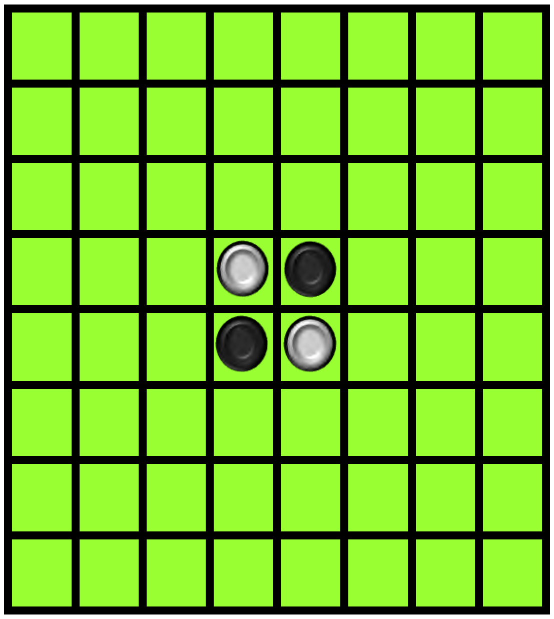


Figure 2.32: Game Phase-1

1. The Game Initiation:

4 discs hold their places same discs are placed diagonal to one another, while opposites are placed in a face to face manner.

2. Black Goes First:

The 🟩 positions on the board, show the placement possibilities of the black discs on the board.

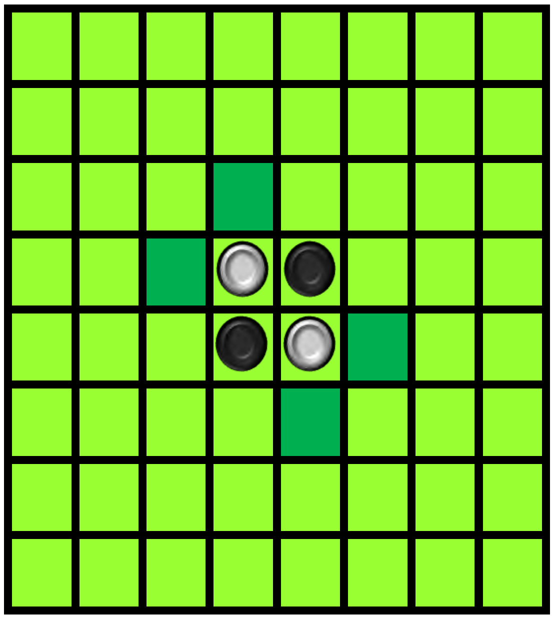


Figure 2.33: Game Phase-2

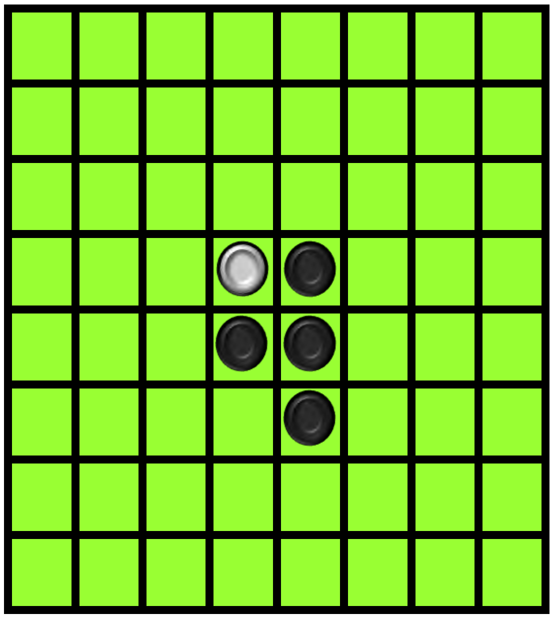


Figure 2.34: Game Phase-3

3. First Move Drawn:

The player to go first will arbitrarily draw his first move and in counter to that the second player or the system will draw the second move, which won't be enough to determine who is going to win. It is a long journey ahead for that to be determined.

4. White Makes the Move:

The 🟦 positions on the board, show the placement possibilities of the white discs on the board.

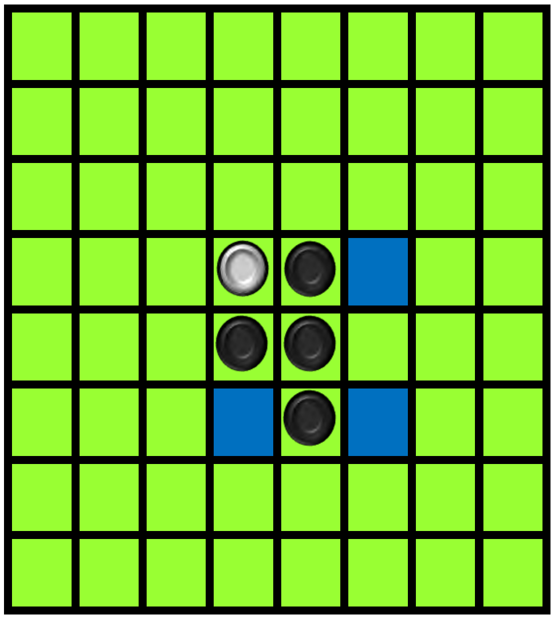


Figure 2.35: Game Phase-4

5. White Makes the Move:

The game proceeds further, but still unpredictable for who gets to win. Either sides progress with equal pace without the any sort of minimax function or Alpha-Beta pruning.

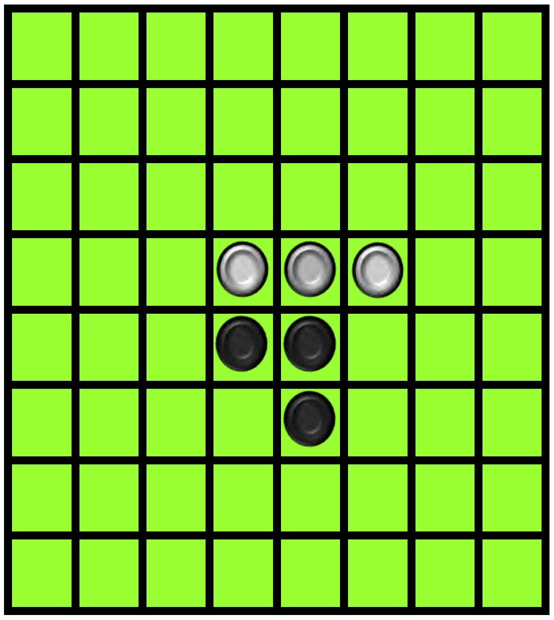


Figure 2.36: Game Phase-5

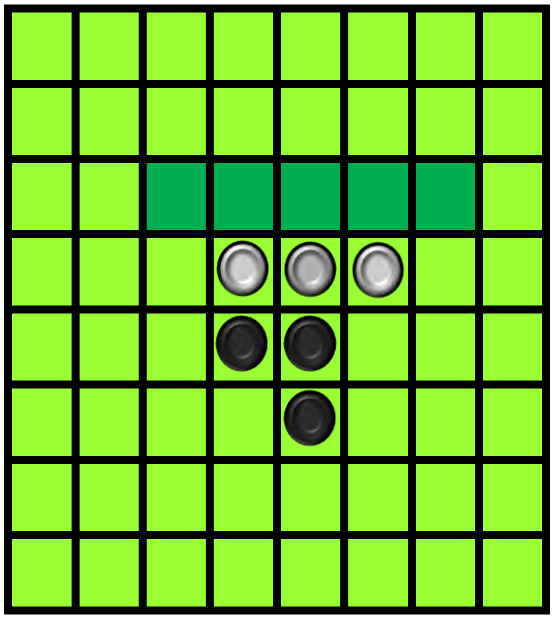


Figure 2.37: Game Phase-6

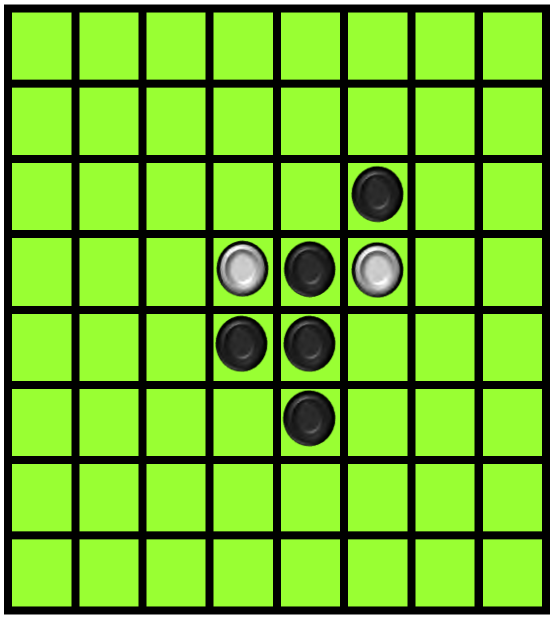


Figure 2.38: Game Phase-7

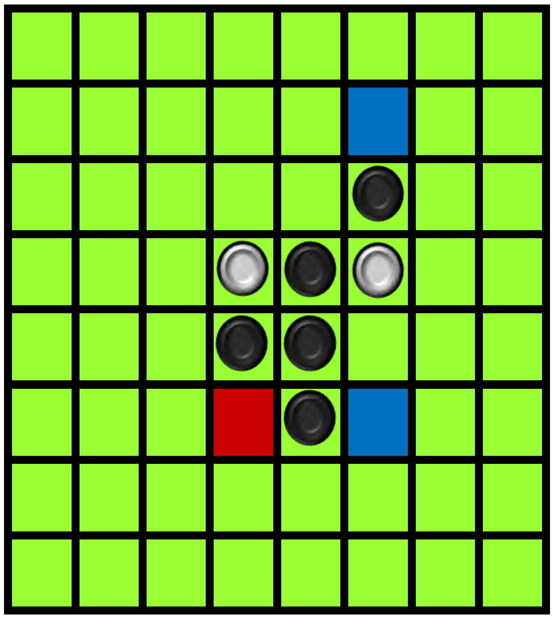


Figure 2.39: Game Phase-8

6. Alpha-Beta Pruning Encountered:

Compared to the rest of the moves to be made, the 🟥 placement position is a move that is must to be taken. The system can only hint it out for the player, but it is up to the player whether he/she gets to choose the move and place the disc on the position that leads to the one step ahead of beating the other player/system.

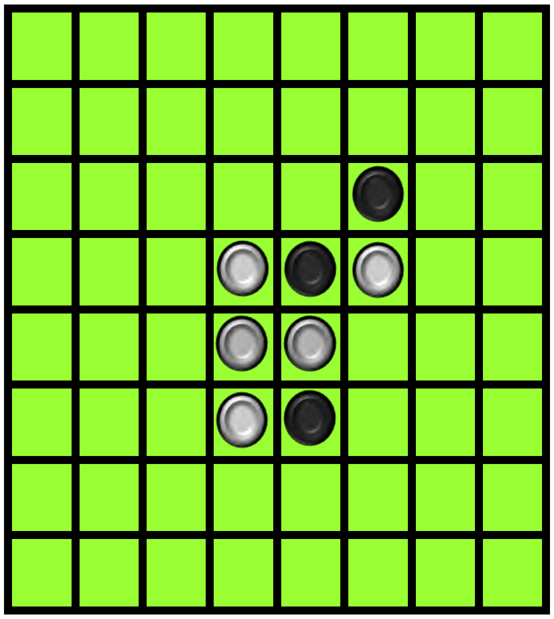


Figure 2.40: Game Phase-9

7. Sticky Situation Equals Go for Your Gut:

Two places on the board correspond to the Alpha-Beta pruning algorithm by cutting down the rest of the moves as shown in Figure 42: Game Phase-10. Here the choice of which best move to choose comes. The only way to find out is to draw all the game trees after each move has been executed on both sides, follow the path that would lead to your victory and prune the rest at each step as you move one step ahead.

