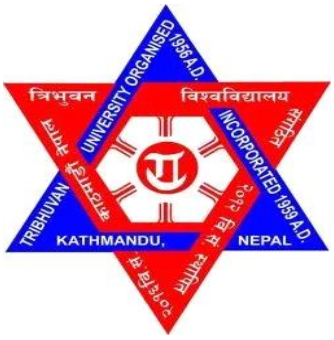


TRIBHUVAN UNIVERSITY

Institute of Science and Information Technology

"Baghchal Game using negamax algorithm and alpha-beta pruning with Zobrist hashing and killer heuristics"

A Project Report

(Course Code: CSC412)

Submitted to:

Office of the dean

Institute of Science and Information Technology Tribhuvan University

Kirtipur, Nepal

In Partial Fulfillment of the Requirement for the bachelor Degree in Computer Science and Information Technology

Submitted by:

Prabal Rai (15149/074)

Nabina Kusi (15146/074)

Sumitra Shrestha (15161/074)

SUPERVISOR'S RECOMMENDATION

It is my pleasure to recommend that a report on **"Baghchal game using negamax algorithm and alpha-beta pruning with Zobrist hashing and killer heuristics"** has been prepared under my supervision by **Ms. Sumitra Shrestha, Mr. Prabal Rai and Ms. Nabina Kusi** in partial fulfillment of the requirement of the degree of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT). This report is satisfactory and is an original work done by them to process for the future evaluation.

……

Mr. Nabin Ghimire

Supervisor

Department of Computer Science and IT

Bhaktapur Multiple Campus

Date……...

CERTIFICATE OF APPROVAL

The undersigned certify that he has read and recommended to the Department of Computer and Information Technology for acceptance, a project report entitled "**Baghchal game using negamax algorithm and alpha-beta pruning with Zobrist hashing and killer heuristics**" submitted by **Ms. Sumitra Shrestha, Mr. Prabal Rai, Ms. Nabina Kusi** in partial fulfillment for the degree of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT), Institute of Science and Technology, Tribhuvan University.

…….. …….

Mr. Nabin Ghimire **External Examiner**

Supervisor

Department of Computer Science & IT

Bhaktapur Multiple Campus

……

Sushant Poudel

Program Coordinator

ACKNOWLEDGEMENT

The completion of this project would not have been possible without the kind support and assistance of many individuals, and we are immensely blessed to have got this all along the duration of our projects. We would like to extend our profound gratitude to each and every one of them.

We express our sincere thanks to **Mr. Sushant Poudel**, CSIT Coordinator, Bhaktapur Multiple Campus to encourage us to the highest peak and to provide us with this opportunity to prepare the project.

We would like to express our gratitude to our project supervisor **Mr. Nabin Ghimire** who took keen interest on our project and guided to throughout the project by providing all the necessary ideas, information and knowledge for the developing of our game project.

We are also immensely obliged to our friends for their elevating inspiration, kind supervision and encouraging guidance for the completion of this project.

With respect
Prabal Rai (15149/074)
Nabina Kusi (15146/074)
Sumitra Shrestha (15161/074)

ABSTRACT

Baghchal game is a traditional game played from old times. It is a multiplayer game that contains tigers and goats that are moved in a board to gain the winning condition. Baghchal game presented here is a single player digital version of the traditional baghchal game that pits the human player with the system. The baghchal game allows a human player to play a game of baghchal with the computer AI where each of the players (human and system) move their respective pieces in the board to gain the winning condition. The project uses negamax algorithm and alpha beta pruning to allow the system to predict the most beneficial move for the system to win the game. Additionally, Zobrist hashing and killer heuristic are used to maximize the efficiency of the used algorithm. Negamax algorithm is the minimized version of the minimax algorithm (and algorithm actively used decision making algorithm). Alpha-beta pruning is used to optimize the existing negamax algorithm. Similarly, Zobrist hashing and killer heuristic are used to optimize alpha-beta pruning.

Keywords: Baghchal, traditional game, computer AI, negamax, alpha-beta pruning, Zobrist hashing, killer heuristic, Optimization.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI: Artificial Intelligence

2D: 2 Dimensional

IDE: Interface Development Environment

e.g.: Example

PM: Post Méridien

OS: Operating System

UI: User Interface

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION:

A digital game (or simply a game) is an interactive program for one or more players, meant to provide entertainment at the least, and quite possibly more. An adaptation of 'traditionally' played game, with rules, player representation, and environment managed through electronic means. Games nowadays use different features and technologies in order to make the games more realistic and more entertainment.

Strategic game is a game (e.g., a board game) in which the players' uncoerced, and often autonomous, decision-making skills have a high significance in determining the outcome. Almost all strategy games require internal decision tree-style thinking, and typically very high situational awareness. [1]. These games generally require implementation of some type of algorithms such that the players (kind of) mimic human thinking or predict some in order to win the game. Some of the algorithms include path finding algorithm, decision-making algorithms. The most used algorithms in strategic games are minimax algorithm and alpha-beta pruning popular strategic games like chess, checkers and baghchal use these algorithms to make the computer AI harder to defeat

Baghchal, or “Moving Tiger”, is a strategic, two-player board game that originated in Nepal. It is a game played between two opponents, one player controls four tigers and the other player controls up to twenty goats. The tigers hunt the goats while the goats attempt to block the tigers’ movements. Games like these are usually played physically using stones and board in the physical presence of two individuals (humans), one playing as tiger and the other playing as goat. But when played digitally, two possible options are presented. Either a human can play against another human (using multiplayer networking) or a human can play against the computer (AI). When playing against the computer, various technologies such as decision-making algorithm, path finding algorithm, neural networking, etc. are used.

1.2 PROBLEM STATEMENT:

Algorithmic implementations in gaming have developed to such a degree that these implementations have created realistic and harder-to-deal-with components in gaming. This has resulted in users working harder to cope up with these implementation (for e.g., player defeating AI generated opponent) which hence forces the game developers to make necessary improvements on the algorithm implementations for making the result more efficient. Strategic games like chess and baghchal generally use algorithms like minimax and alpha-beta pruning. However, these implementations may create bug during its implementation and the end product may be too complex due to which additional implementations are to be required.

Following are the problems faced during creating these strategic games and using the algorithms:

- Minimax algorithm gets really slow for complex games such as chess, checkers, etc. This type of games has a huge branching factor, and the player has lots of choices to decide. [2]
- One disadvantage of the minimax algorithm is that each board state has to be visited twice: one time to find its children and a second time to evaluate the heuristic value. [3]
- Alpha-beta pruning requires a set depth limit, as in most cases, it is not feasible to search the entire game tree. [4]

Also, after doing some research, we found that baghchal has lost its popularity comparing with other similar games like chess. AI in gaming has developed to such a degree that AI has been implemented in most of the components and applications of gaming. But even after all that, typical Nepalese games like baghchal are not implemented properly using algorithmic functionalities.

1.3 OBJECTIVES:

A measure of game refinement can be employed to assess the degree of game sophistication. Considering these facts, AI in gaming has developed to such a degree that it has been implemented in most of the components and applications of gaming.

The purpose of this project is to create an interactive 2D Baghchal game. The game is a single player game where the opponent uses decision making algorithm for prediction and carrying out the rewarding moves. The computer AI uses negamax algorithm and alpha-beta pruning for predicting the moves and Zobrist hashing & killer heuristics to increase the efficiency of alpha-beta pruning.

The prime objectives of this project are listed below.

1. Creating a 2D interactive game. Here “interaction” meaning that the opponent (System) interacts with the player by providing effective response for the move that the player made.
2. Creating a Baghchal game using AI technologies.
3. Implementing negamax algorithm and alpha-beta pruning in Baghchal game.
4. Using Zobrist hashing & killer heuristics to increase the efficiency of alpha-beta pruning

1.4 SCOPE AND LIMITATIONS:

Strategic games that pit the human players with the computer opponent requires use of technologies (technologies that either defines the correct decision or one that accurately predicts the possible moves for victory) in order to provide the human a tough competition for winning a game (in our case it is baghchal). The presented project provides an interface which allows a human player to play an interactive game of baghchal with the computer AI (system). This project uses decision making algorithm based on the present scenario in order to determine the most efficient move such that the system wins the game of baghchal. The algorithms used provide the systems a certain kind of decision-making capacity which helps the system decide which move would allow the system gain the winning condition. Rather than the commonly used method in strategi game making (where most effective move is predicted using positioning of all the items in the board i.e., each and every combination is calculated) in the project only the moves of the player are used for prediction.