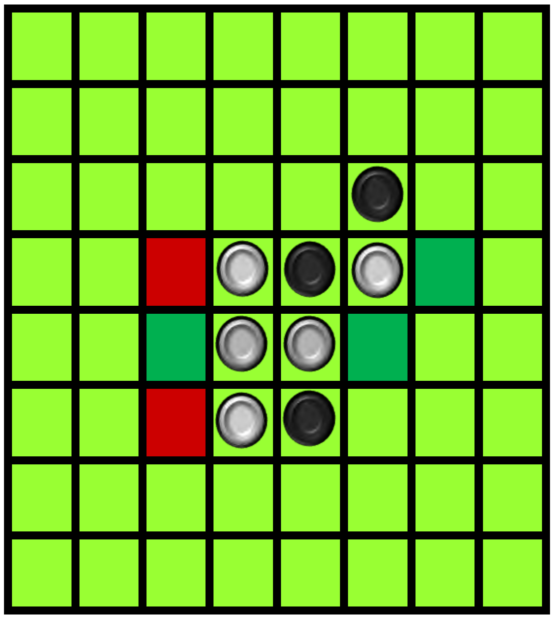


Figure 2.41: Game Phase-10

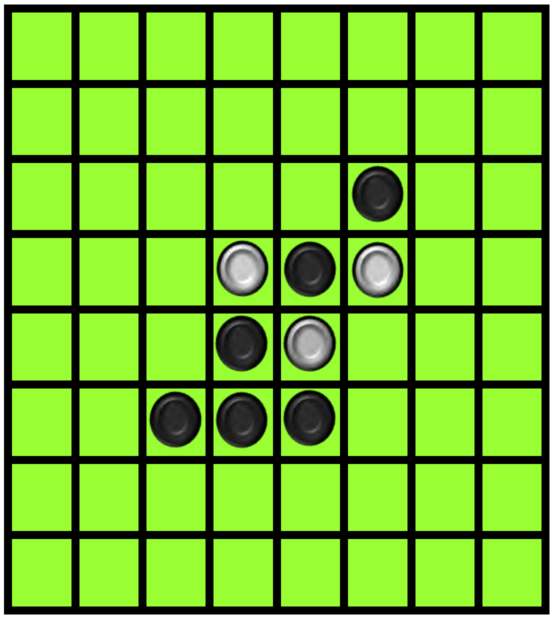


Figure 2.42: Game Phase-11

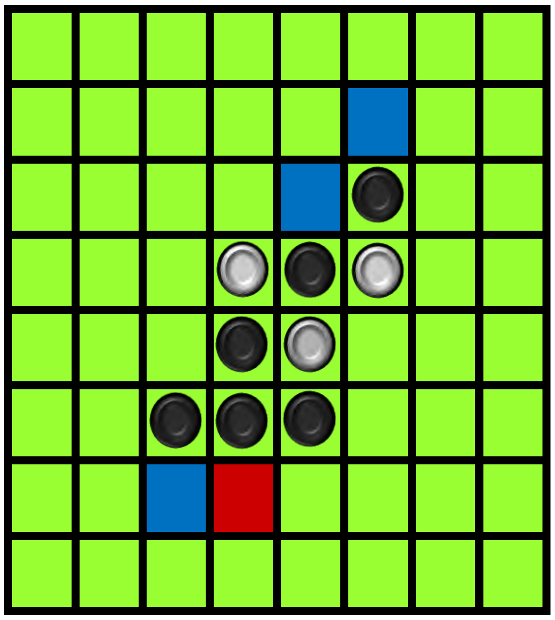


Figure 2.43: Game Phase-12

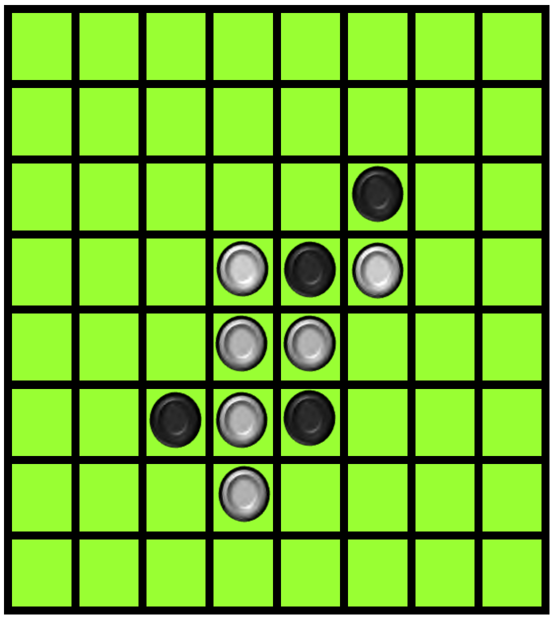


Figure 2.44: Game Phase-13

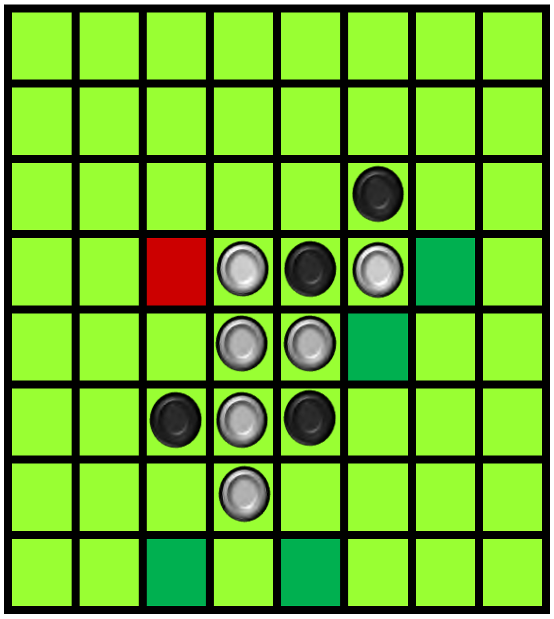


Figure 2.45: Game Phase-14

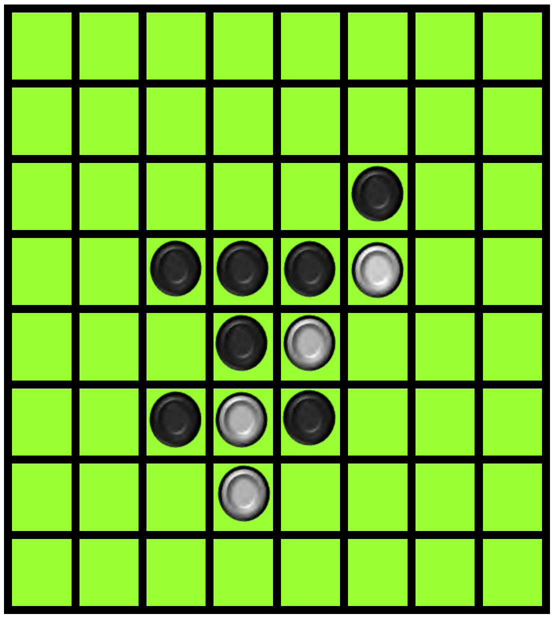


Figure 2.46: Game Phase-15

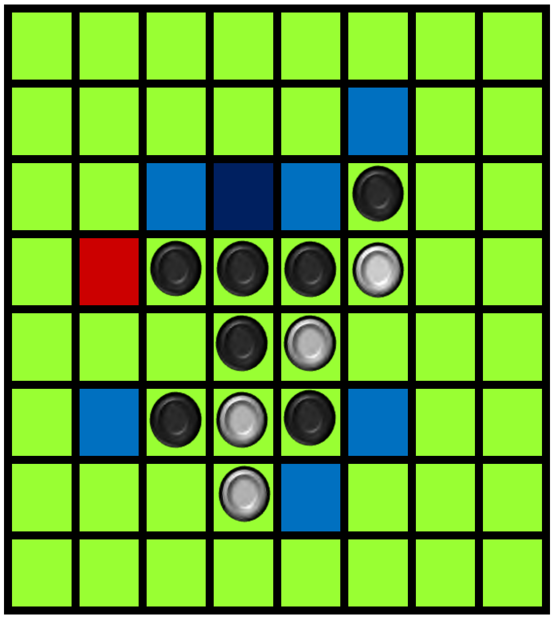


Figure 2.47: Game Phase-16

8. Good, Better or Best:

In the figure, it is white's turn to make a move, but as you can see the player has to choose the 🟥 placement position as the next move instead of 🟦 and ■ because it leads to a possibility of scoring more than the rest of the moves. The same goes for 🟩 and 🟫

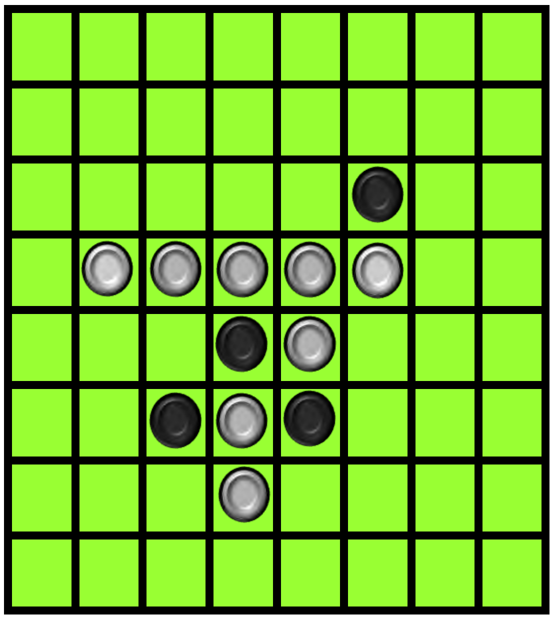


Figure 2.48: Game Phase-17

9. Conclusion:

Alpha Beta Pruning helps in choosing the best possible move from a set of available moves. The previous figure is a brief indication of why the red placement position was chosen from the rest of the moves.

Chapter 3: SOFTWARE REQUIREMENT SPECIFICATION (SRS)

3. Software Requirement Specification (SRS)

3.1 Introduction

This chapter is mainly associated with the scope description and overview of everything included in this SRS document. Other than that, the purpose of the document is described, and a list of abbreviations and definitions are provided.

3.1.1 Purpose

The purpose of this document is to give a detailed description of the requirements for "Othello". This document covers the functional as well non-functional requirements of the system that happen to be the essential features of the system that are fixed, known, and agreed to be delivered. In addition to the basic requirements, it will also describe the external interface requirements.

3.1.2 Scope

The scope of this project is to provide a detailed description of project management plan for Desktop based board game. Othello is intended to provide entertainment to users through many interesting features. It is a two-player game implemented as a desktop application that is having an additional feature to compete against the computer itself. It is mainly for entertainment purposes. The goal of this project is to provide a leisurely, but challenging gaming experience that will grab and hold their attention and interest as well as intellectually stimulate them.

Major Inputs:

- An individual player's move.
- Theme selection.
- Level selection i.e. Easy, Medium and Hard.
- Hint demand.

Major Outputs:

- Computer's move.
- Hints.

Functionalities:

- Prediction of the best possible move during both cases when we want a hint and when the computer decides its own move against the player.
- Applying the best possible move when playing against the computer.

Constraints:

- Desktop based.
- Completion on the project before the end of 8th semester.

3.2 Objective

To entertain and provide a learning edge for amateur players with the three different levels where they can earn diamonds by beating the opponent and spend them on customizable.

3.3 Definitions, Acronyms and Abbreviations

Table 3.1: Definitions, Acronyms and Abbreviations

Term	Definition
AI	Artificial Intelligence
Player	Someone who interacts with the application
DB Handler	User who has specific privileges to handle and manage system DB
DB	Database
Stakeholder	Any person who has interaction or has indirect interest with the system and is not a developer
UC	Use Case
SSD	System Sequence Diagram
SD	Sequence Diagram

3.4 Overview

The remainder of this document includes two sections. The second one provides an overview of the system functionality and system interaction with other systems. This section also introduces different types of stakeholders and their interaction with the system. Further, the section also mentions the system constraints and assumptions about the product.

The third section provides the requirements specification in detailed terms and a description of the different system interfaces.

3.5 Overall Description

This section deals with the overview of the whole application. The application will be explained in its context to discuss the system interaction with users (players). It will also describe what type of stakeholders that will use the system and what functionality is available for each type. At last, the constraints and assumptions for the system will be presented.

3.5.1 Product Perspective

This product is independent and totally self-contained.

3.5.2 Product Function

Following are the major functions of the application:

- The system predicts the best possible moves during both cases; when we want the player wants a hint and when the system decides its own move against the player.
- The player can choose a specific mode of either playing against another player or playing against the computer itself.
- The player can select a particular theme to make the best of what the game has to offer.
- The player can always shift from the traditional, but difficult 8x8 board gameplay experience to the easy 4x4 novice board and 6x6 mediocre board gameplay experience, so that players of every age can have fun playing the game.

3.6 Users

Each player counts as eligible to use the application.

3.7 Assumptions and Dependencies

The player (s) is/are assumed to be aware of the basic know-hows of how to use a PC.

3.8 Software System Attributes

3.8.1 Reliability

System should be reliable. The system should be able to work properly all the time, i.e., to the extent to which it works as and when needed. The system should give proper response against every leak performed by insider user.

3.8.2 Availability

System should be available to user (player (s)) at any time.

3.8.3 Security

When playing the game, there shouldn't be any threats from other softwares/applications. The system should stop the interference of any external application's activity that will disrupt the application from working properly.

3.8.4 Portability

Portable with all Windows based PC's, but not any other OS based PC's or devices.

3.8.5 Performance

System should be able to deal with two players at most. Loading time shouldn't affect the interest of the player (s) in the application.

3.9 Use Cases List

- Play
    - Play Against Player
    - Play Against Computer
- Select Level
- Customize
- Get Hint
- Quit Game

3.10Use case Diagram

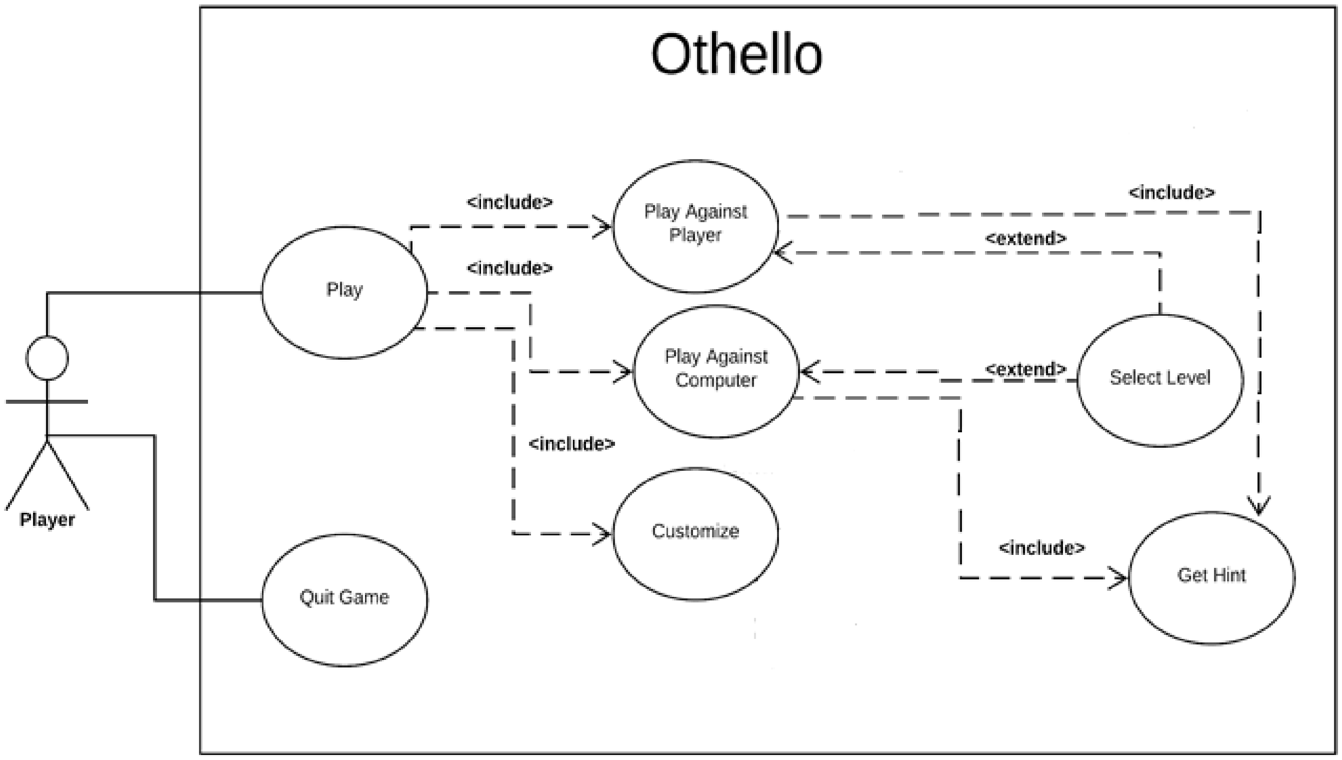


Figure 3.1: Use Case Diagram

3.11 Use case Description

3.11.1 UC-1: Play

Table 3.2: UC-1: Play

ID	UC-1
Name	Play
Primary Actors	Player
Stakeholders	Developer of the Game
Description	The use case, "Play" is performed by the player after which the computer will open the screen displaying the sub-options versus-player/versus-computer, board-size (difficulty level), and game-mode (game theme selection).
Pre-condition	The application is executed, and the home interface of the game is displayed by the system.
Post-condition	The system opens the interface leading to the options of versus-player/versus-computer, board-size (difficulty level), and game-mode (game theme selection).
Main Success Scenario	1. Player opens the application/game. 2. The system starts running the application on the backend. 3. Player starts playing the game. 4. System opens the next interface that displays the sub-classified options i.e. versus-player and versus-computer. 5. Player selects the board-size (difficulty-level), versus-player/versus-computer, and the game-mode (game theme selection).
Alternate Scenarios or Extensions	1a. The application fails to open. 1. The player is prompt to restart the system. 2. The player is prompt to increase the size of the RAM of the system, a very costly option to consider.

	4a. The system fails to open the next interface. 1. The player is prompt to restart the game. 2. The player is prompt to report to the developer of the game, regarding the issue encountered.
Frequency	Continuous at the rate at which the game is to be played as per daily basis.

3.11.2 UC-2: Play Against Player

Table 3.3: UC-2: Play Against Player

ID	UC-2
Name	Play Against Player
Primary Actors	Player
Stakeholders	Developer of the Game
Description	The use case, “Play Against Player” is performed by the player after which the system will set the game to be played against another player.
Pre-condition	The player hits the versus-player button.
Post-condition	The game between two players is initiated.
Main Success Scenario	1. System asks the player to choose to turn. 2. Players choose their turns. 3. The system sets the game to be played against the players.
Alternate Scenarios or Extensions	1a. The system fails to ask the player to choose to turn. 1. The player is prompt to restart the game. 2. The player is prompt to report to the developer of the game, regarding the issue encountered. 2a. The players can’t choose their turns. 1. The players are prompt to restart the game. 2. The players are prompt to report to the developer of the game, regarding the issue encountered. 3a. The system fails to set the game to be played against another player.

	1. The players are prompt to restart the game. 2. The players are prompt to report to the developer of the game, regarding the issue encountered.
Frequency	Continuous at the rate at which the game is to be played as per daily basis.

3.11.3 UC-3: Play Against Computer

Table 3.4: UC-3: Play Against Computer

ID	UC-3
Name	Play Against Computer
Primary Actors	Player
Stakeholders	Developer of the Game
Description	The use case, "Play Against Computer" is performed by the player after which the system will set the game to be played against the computer.
Pre-condition	The player hits the versus-computer button.
Post-condition	The system will open the screen displaying the sub-options pick-turn and start-the-match.
Main Success Scenario	1. The system will set the game to be played against the computer. 2. The system will let the player to make the first move; owning the black discs by default.
Alternate Scenarios or Extensions	1a. The system fails to set the game to be played against the computer. 1. The player is prompt to restart the game. 2. The player is prompt to report to the developer of the game, regarding the issue encountered. 2a. The system doesn't allow the player, owning the black discs, to make the first move
Frequency	Continuous at the rate at which the game is to be played as per daily basis.

3.11.4 UC-4: Select Level

Table 3.5: UC-4: Select Level

ID	UC-4
Name	Select Level
Primary Actors	Player
Stakeholders	Developer of the Game
Description	The use case, “Select Level” is performed by the player after which the system will set the board size as 4x4, 6x6 or 8x8 depending on the difficulty level which the player has selected.
Pre-condition	The player has either selected the versus-player or versus-against-computer option.
Post-condition	The system either opens the versus-player mode or versus-computer mode, depending on the player’s choice.
Main Success Scenario	1. The system provides three difficulty levels to the player i.e. Easy, Medium and Hard on the home screen of the application. 2. Player selects his/her desired difficulty level to be played.
Alternate Scenario or Extensions	1a. The fails to display the difficulty levels. 1. The player is prompt to report to the developer of the game, regarding the issue encountered. 2a. The player is unable to select his/her desired difficulty level. 1. The player is prompt to restart the game. 2. The player is prompt to report to the developer of the game, regarding the issue encountered.
Frequency	Continuous at the rate at which the game is to be played as per daily basis.

3.11.5 UC-5: Customize

Table 3.6: UC-5: Customize

ID	UC-5
Name	Customize
Primary Actors	Player
Stakeholders	Developer of the Game
Description	The use case, "Customize" is performed by the player after which the system will change the theme of the game according to the player's choice from one the three themes available.
Pre-condition	The player has either selected the versus-player or versus-computer option.
Post-condition	The game between the players or the player vs computer is initiated.
Main Success Scenario	1. Player selects the desired choice of theme for the game. 2. The system opens the customized game mode to be played with either against two players or against the player and the computer.
Alternate Scenarios or Extensions	2a. The system fails to open the customized game mode screen. 1. The player is prompt to report to the developer of the game, regarding the issue encountered.
Frequency	Continuous at the rate at which the game is to be played as per daily basis.

3.11.6 UC-6: Get Hint

Table 3.7: UC-6: Get Hint

ID	UC-6
Name	Get Hint
Primary Actors	Player
Stakeholders	-

Description	The use case, "Get Hint" is performed by the player after which the system will make the best space prominent on the board where the player will place his/her disc.
Pre-condition	The player hits the hint button, when found stuck or not during the game.
Post-condition	The system provides the player demanding the hint with one of the best possibilities to get out of the stuck situation.
Main Success Scenarios	1. Player demands the system for hints. 2. The system checks the adjacent discs of the opposite colored discs when the player (s) demands for hints. 3. System makes use of the algorithm that checks the adjacency and generates hints. 4. The high lightened spaces indicate the adjacent portion on the board on which the opposite colored discs are to be placed. 5. The system directs the player to place the next disc on one of the possibly most appropriate allowed spaces.
Alternate Scenario or Extensions	1a. No more possible hints available. 1. The system displays a pop-up dialogue box telling the player that no more possibilities are available to get out of the stuck situation. 1b. Hints-to-be-demanded has reached its limit. 1. The system displays a pop-up dialogue box telling the player that the player's demand for hints has run out, so the game can no longer provide hints.
Frequency	Continuous at the rate at which the game is to be played as per daily basis.