However, there does exist some limitations that the system possesses. They are:

1. Prediction is done for the moves of players and not all the possible moves. Hence the AI is relatively weak
2. Since only the players are used for prediction, the vision of prediction is pretty limited.

1.5 DEVELOPMENT METHODOLOGY:

The projects or application are considered as successful after well management. There are numerous software development methodologies to choose from but the project can be managed efficiently only upon the best selection of methodology which would fit the project at hand. The project has been implemented following the agile software development methodology, considering the fact that projects created using agile method are liable to change and that some of the method of implementation could have been changed during the construction of the product. While developing the application through agile methodology, core focus is given upon the people, prototypes, collaboration and iteration.

1.6 REPORT ORGANIZATION:

Chapter 1: This chapter explains the overview, introduction, problem statement, objectives, scope and limitation of the presented system.

Chapter 2: This chapter describes the background study and the literature review related to the system

Chapter 3: This chapter covers all the history, methods, requirement specification and feasibility analysis and structured system requirements.

Chapter 4: Design of Advanced Baghchal game project is explained in detail with all the necessary diagrams and brief functionality.

Chapter 5: Process of implementation and testing is described along with all the tools used for the development.

Chapter 6: Conclusion and future scope of the application are explained.

CHAPTER 2: BACKGROUND STUDY AND LITERATURE REVIEW

2.1 BACKGROUND STUDY:

“Baghchal” is said to be a thousand years old game. Some sources say that the game originated in Nepal, and others that it came from further south in India, where similar games have been played for centuries. Whatever its antiquity, the game is still popular today, with traditional brass sets with cast pieces still being made and sold across the world by Nepali craftsmen. In Nepalese tradition, the game is said to be created by Mandodari, daughter of Mayasura, king of the Asuras and Hema, apsara water nymph. Mandodari is Queen of Ravana, king of Lanka according to the Hindu epic Ramayana

Baghchal is a traditional game originated in Indian sub-continent; nowadays is still a popular game, at least in Nepal. It´s one of the more interesting hunting games (those games played between unequal forces with different goals, where pieces use to symbolize animals). Bagh in Nepali means tiger, and chal means move, hence you could translate it as the Tiger Moving Game or Move the Tigers. [5]

People call it by different names in different areas. In Bangladesh this game is known as "Bagh Bondi (Bagh Bandi)". In Hindi it is "Bagh Bakri", in Tamil Nadu "Adu Puli Aatam", in Telugu "Puli Meka", in Karnataka "Adu Huli", in Punjab it is "Sher Bakar", in Orissa (Odisha) it is "Bagha Chheli" and in Nepal it is called as "Bagh Chal".

Two sides take part in the game: 4 tigers trying to capture the 20 goats who defend themselves by blocking the tigers. The gameboard consists of a grid of 25 points with lines of valid movement connecting them [5] where at the start of the game all four tigers are placed on the four corners of the grid, facing the center. No goats are placed on the board during the initial setup. [6]

The player controlling the goats moves first, by placing a goat onto a free intersection on the board. Tigers may move along the lines from one intersection to another. Once all of the goats have been placed on the board, goats must move in the same fashion as the tigers, one intersection to another. Moves alternate between players. Tigers capture goats by jumping over them to an adjacent free position

Rules for tigers:

1. They can move to an adjacent free position along the lines.

2. They can capture goats during any move, and do not need to wait until all goats are placed.

3. They can capture only one goat at a time.

4. They can jump over a goat in any direction, as long as there is an open space for the tiger to complete its turn.

5. A tiger cannot jump over another tiger.

The goats must move according to these rules:

1. Goats cannot move until all goats have been positioned on the board.

2. They must leave the board when captured.

3. They cannot jump over tigers or other goats.

The game is over when either, the tigers capture five goats, or the goats have blocked the tigers from being able to move. [6]

2.2 LITERATURE REVIEW:

Baghchal being a strategic turn-based game such as chess, checkers, or shogi requires similar methods of implementations while creating an intelligent AI that can decide which move is more effective to win the game, and not just randomly placing items here and there. There have been a lot of implementations in chess, go and shogi using minimax algorithm and alpha beta pruning. Similarly, in some cases instead of using any algorithm an actual AI is trained (either using supervised or unsupervised learning) such that the trained AI plays a game of chess like any genius would do. Some of the famous chess AI (chess engines) are: [7]

1. AlphaZero
2. Stockfish
3. Leela Chess Zero
4. Komodo Chess
5. Deep Blue
6. Houdini Chess
7. Shredder Chess

Works in Baghchal: Bagh Chal is a relatively simple board game in terms of game tree complexity, The baghchal programs found online use search algorithms based on variants of the minimax algorithm such as alpha-beta pruning to traverse the game tree.

Prior works have been done to evaluate the game under optimal play and even exhaustively analyze the endgame phase of the game using retrograde analysis. In their book called Games of No Chance 3, authors Lim Yew Jin and Jurg Nievergelt even prove that Tigers and Goats is a draw under optimal play. [8]

Soyyz Jung Basnet: One notable work done for implementing AI in baghchal is that of Soyuz Jung Basnet whose project is inspired by AlphaZero, a general reinforcement learning agent by Google DeepMind. Instead of creating an agent that uses brute-force methods to play the game, his project takes a different route where the agent learns to improve its performance by continually playing against itself. It uses a single deep residual convolutional neural network which takes in a multilayered binary board state and outputs both the game policy and value, along with Monte Carlo Tree Search.

CHAPTER 3: SYSTEM ANALYSIS

3.1 SYSTEM ANALYSIS:

3.1.1 REQUIREMENT ANALYSIS

Requirement analysis was done in order to determine the actual needs that are to be satisfied in order to create the desired product. There are two types of requirements:

i. Functional Requirements
ii. Non-functional Requirements

i. Functional Requirements

The prepared application must be able to carry out the following:

- Allow a single user (human) to use that application at that instant (no multiplayer).
- Allow the opponent (system AI) to decide the moves using negamax algorithm and alpha beta pruning.
- Create a turn-based interface where both parties (the human use and the system AI) carry out the allowed operations (moving and/or killing opponents) turn wise.
- Save the state/position of the user and the opponent so that the system can predict the user's movement

Use Case Diagram:

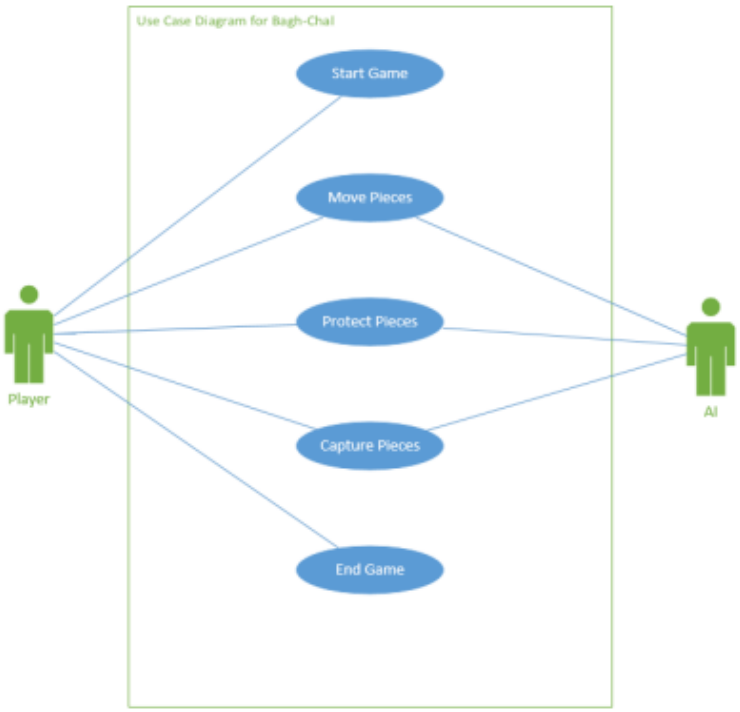


Figure 1: Use Case Diagram

ii. Non-functional Requirements

Non-functional requirements are used to describe the overall qualities or attributes of the system and how well or to what standard a function should be provided. The non-functional requirements used in Baghchal game are as follows:

i. Interface

OpenGl 3.0 is used for 2D graphics rendering and the 2D graphics are created using photoshop and similar tools. The Interface is simple and user friendly and also easy to use.

ii. Performance

The purpose of system development must be justified. The system should be able to provide accuracy to the user. Accuracy in this context means effective moves that allows the system to gain winning condition whereas forcing the human player to think of new ways to win the system AI.

iii. Usability

The system represents an online baghchal game and must provide accurate and genuine results to the users. The product should provide genuine competition to the human player while playing the game of baghchal

iv. Adaptability

The system should be responsive and adaptive to various devices such as android, iPhone, tablets as well as other personal computers. The system also must be compatible with various operating systems like macOS, windows, Linux.