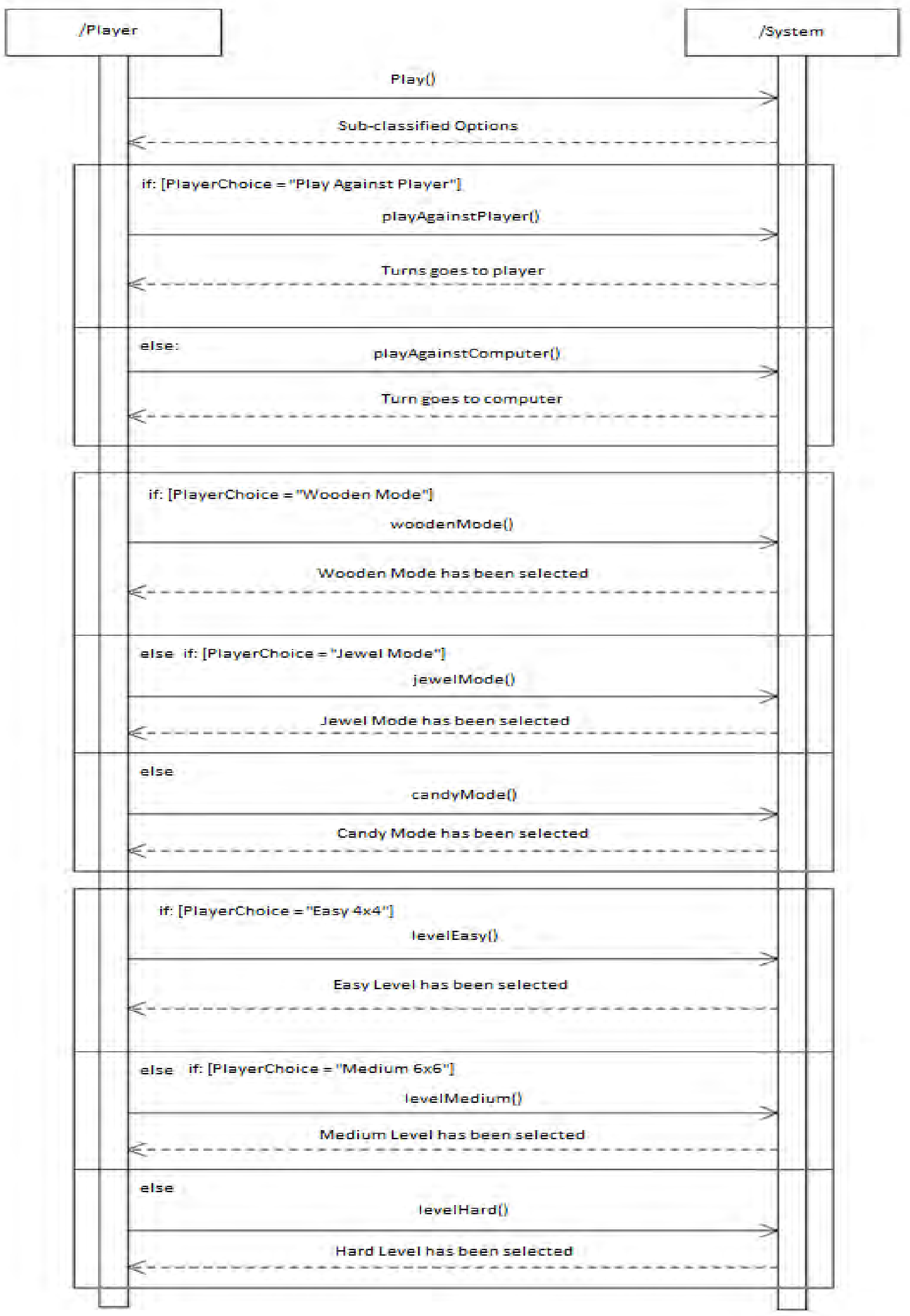
3.11.7UC-7: Quit Game

Table 3.8: UC-7 Quit Game

ID	UC-7
Name	Quit Game
Primary Actors	Player
Stakeholders	Developer of the Game
Description	The use case, "Quit Game" is performed by the player where the system will exit all the processing of the application any further and will turn to the regular processing of the PC/laptop of the user (player).
Pre-condition	The player selects the quit button.
Post-condition	The system terminates the application and quit the game from further processing.
Main Success Scenarios	1. Player quits the game. 2. The system stops the further processing of the game and exits it.
Alternate Scenario or Extensions	2a. The system is implicitly unable to exit the processing of the game. 1. Player presses Alt + Ctrl + Delete to open Task Manager to manually exit the application from further processing. 2. If the issue happens time and gain, then the player is prompt to report to the developer of the game, regarding the issue encountered.
Frequency	Continuous at the rate at which the game is to be played as per daily basis.

3.12 System Sequence Diagrams (SSDs)

3.12.1 Play SSD



/Player
/System
Play()
Sub-classified Options
if: [PlayerChoice = "Play Against Player"]
playAgainstPlayer()
Turns goes to player
else:
playAgainstComputer()
Turn goes to computer
if: [PlayerChoice = "Wooden Mode"]
woodenMode()
Wooden Mode has been selected
else if: [PlayerChoice = "Jewel Mode"]
jewelMode()
Jewel Mode has been selected
else
candyMode()
Candy Mode has been selected
if: [PlayerChoice = "Easy 4x4"]
levelEasy()
Easy Level has been selected
else if: [PlayerChoice = "Medium 6x6"]
levelMedium()
Medium Level has been selected
else
levelHard()
Hard Level has been selected

Figure 3.2: Play SSD

3.12.2 Get Hint SSD

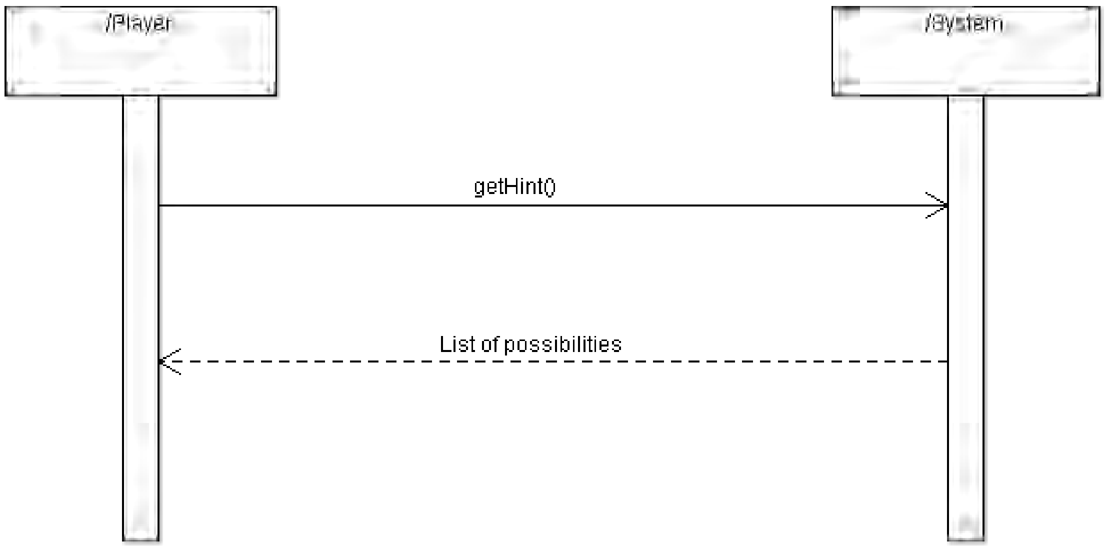


Figure 3.3: Get Hint SSD

3.12.3 Quit Game SSD

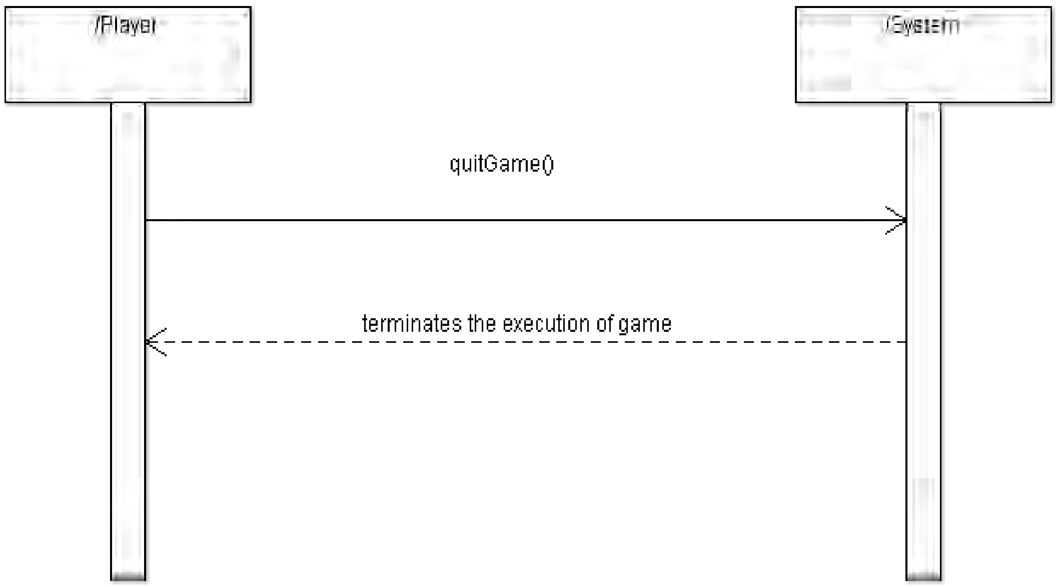


Figure 3.4: Quit Game SSD

3.13Domain Model

A domain model is a visual representation of conceptual classes or real-world objects in a domain of interest. They have also been called conceptual models, domain object models, and analysis object mode.

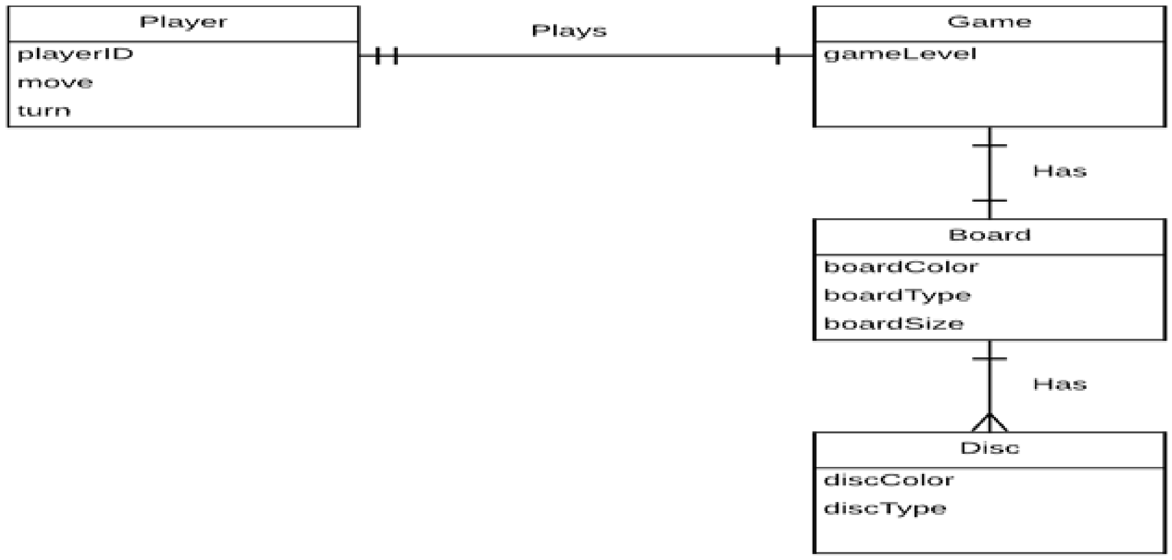


Figure 3.5: Domain Model

Chapter 4: SOFTWARE DESIGN DESCRIPTION (SDD)

4. Software Design Description

4.1 Introduction

Software Design Description (SDD) is a document which provides the complete description of the design of the software to be developed before the actual development. The SDD document describes the system architecture design in detail and provides a complete description of the different components. It also describes that how the different components will communicate with each other. The SDD document also contains the interface design, architecture diagram, sequence diagrams and class diagram.

4.1.1 Requirement Traceability Matrix

Requirement traceability matrix is the matrix which keeps track of the entire requirement throughout the development of the software. The requirement traceability matrix is required during the validation process to check either we have fulfilled the entire requirements or not.

Table 4.2: Requirement Traceability Matrix

Requirement ID	Requirement Name	Test Cases	Sequence Diagrams
UC-1	Play	TC-1	Figure 4.2: Sequence Diagram
UC-2	Play Against Player	TC-2	
UC-3	Play Against Computer	TC-3	
UC-4	Select Level	TC-4	
UC-5	Customize	TC-5	
UC-6	Get Hint	TC-6	
UC-7	Quit Game	TC-7	

4.2 Software Architecture Design

Software architecture design is the process of defining a structured solution that meets all the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. Architectural design is the resolution of the requirements in the design of the software, the hardware and networking, and so forth.

4.3 Chosen System Architecture

2-Tier Architecture is used for the development of the MRA. The top-most layer is the Presentation layer. And the second layer is the Application layer which contains the business logic. We have use 2-Tier architecture instead of 3-Tier architecture because our application won't need a database.

4.3.1 Presentation Layer

Presentation tier is the top most tier, which can be accessed by the player (s) directly. This tier allows the player (s) to interact with it. And this tier will also present the result of actions/moves made to the player (s).

4.3.2 Business Layer

Business tier has the core logic of the system just like Alpha-Beta Pruning. This is the tier which performs the actual processing. This tier will also communicate directly with file containing the gems. Business tier contains 6 modules:

1. Player
2. Board
3. Game
4. Disc

- **Player:**
  Player Module will also directly interact with the Game module. This will keep track of the player's game and the entities it interacts with.

- **Game:**
  Game Module is used to reduce the coupling in between Presentation Layer and Business Layer. This module happens to manage the actual gameplay.

- **Board:**
  Board Module will directly interact with the Game Module and with the Player in the Presentation Layer. It manages the size of the board based on what level of game difficulty the player chooses to play, and the board type which shows the association of what kind of board the player has selected to play with, other than the traditional board type. The interaction of the Board Module with the presentation layer depicts the customization property of the board.

- **Discs:**
  Discs Module will directly interact with the Game Module and with the Player in the Presentation Layer. It manages the disc type which shows the association of what kind of disc the player has selected to play with, other than the traditional white and black ones. The interaction of the Disc Module with the presentation layer depicts the customization property of the board.

4.4 Architecture Diagram

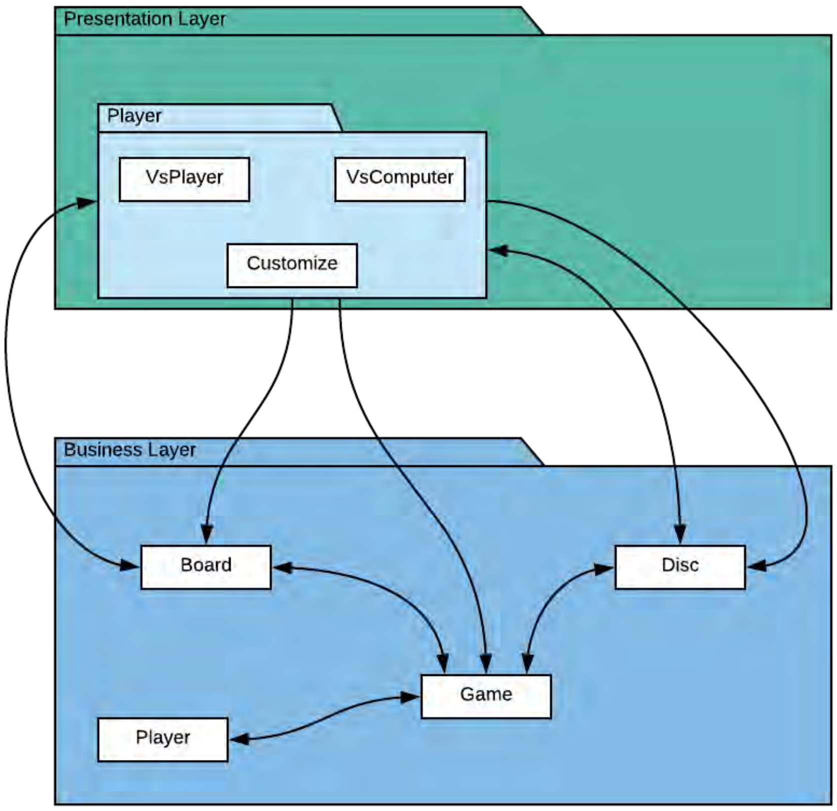


Figure 4.1: Architecture Diagram

4.5 Objects and Actions

A Sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are developed against the use cases.

4.5.1 Sequence Diagram

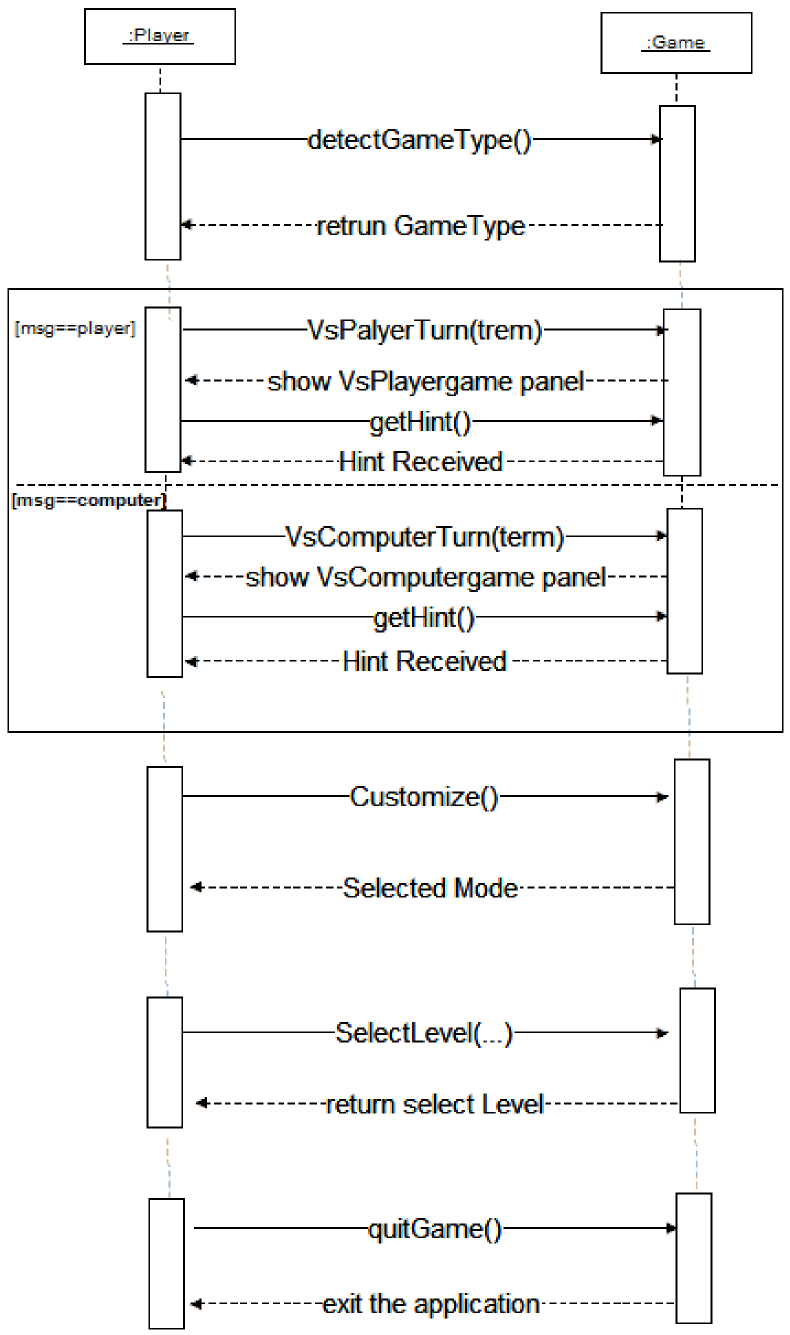


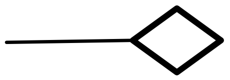
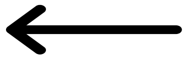
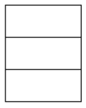
Figure 4.2: Sequence Diagram

4.6 Class Diagram

Class diagrams depict the software classes and their relationships. This diagram defines individual classes along with their attributes, types of the attributes, and operations, associations between classes and navigability (direction of association) that define attribute visibility, and also define non-attribute visibility.

1. Player
2. Game
3. Board
4. Disc

Table 4.2: Notations and their meaning

Notation	Meaning
	**Aggregation:** It is an association that represents a part-whole or part-of relationship.
	**Association:** It is used when one object wants another object to perform a service for it.
	**Class:** In UML, shows architecture and features of the designed system.

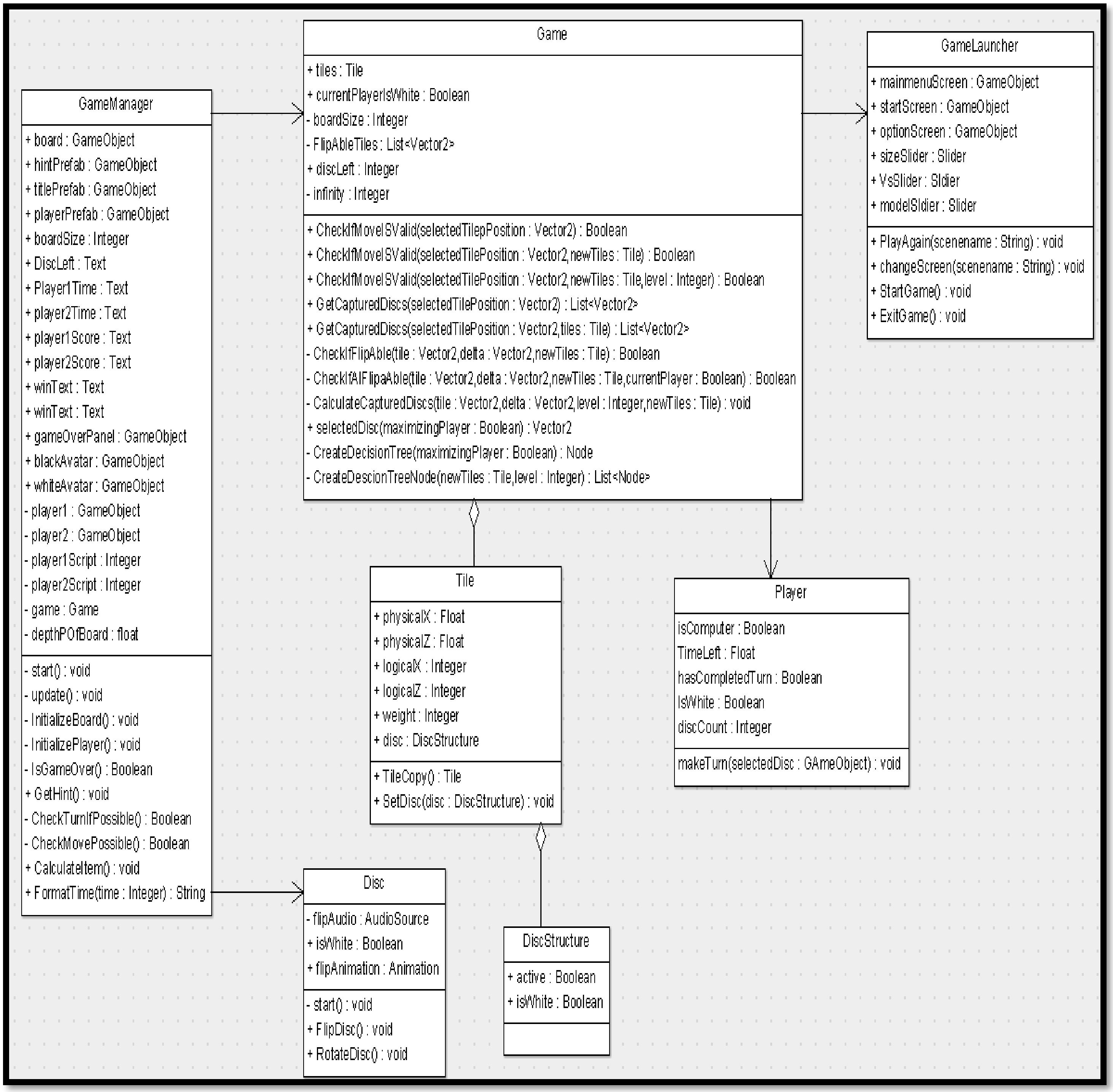
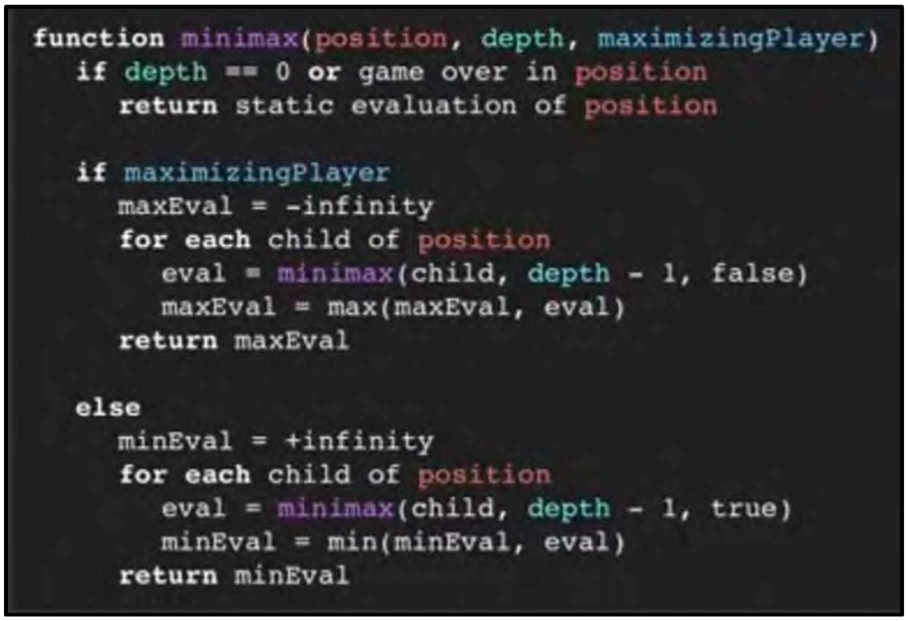