3.1.2. FEASIBILITY STUDY

The feasibility study determines if the system can be built successfully with available cost, time and effort. The study is conducted by analyzing the collected requirements.

i. Technical

All the tools and software products required to construct this project are easily available on the web. It does not require a special environment to execute. It needs an IDE. All these aspects are easily affordable. The application requires simple user interfaces but the implementation of functions is complex.

ii. Operational

The system is reliable, maintainable, usable, sustainable, supportable and affordable. Therefore, this system is operationally feasible.

iii. Economic

As this system is not tested in the working field and no more equipment needs to be bought, also the system is made from the equipment that is already present, so the new system is economically feasible.

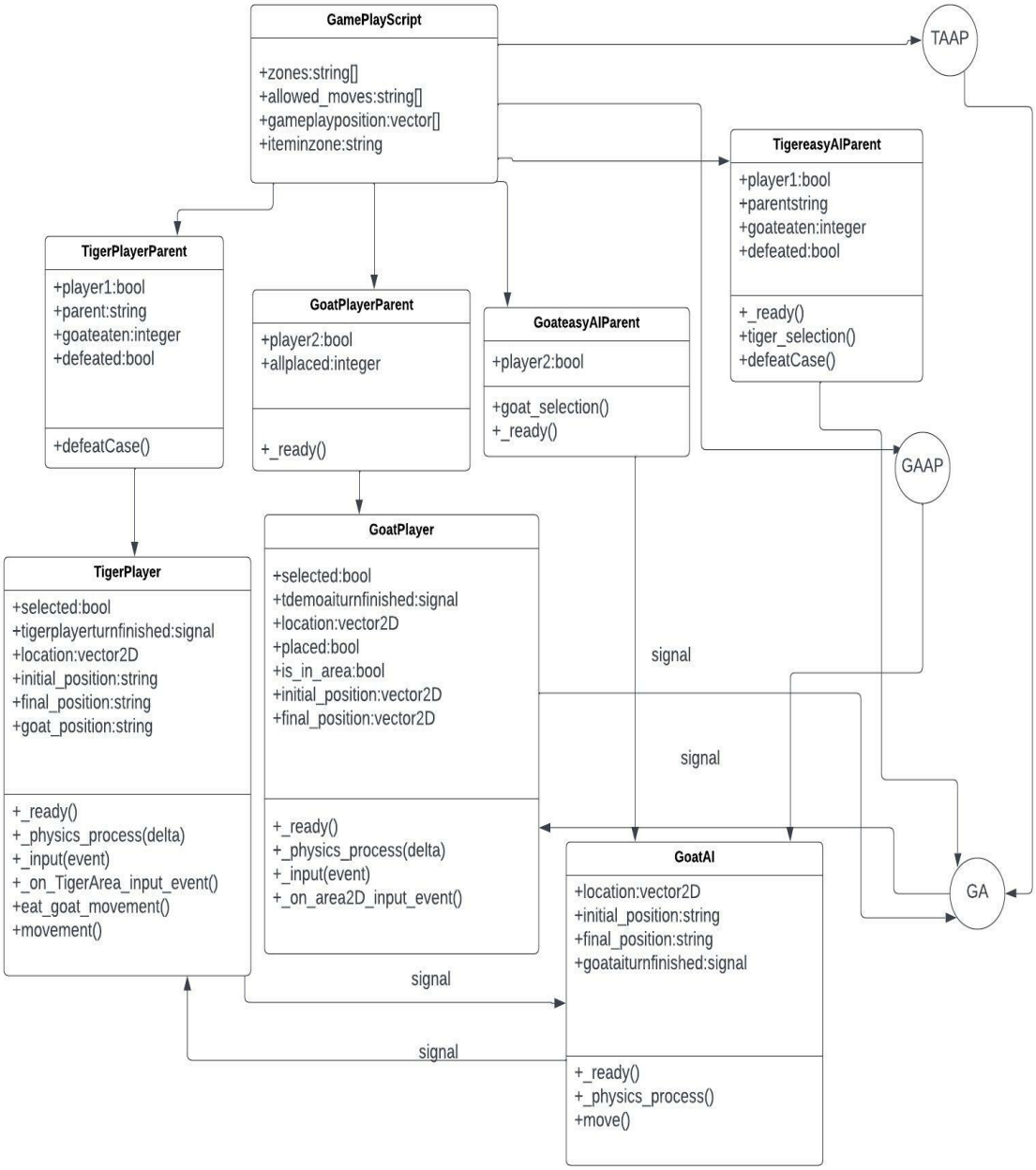
iv. Schedule:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Work	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th
2	Analysis	3w													
3	Design				3w										
4	Implementation						5w								
5	Testing								5w						
6	Documentation										5w				
7	Review											3w			
8	Presentation													2w	
9	w => weeks														