Game
+ tiles : Tile
+ currentPlayerIsWhite : Boolean
- boardSize : Integer
- FlipAbleTiles : List<Vector2>
+ discLeft : Integer
- infinity : Integer
+ CheckIfMoveISValid(selectedTilepPosition : Vector2) : Boolean
+ CheckIfMoveISValid(selectedTilePosition : Vector2,newTiles : Tile) : Boolean
+ CheckIfMoveISValid(selectedTilePosition : Vector2,newTiles : Tile,level : Integer) : Boolean
+ GetCapturedDiscs(selectedTilePosition : Vector2) : List<Vector2>
+ GetCapturedDiscs(selectedTilePosition : Vector2,tiles : Tile) : List<Vector2>
- CheckIfFlipAble(tile : Vector2,delta : Vector2,newTiles : Tile) : Boolean
- CheckIfAIFlipaAble(tile : Vector2,delta : Vector2,newTiles : Tile,currentPlayer : Boolean) : Boolean
- CalculateCapturedDiscs(tile : Vector2,delta : Vector2,level : Integer,newTiles : Tile) : void
+ selectedDisc(maximizingPlayer : Boolean) : Vector2
- CreateDecisionTree(maximizingPlayer : Boolean) : Node
- CreateDescionTreeNode(newTiles : Tile,level : Integer) : List<Node>
GameLauncher
+ mainmenuScreen : GameObject
+ startScreen : GameObject
+ optionScreen : GameObject
+ sizeSlider : Slider
+ VsSlider : Sldier
+ modelSldier : Slider
+ PlayAgain(scenename : String) : void
+ changeScreen(scenename : String) : void
+ StartGame() : void
+ ExitGame() : void
GameManager
+ board : GameObject
+ hintPrefab : GameObject
+ titlePrefab : GameObject
+ playerPrefab : GameObject
+ boardSize : Integer
+ DiscLeft : Text
+ Player1Time : Text
+ player2Time : Text
+ player1Score : Text
+ player2Score : Text
+ winText : Text
+ winText : Text
+ gameOverPanel : GameObject
+ blackAvatar : GameObject
+ whiteAvatar : GameObject
- player1 : GameObject
- player2 : GameObject
- player1Script : Integer
- player2Script : Integer
- game : Game
- depthPOfBoard : float
- start() : void
- update() : void
- InitializeBoard() : void
- InitializePlayer() : void
- IsGameOver() : Boolean
+ GetHint() : void
- CheckTurnIfPossible() : Boolean
- CheckMovePossible() : Boolean
+ CalculateItem() : void
+ FormatTime(time : Integer) : String
Tile
+ physicalX : Float
+ physicalZ : Float
+ logicalX : Integer
+ logicalZ : Integer
+ weight : Integer
+ disc : DiscStructure
+ TileCopy() : Tile
+ SetDisc(disc : DiscStructure) : void
Player
isComputer : Boolean
TimeLeft : Float
hasCompletedTurn : Boolean
IsWhite : Boolean
discCount : Integer
makeTurn(selectedDisc : GAmeObject) : void
Disc
- flipAudio : AudioSource
+ isWhite : Boolean
+ flipAnimation : Animation
- start() : void
+ FlipDisc() : void
+ RotateDisc() : void
DiscStructure
+ active : Boolean
+ isWhite : Boolean

Figure 4.3: Class Diagram

4.7 Pseudo Code of Minimax Algorithm

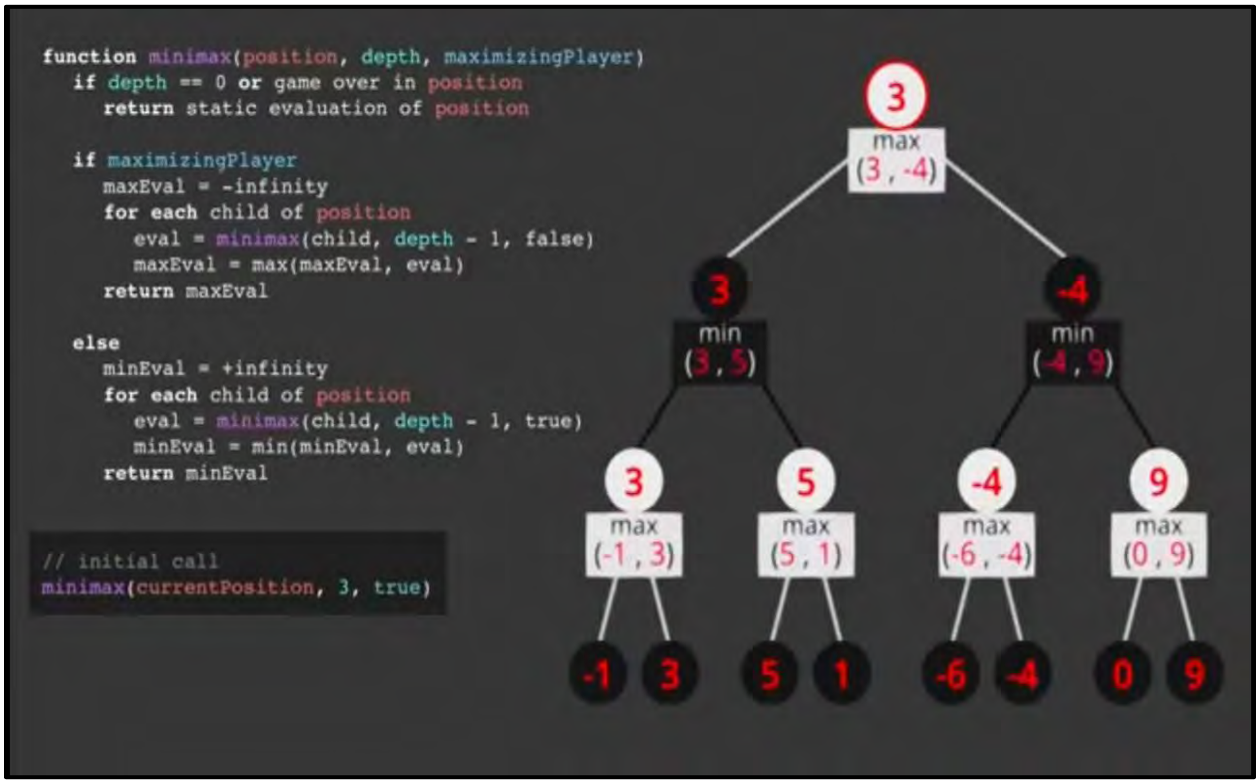
Now, that basic idea has been attained we can see how it can be written in the form of a code.



```
function minimax(position, depth, maximizingPlayer)
  if depth == 0 or game over in position
    return static evaluation of position

  if maximizingPlayer
    maxEval = -infinity
    for each child of position
      eval = minimax(child, depth - 1, false)
      maxEval = max(maxEval, eval)
    return maxEval

  else
    minEval = +infinity
    for each child of position
      eval = minimax(child, depth - 1, true)
      minEval = min(minEval, eval)
    return minEval
```

Figure 4.4: Minimax Algorithm Pseudo Code

We have a function *minimax* which takes the current position, a depth how many moves ahead we want to search, and a variable called *maximizingPlayer*. We begin by checking either the depth is equal to zero or if the game is over from the current position, which is if the case then we return the static evaluation of that position. Otherwise, if it is currently the turn of the maximizing player which of course means that it is white's turn to move, then we want to find the highest evaluation that can be obtained from this position. We create a variable called *maxEval* which is used for maximum evaluation and initialize it with negative infinity. We then loop through all the children of the current position and here mean positions that can reached in a single move. To find the evaluation of each child, we create a variable called *eval* and assign it the recursive call of the minimax function, passing in the child with the depth -1 and false as it will be the other player's turn. We then set *maxEval* equal to whichever is greater between the current evaluation and the evaluation of the child position. Once we have evaluated all the children, we can return the maximum evaluation that we have found. Now, we do essentially the same thing for the minimizing player creating *minEval* and setting its value to positive infinity. For each child position we call minimax recursively, passing in the child, depth as -1 and the *maximizing player* as true. *minEval* is then set to whichever is smaller between the current minimum evaluation and the child evaluation and finally we return the minimum evaluation, *minEval*.

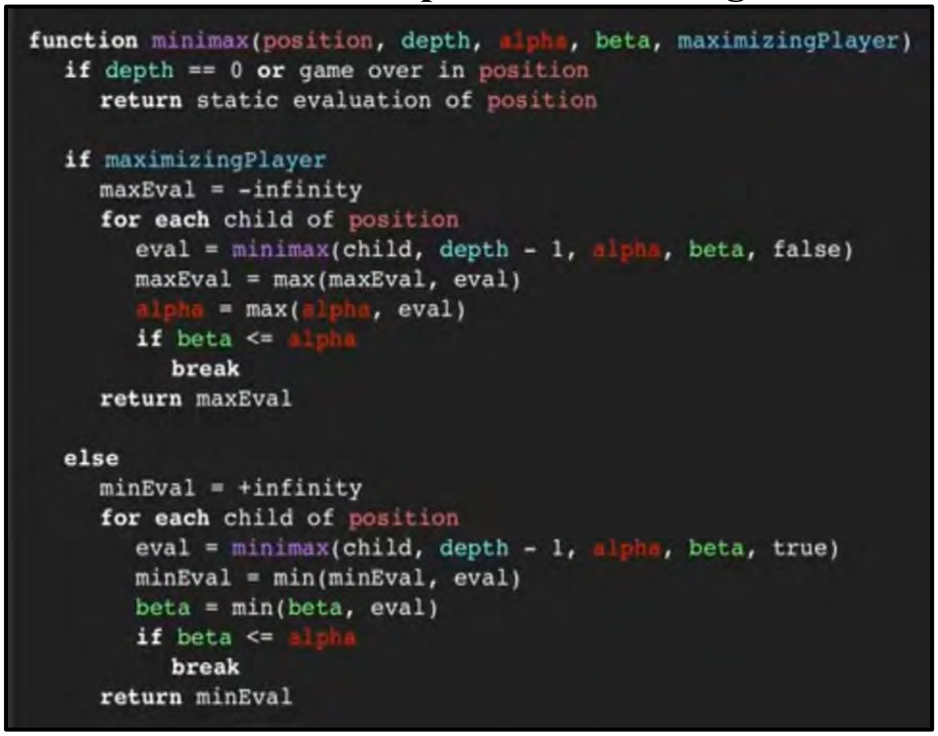


```
function minimax(position, depth, maximizingPlayer)
  if depth == 0 or game over in position
    return static evaluation of position

  if maximizingPlayer
    maxEval = -infinity
    for each child of position
      eval = minimax(child, depth - 1, false)
      maxEval = max(maxEval, eval)
    return maxEval

  else
    minEval = +infinity
    for each child of position
      eval = minimax(child, depth - 1, true)
      minEval = min(minEval, eval)
    return minEval
```

```
// initial call
minimax(currentPosition, 3, true)
```

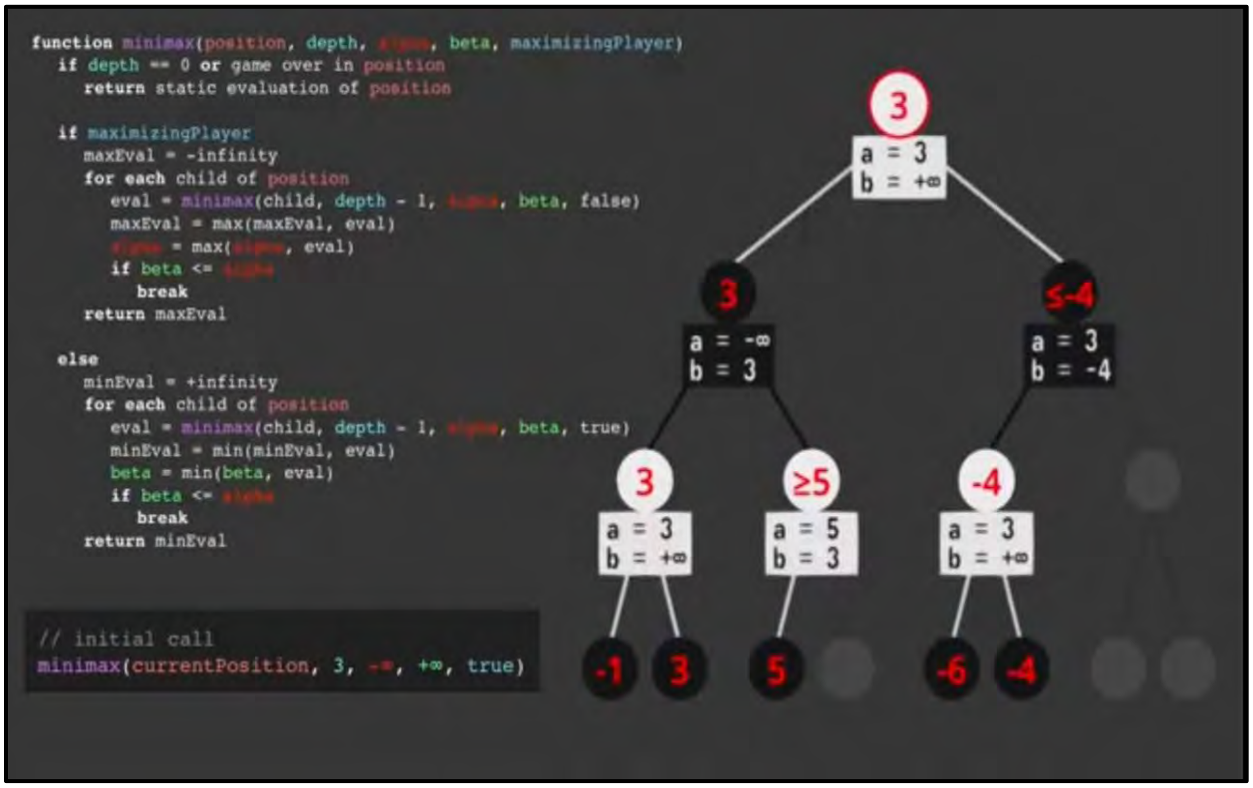
Figure 4.5: Minimax Pseudo Code and Final Search Tree

4.8 Pseudo Code of Alpha-Beta Pruning



```
function minimax(position, depth, alpha, beta, maximizingPlayer)
   if depth == 0 or game over in position
      return static evaluation of position

   if maximizingPlayer
      maxEval = -infinity
      for each child of position
         eval = minimax(child, depth - 1, alpha, beta, false)
         maxEval = max(maxEval, eval)
         alpha = max(alpha, eval)
         if beta <= alpha
            break
      return maxEval

   else
      minEval = +infinity
      for each child of position
         eval = minimax(child, depth - 1, alpha, beta, true)
         minEval = min(minEval, eval)
         beta = min(beta, eval)
         if beta <= alpha
            break
      return minEval
```

Figure 4.6: Alpha-Beta Pruning Pseudo Code



```
function minimax(position, depth, alpha, beta, maximizingPlayer)
   if depth == 0 or game over in position
      return static evaluation of position

   if maximizingPlayer
      maxEval = -infinity
      for each child of position
         eval = minimax(child, depth - 1, alpha, beta, false)
         maxEval = max(maxEval, eval)
         alpha = max(alpha, eval)
         if beta <= alpha
            break
      return maxEval

   else
      minEval = +infinity
      for each child of position
         eval = minimax(child, depth - 1, alpha, beta, true)
         minEval = min(minEval, eval)
         beta = min(beta, eval)
         if beta <= alpha
            break
      return minEval
```

```
// initial call
minimax(currentPosition, 3, -∞, +∞, true)
```

Figure 4.7: Alpha-Beta Pruning Pseudo Code and Final Search Tree

We will start off with the code that was previously used for Minimax Algorithm. We simply add two more parameters i.e. *alpha* and *beta* which will essentially keep track of the best score either side can achieve assuming the best play from the opponent. We update our recursive calls of *minimax* by passing two more values. Now, for the maximizing player we will set *alpha* to whichever is greater between *alpha* and the latest evaluation, *eval* and if *beta* is less than or equal to *alpha*, we will break out of the loop. Similarly, for the minimizing player, we will set *beta* to whichever is smaller between *beta* and the latest evaluation and once again if *beta* is less than or equal to *alpha*, we will break out of the loop.

4.9 User Interface Design

It is the process in which we create the prototype of the screen images. In this section, we draw the interface based on our requirement. And later during the implementation we will use them for the actual development.

4.9.1 Home Interface

The home interface is the representation that the player first gets to see after the application is up and running. It consists of five options, two of which are buttons and three are scrollable i.e. Start, Board Size, Versus, Mode and Exit. Start, as the name describes takes the player to play the game when pressed. Board Size allows the player to select and change the difficulty level according to the size of the board. The bigger the board, the harder it gets. Versus allows the player to choose

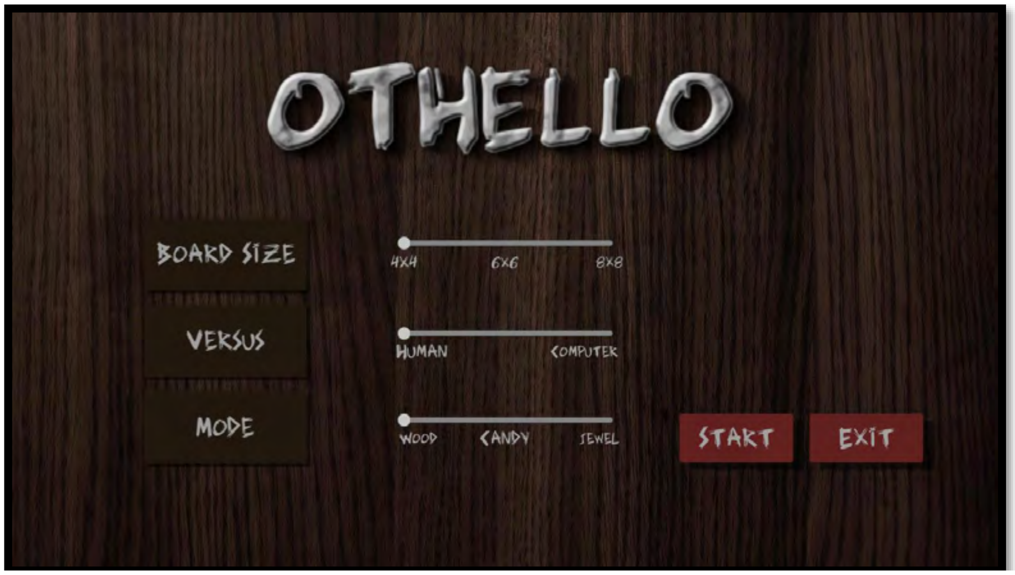


Figure 4.8: Home Interface 1

between the computer and another player to play against. Mode is for changing the theme of the game so that the game environment doesn't bore out the player from the same old environment. Lastly, Exit simply terminates the application from further processing.

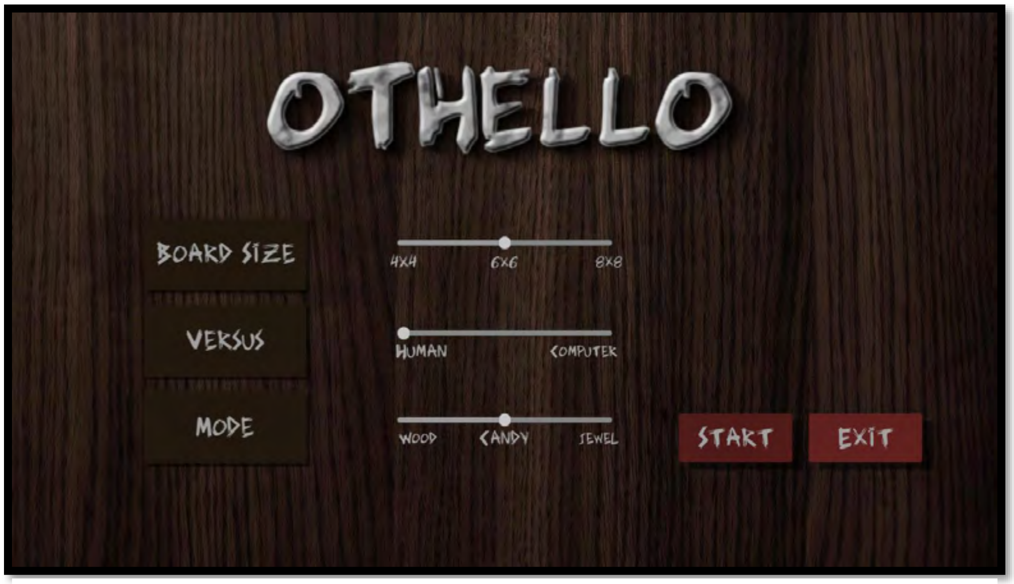


Figure 4.9: Home Interface 2

4.9.2 Game Play Interface

It brings forth the game environment specified by the player by selecting the desired board size

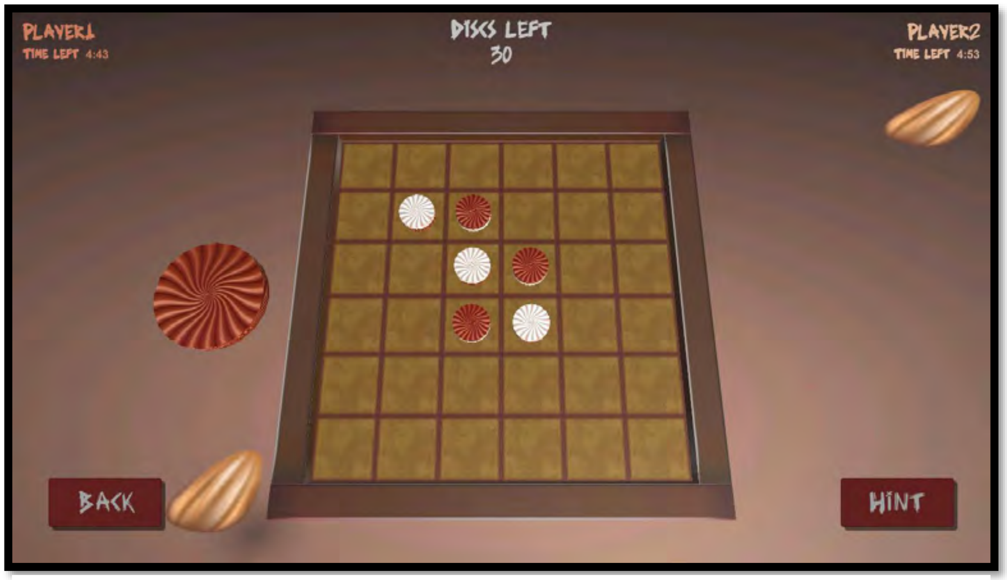


Figure 4.10: Game Play Interface 1

which specifies the difficulty to be set during the gameplay, the appropriate opponent either being another player or the computer itself, and lastly, the theme to fit to the comfort of the player (s). This also includes the number of discs left on the board which is located on the top center of the screen, and on the left and right side of which is the time of each of the players, with the player on the right side of the screen can also be the computer depending on who the opponent has been set. The interface can shift back to the Home Interface by pressing the Back Button, positioned on the bottom-left corner of the screen.