Figure 2: Schedule

3.1.3. ANALYSIS:

i. Object Modeling using class and Object diagram



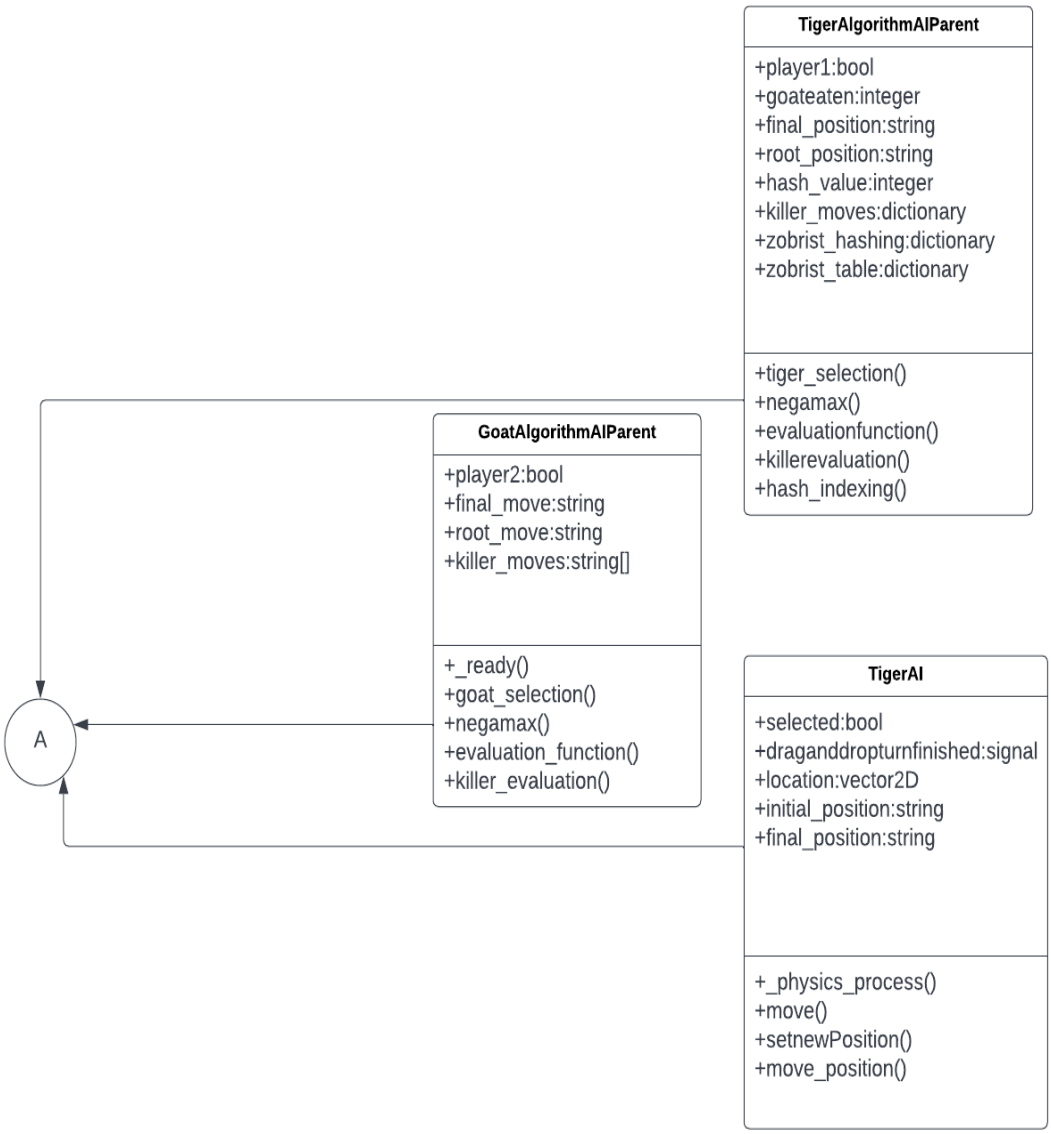


Figure 3: Class Diagram

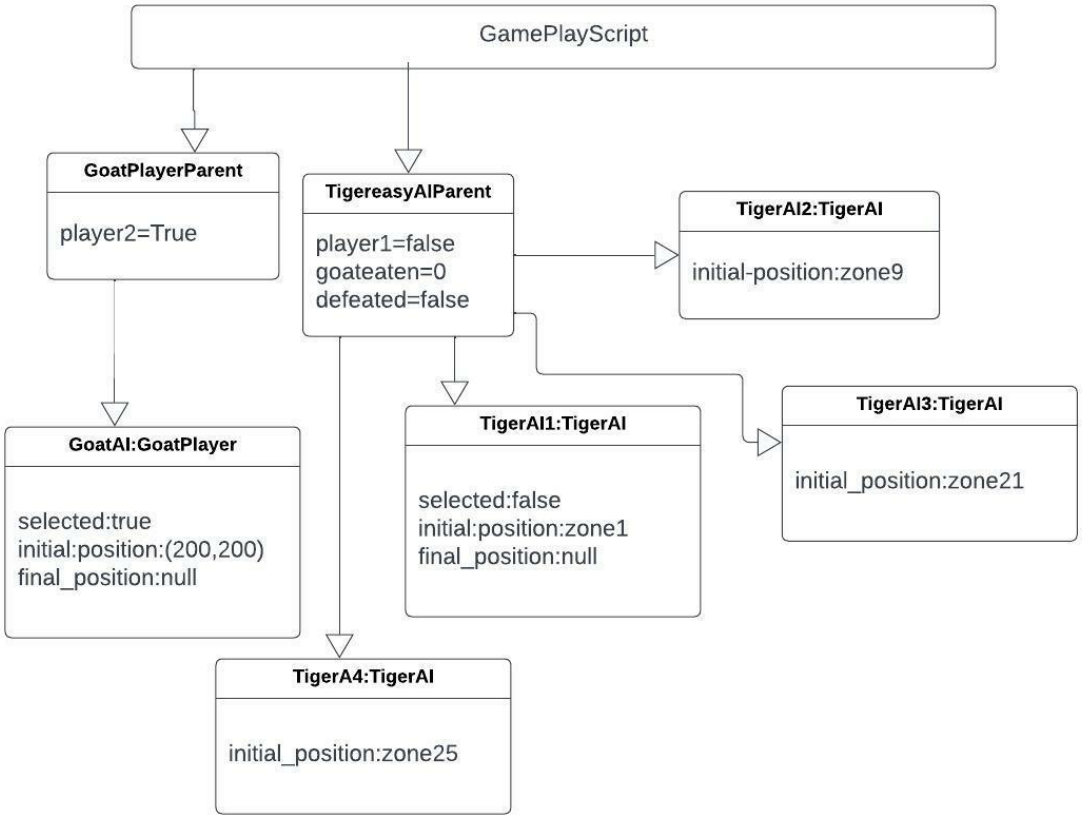


Figure 4: Object Diagram

ii. Dynamic Modeling using State and Sequence Diagram

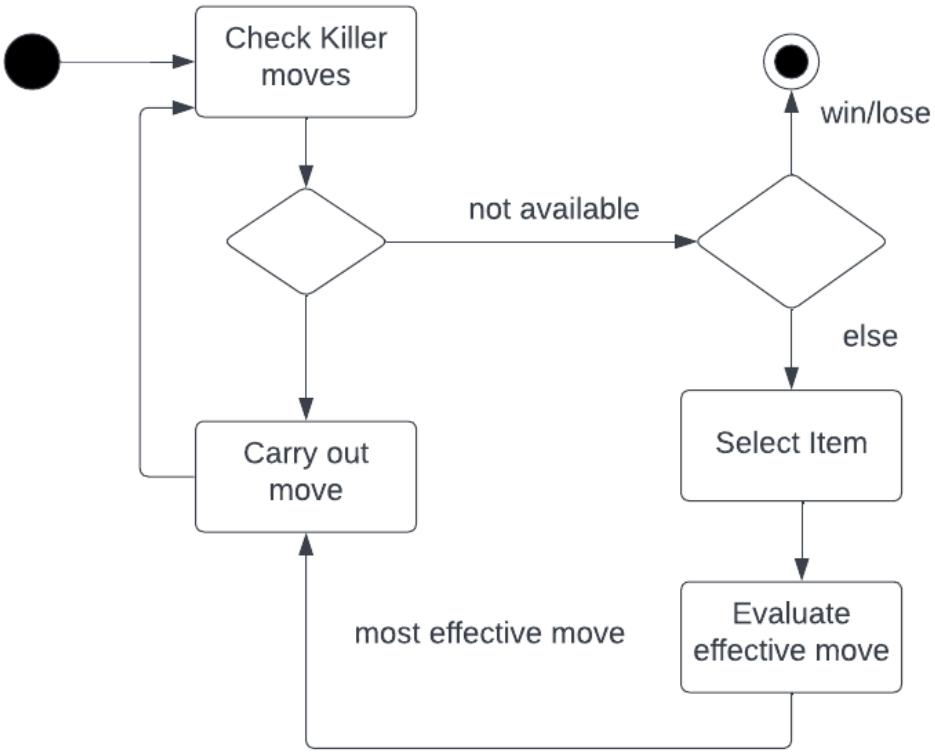


Figure 5: State Diagram for Intelligent AI

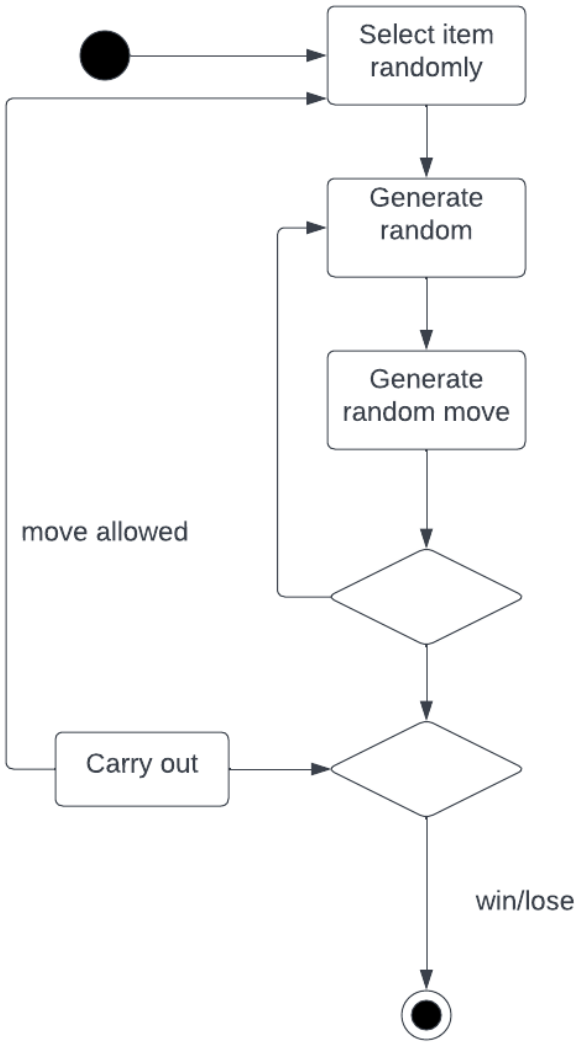


Figure 6: State Diagram for Random AI

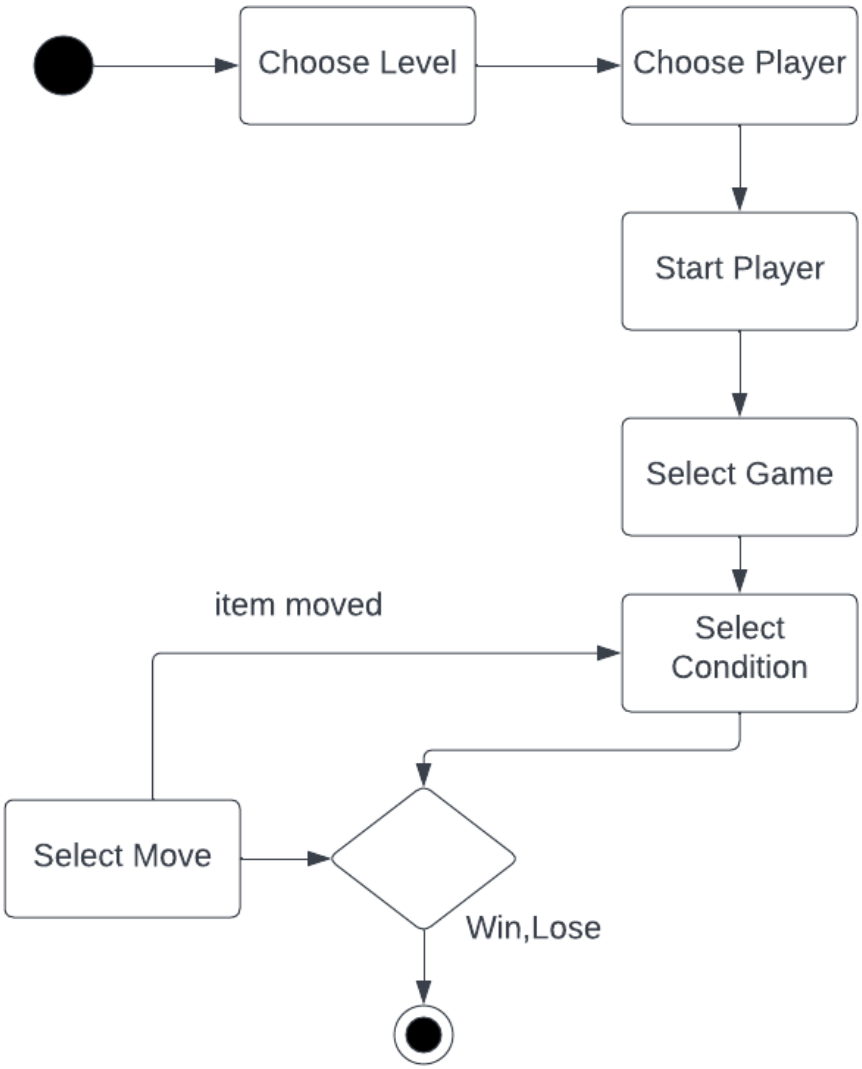


Figure 7: State Diagram for User

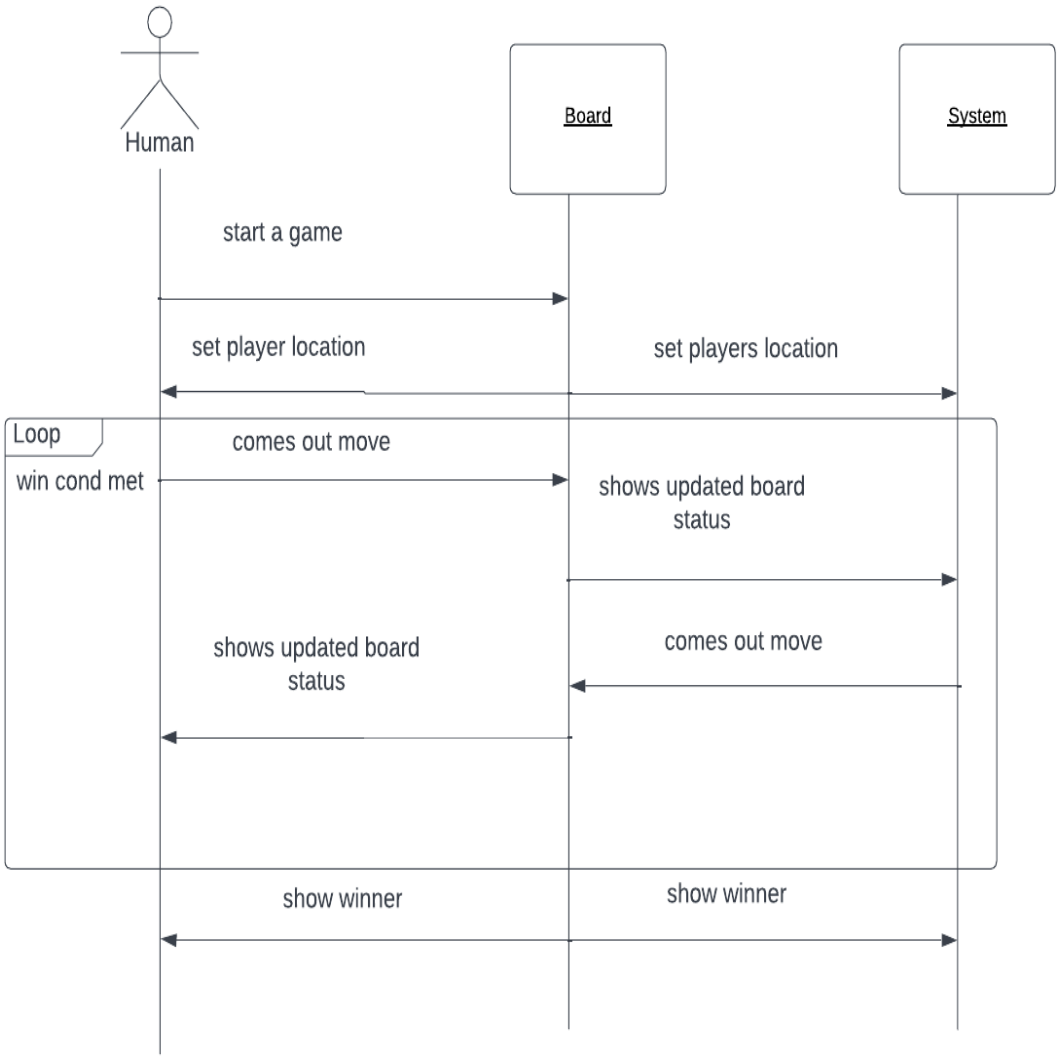
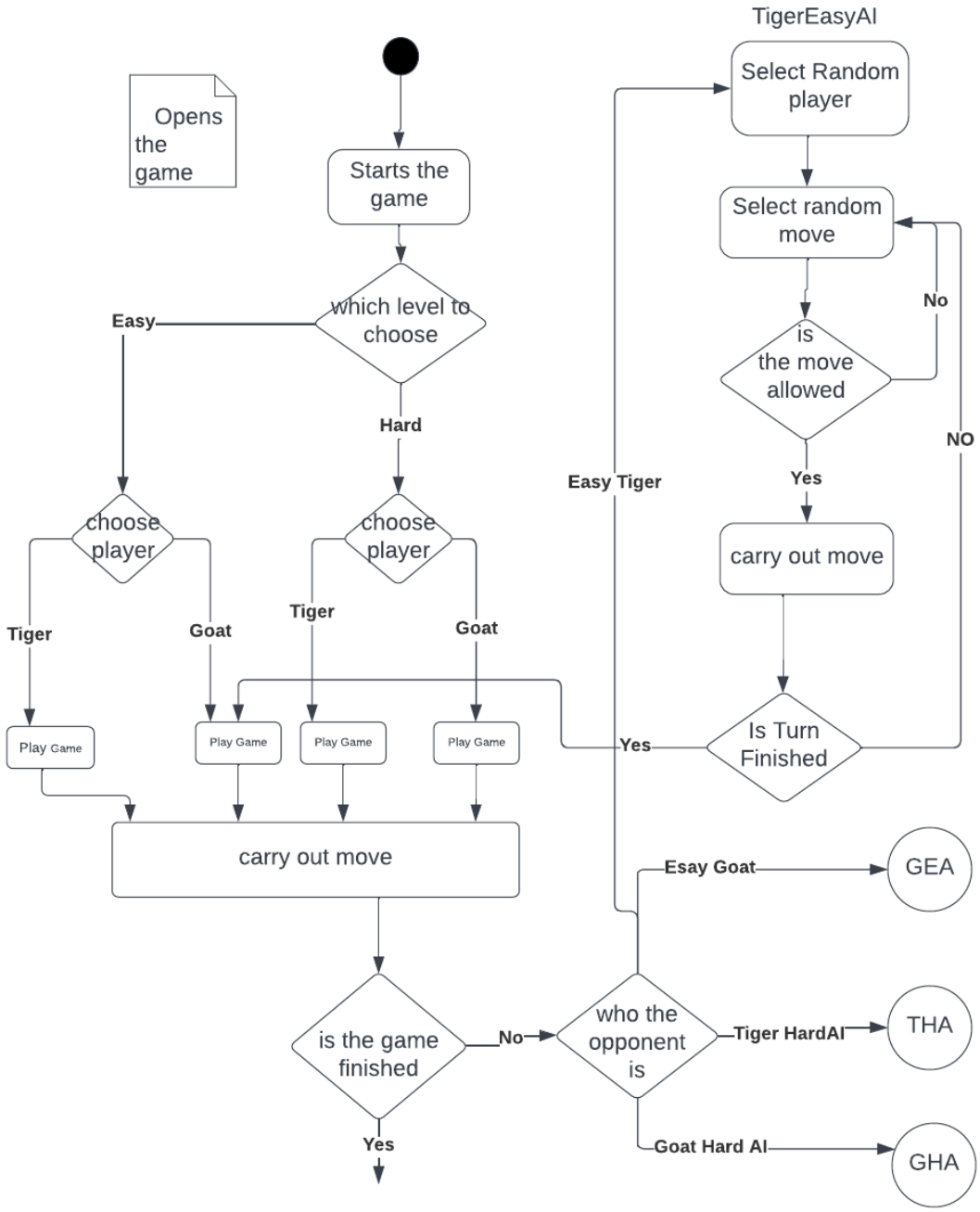