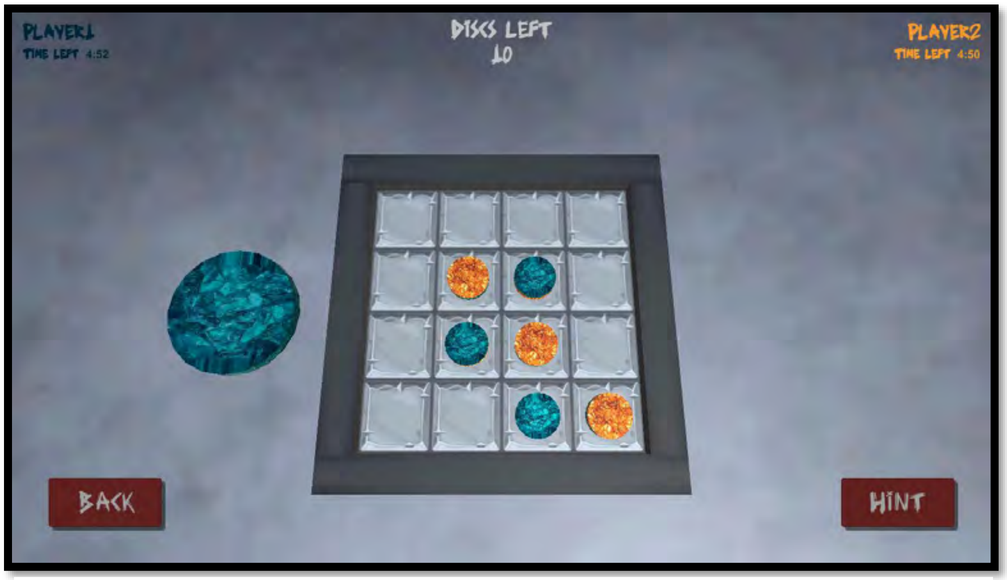


Figure 4.11: Game Play Interface 2

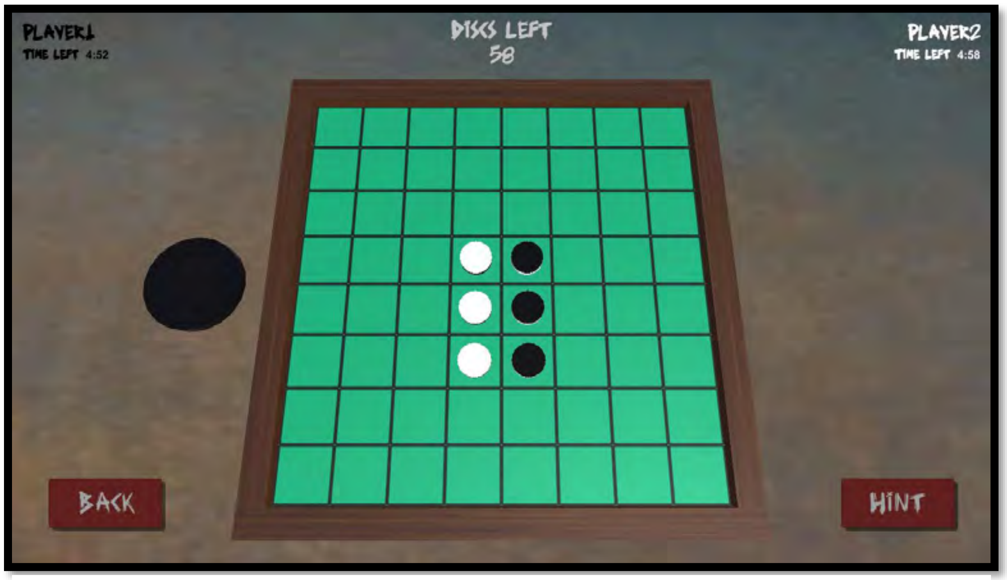


Figure 4.12: Game Play Interface 3

4.9.3 Game Over Interface

Friendly matches are always fun, so Othello is fully prepared for it. It presents to the players three different board sizes referred to as difficulty levels when competing against the computer itself, but to a player vs player mode of play, it becomes more interesting. There is fixed time for each player to make a move. When the time is up for either one of the players, then the game is over for both and the winner is shown with a panel appearing on the Game Play Interface, showing the results of how many discs have been secured by both the players out of the total number of discs.

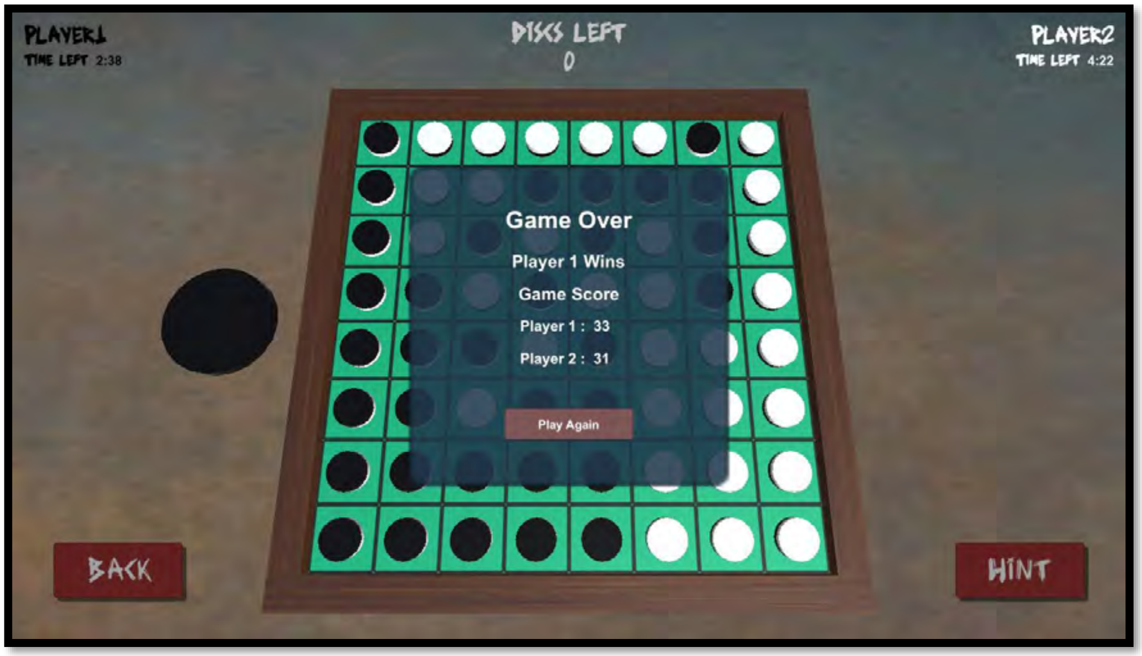


Figure 4.13: Game Over Interface

Chapter 5: SOFTWARE TEST DOCUMENT (STD)

5. Software Test Document

5.1 Introduction

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

5.1.1 Project Overview

This is a 3D board game project based on the realistic and tangible version played in real life, but with some extensions towards additional features. The basic concept of the game includes an 8x8 board where the player with the black discs goes first and the discs are to be placed adjacent or next to their oppositely colored neighboring discs in doing so, the player with most discs on the board wins. Additional features include the user can have the option of selecting to play against another player or against the computer itself. He/she can have a choice of selecting different configurations of the disc's before commencing the game, this is just to shift from the traditional black and white discs. It will have 3 different difficulty levels i.e. Easy, Medium and Hard. Player can unlock new custom boards and discs by earning and spending diamonds (incentive and appreciation of playing the game more often) after beating another player or the computer. The player when playing against the player can add comments that would get deleted or fade away shortly. The board game environment is three dimensional giving a realistic stimulation.

5.1.2 Test Approach

A test approach is the test strategy implementation of a project, defines how testing would be carried out. Testing technique used for **Othello** is beta testing (black box testing) for the time being. The main purpose of this testing is to evaluate the system's compliance with the business requirements and verify if it has met the required criteria for making an action. Beta testing reduces product failure risks and provides increased quality of the product through customer validation.

Another approach that I have used is load testing to determine the system's behavior during the decisions to be made during hint providing and countering a move with the best move possible. This type of testing has been done to check the performance of the product.

5.2 Test Plans

5.2.1 Features to be tested

Features to be tested are all according to user perspective. For example

- Play
- Play Against Player
- Play Against Computer
- Select Level
- Customize
- Get Hint
- Quit Game

5.2.2 Testing tools and environment

Following tools and environment are used for testing:

- PC/laptop
- Windows operating system

5.3 Test Cases

A test case describes an input, action, or an event and an expected response, to determine if a feature of a software application is working correctly.

5.3.1 TC-1: Play

Table *5.1* is a test case for **Play**. This test case tells us about testing of **Play** scenario.

Table 5.1: Test Case for Play

ID	TC-1
Description	Check that the player can successfully play the game.
Setup	The application is installed in the PC under use.
Instructions	1. Open the game/application by placing the cursor on the game's icon. 2. Double-click on the game's icon to open the game or right-click on the game's icon and click open. 3. Click on the play option.
Expected Result	The game/application has successfully opened on the desktop screen.
Actual Result	The game/application has successfully opened on the desktop screen.

Verdict	Pass

5.3.2 TC-2: Play Against Player

Table *5.2* is a test case for **Play Against Player**. This test case tells us about testing of **Play Against Player** scenario.

Table 5.2: Test Case for Play Against Player

ID	TC-2
Description	Check that the player can successfully select the play against another player mode of play.
Setup	The application is installed in the PC under use.
Instructions	1. Select the player vs player mode. 2. Click on the play button.
Expected Result	The player successfully selects play against another player mode of play.
Actual Result	The player successfully selects play against another player mode of play.
Verdict	Pass

5.3.3 TC-3: Play Against Computer

Table *5.3* is a test case for **Play Against Computer**. This test case tells us about testing of **Play Against Computer** scenario.

Table 5.3: Test Case for Play Against Computer

ID	TC-3
Description	Check that the player can successfully select the play against computer mode of play.
Setup	The application is installed in the PC under use.
Instructions	1. Select the player vs computer mode. 2. Click on the Start button.
Expected Result	The player successfully selects play against computer mode of play.
Actual Result	The player successfully selects play against computer mode of play.
Verdict	Pass

5.3.4 TC-4: Select Level

Table *5.4* is a test case for **Select Level**. This test case tells us about testing of **Select Level** scenario.

Table 5.4: Test Case for Select Level

ID	TC-4
Description	Check that the player can successfully select anyone of the three difficulty levels available.
Setup	The application is installed in the PC under use.
Instructions	1. Open the application, the Home screen will appear. 2. Select on the player vs player/computer mode. 3. Select anyone of the three difficulty levels available i.e. 4x4 for Easy, 6x6 for Medium and 8x8 for Hard.
Expected Result	The player successfully selects his/her desired difficulty level.
Actual Result	The player successfully selects his/her desired difficulty level.
Verdict	Pass

5.3.5 TC-5: Customize

Table *5.5*is a test case for **Customize**. This test case tells us about testing of **Customize** scenario.

Table 5.5: Test Case for Customize

ID	TC-5
Description	Check that the player can successfully customize the game pieces/items.
Setup	The application is installed in the PC under use.
Instructions	1. Open the Application, the Home screen will appear. 2. Select anyone of the three modes i.e. Wood for Default, Candy for the 2nd theme, and Jewel for the 3rd and last theme, on the Mode scroll bar. 3. Click on the Start button.
Expected Result	The player successfully customizes the game pieces/items based on the selected theme.
Actual Result	The player successfully customizes the game pieces/items based on the selected theme.
Verdict	Pass

5.3.6 TC-6: Get Hint

Table *5.6*is a test case for **Get Hint**. This test case tells us about testing of **Get Hint** scenario.

Table 5.6: Test Case for Get Hint

ID	TC-6
Description	Check that the players can successfully get hints in a limited number of times from the system.
Setup	The application is installed in the PC under use.
Instructions	1. Select the Versus option with either the Player Vs Player or Player Vs Computer mode of play. 2. Click on the Start button. 3. Click on the get hint button on the bottom right corner of the screen, the system will indicate the best possible move to execute on the game board, based on the Minimax Algorithm.
Expected Result	The players can successfully get hints from the system.
Actual Result	The players can successfully get hints from the system.
Verdict	Pass

5.3.7 TC-7: Quit Game

Table *5.7*is a test case for **Quit Game**. This test case tells us about testing of **Quit Game** scenario.

Table 5.7: Test Case for Quit Game

ID	TC-11
Description	Check that the player can successfully exit the running application/game.
Setup	The application is installed in the PC under use.
Instructions	1. Open the Home screen of the game. 2. Click on the Exit button.
Expected Result	The player successfully quits the game.
Actual Result	The player successfully quits the game.
Verdict	Pass

Chapter 6: CONCLUSION AND FUTURE ENHANCEMENTS

6. Conclusion and Future Enhancements

6.1 Introduction

This document describes the project conclusions and future enhancements -what type of new features can be added with time.

6.1.1 Summary

This project allows organization to play the traditional and good old-fashioned Othello game with new extended features. This application provides functionalities to players of all ages.

6.1.2 Conclusions

We are now able to customize out traditional board and discs with the discs and boards available in our vault where can add more exclusive boards and discs by unlocking them with required number of gems. We can play not just against other players, but against the computer as well on three different levels of difficulty, to improve our skills and build more understanding of the game.

6.1.3 Future Enhancements

In future, the application can be enhanced by:

- Making internet connectivity like Facebook or any other popular social network where the game can be made available, in order to play against other players online.
- Making hotspot connectivity to play against other players on different PC's.
- Make a mobile application i.e. Android or IOS, of the desktop-based game.
- Make an option available for the player, to make the player design his/her customizations on the discs and board.

6.2 References

[1] “Introduction”, Google, [Online]. Available:
https://gogameguru.com/tag/deepmind-alphago-lee-sedol/. [Accessed 12 12 2018]

[2] “Software Process Model”, Google, [Online]. Available:
https://www.weblineindia.com/wp-content/uploads/2017/07/spiral-model-methodology_weblineindia.png. [Accessed 9 2 2019]

[3] “Similar Games”, Google, [Online]. Available:
http://www.worldothello.nu/?q=content/reversi-versus-othello. [Accessed 18 12 2018]

[4] “Game Rules”, Google, [Online]. Available:
http://www.flyordie.com/games/help/reversi/en/games_rules_reversi.html. [Accessed 18 12 2018]

[5] “Minimax Algorithm”, Google, [Online]. Available:
Provincial Healthcare Index 2013 (Bacchus Barua, Fraser Institute, January 2013, page-25). [Accessed 26 12 2018]

[6] “Minimax Algorithm Working, Alpha-Beta Pruning”, Youtube, [Online]. Available:
Algorithms Explained – minimax and alpha-beta pruning by Sebastian Lague. [Accessed 5 1 2018]