Figure 8: Sequence Diagram

iii. Process Modeling using Activity Diagram



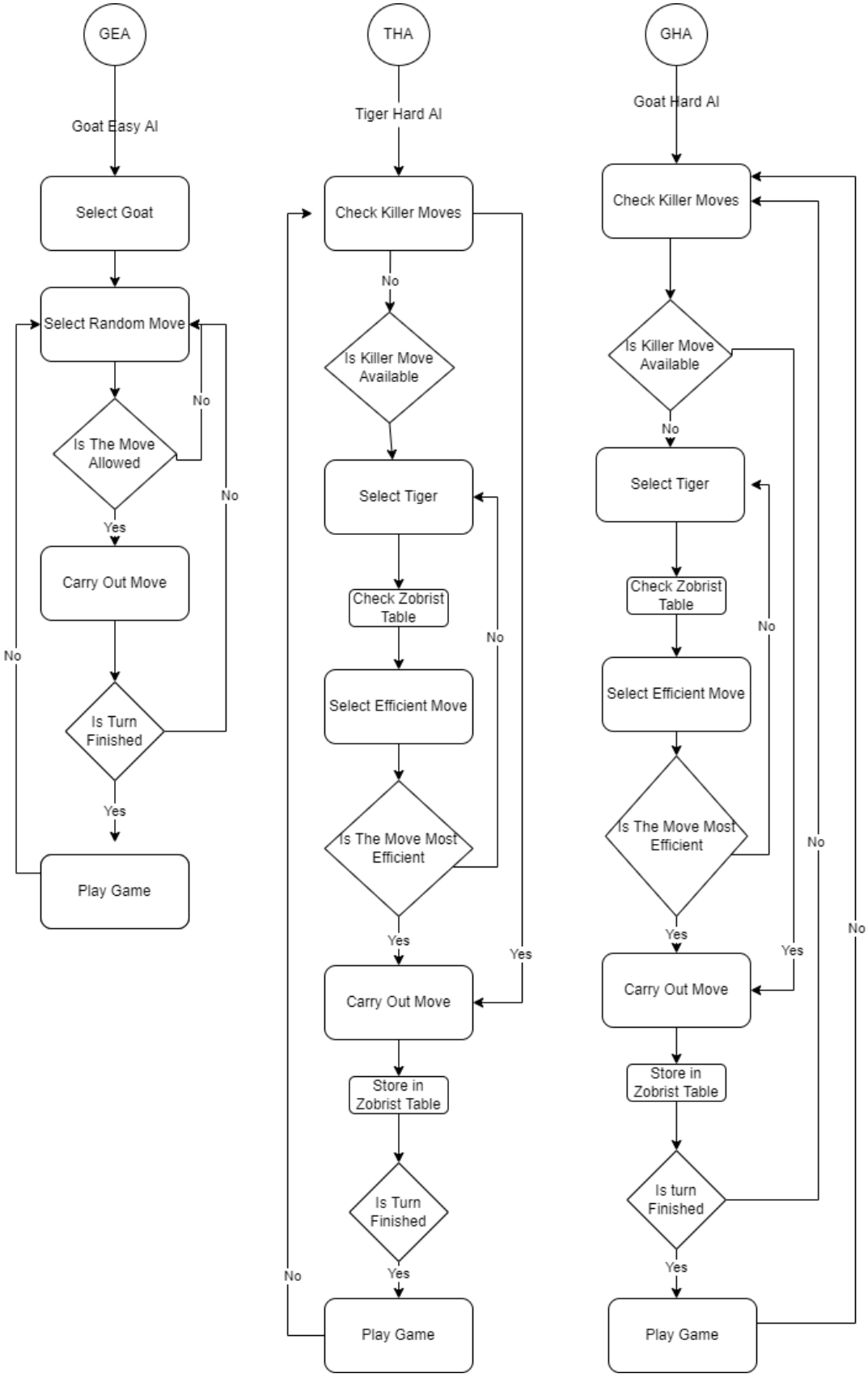
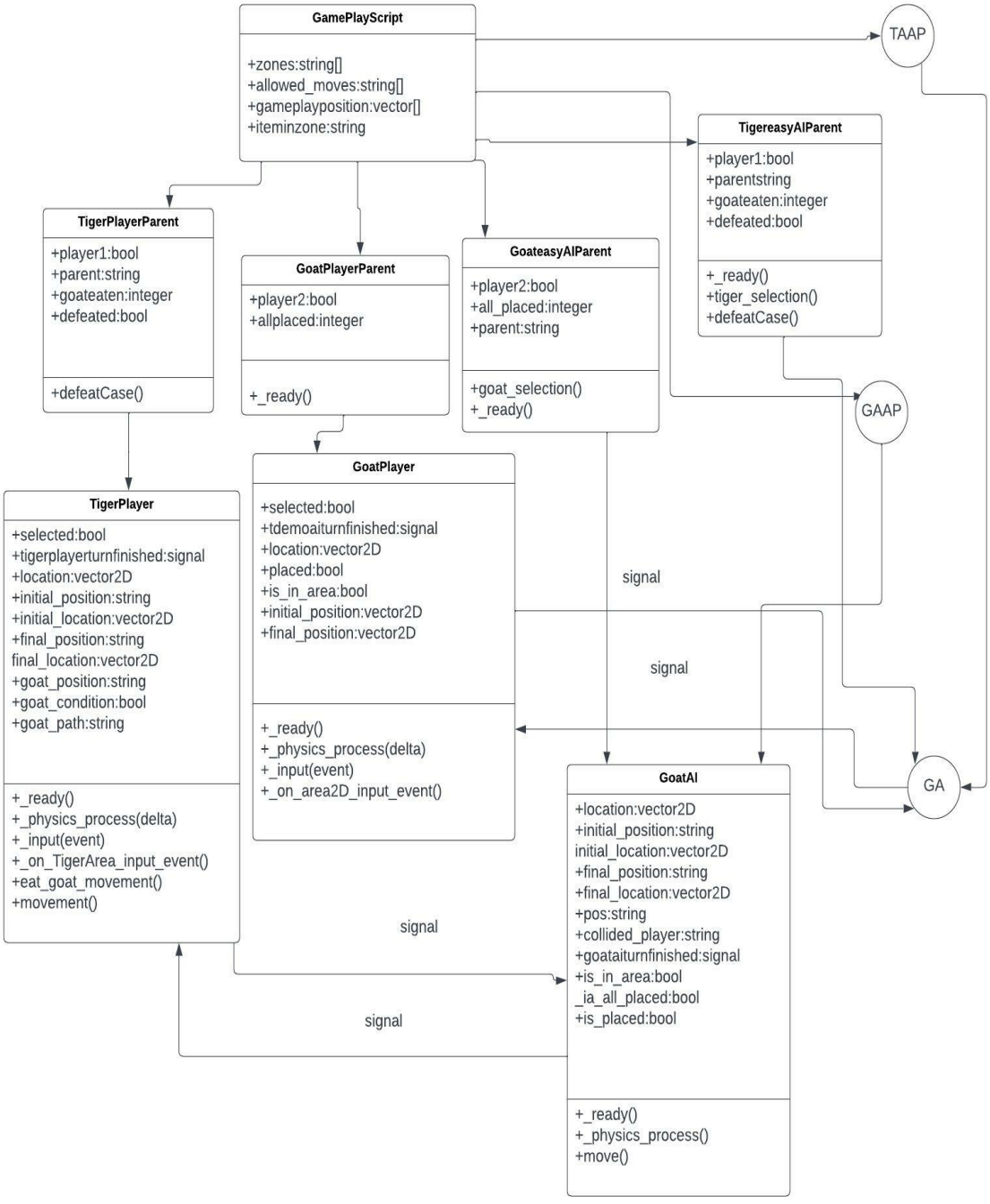


Figure 9: Activity Diagram

CHAPTER 4: SYSTEM DESIGN

4.1 DESIGN:

i. Refinement of Class, Object, State, Sequence and Activity diagrams



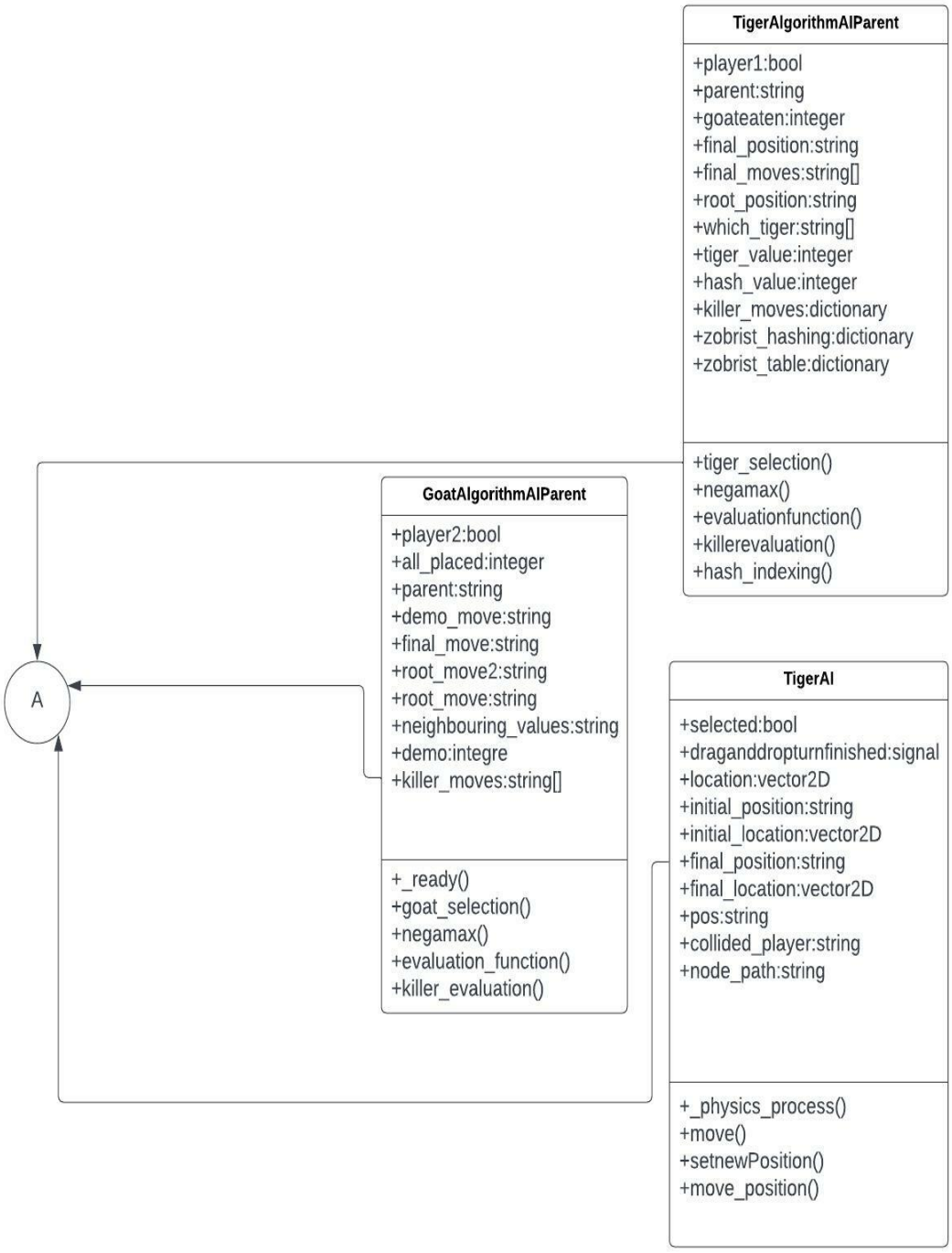


Figure 10: Refined class diagram

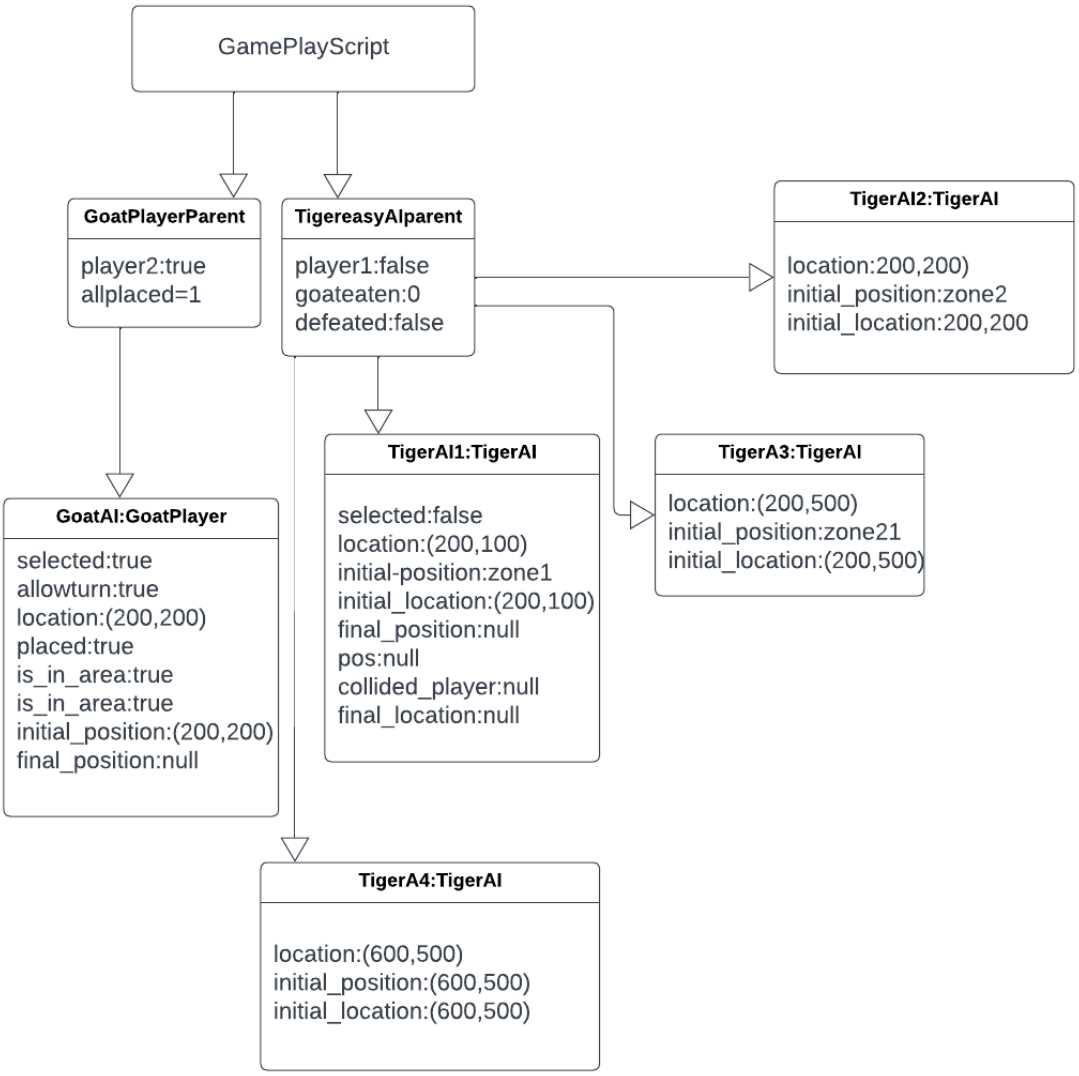


Figure 11: Refined object diagram

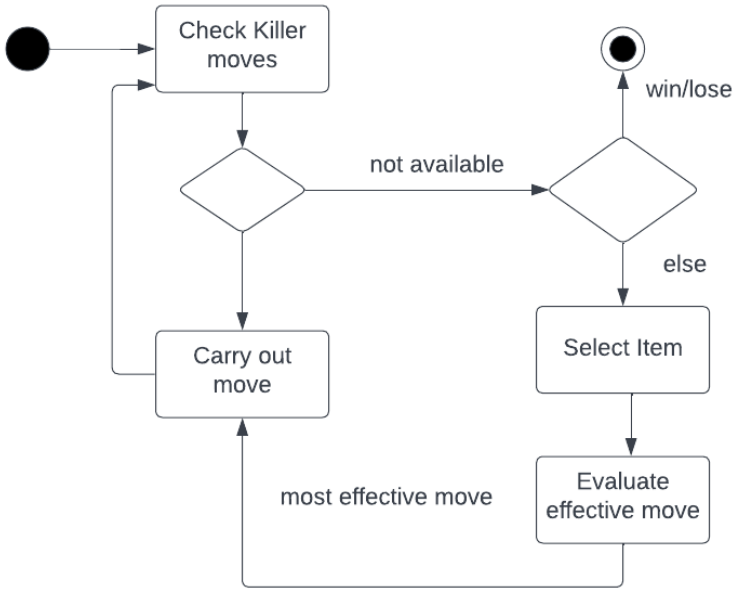


Figure 12: Refined state diagram for intelligent AI

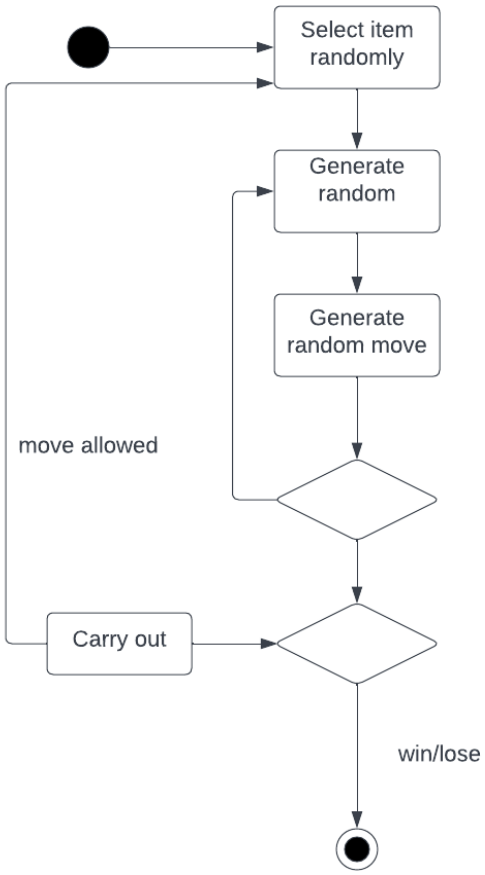


Figure 13: Refined state diagram for random AI

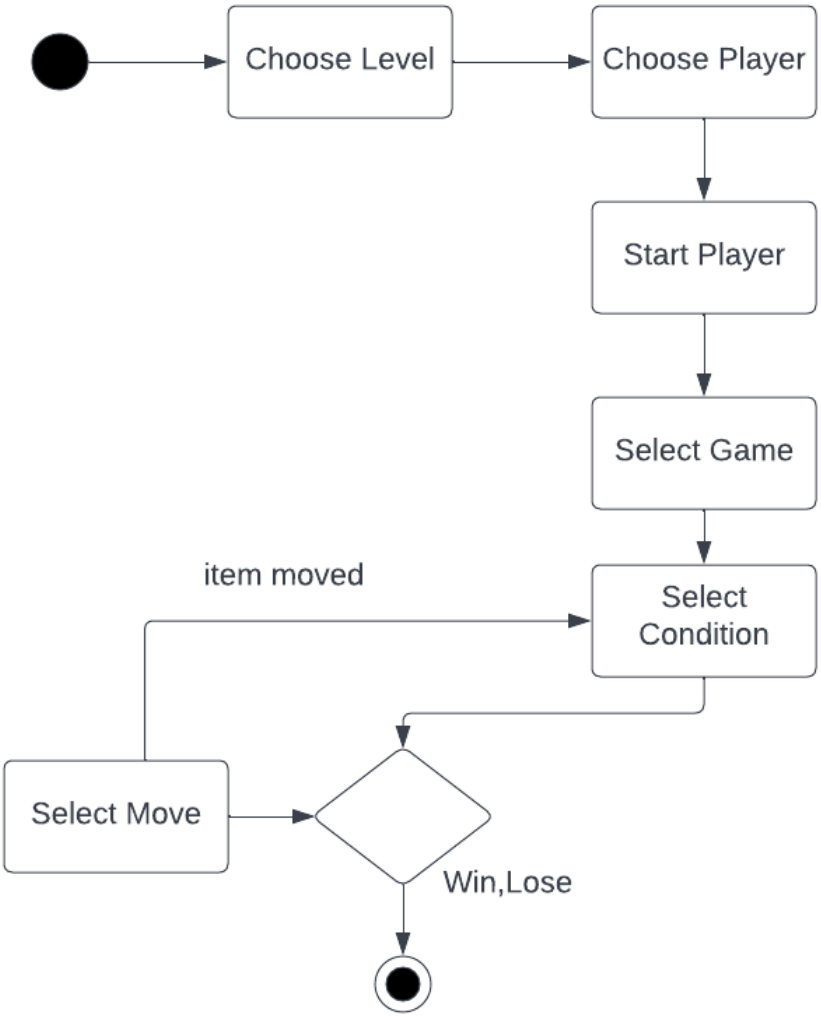


Figure 14: Refined state diagram for user

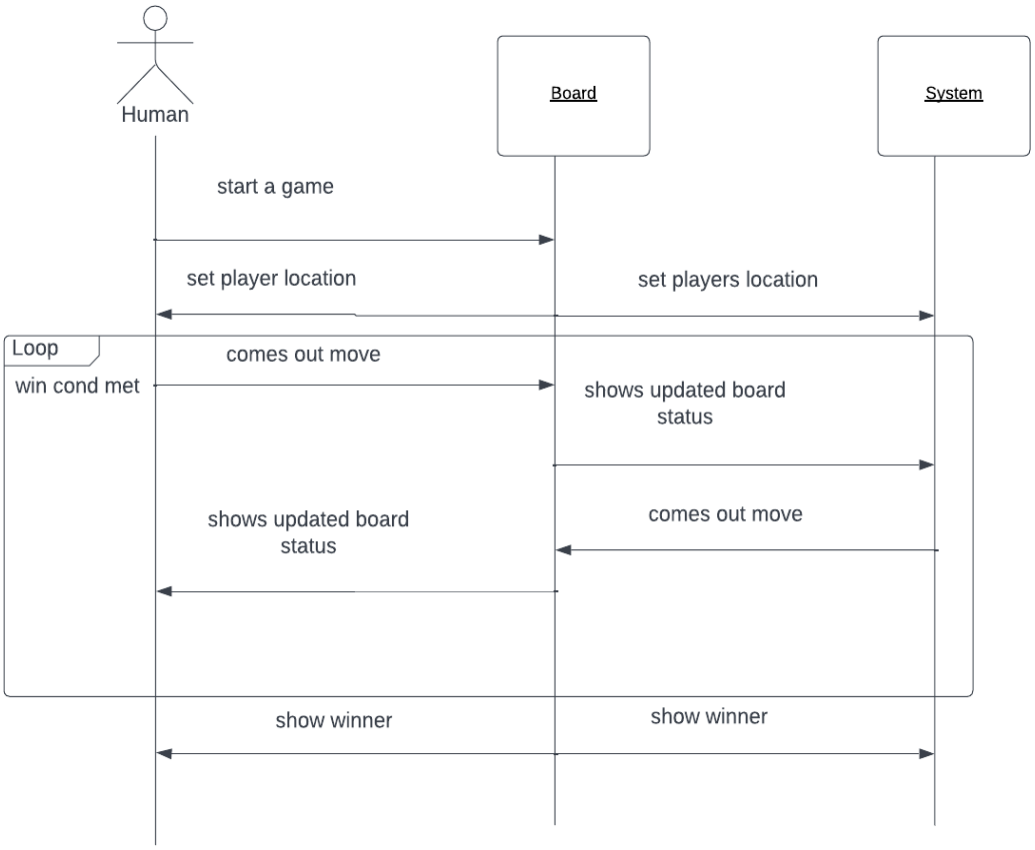
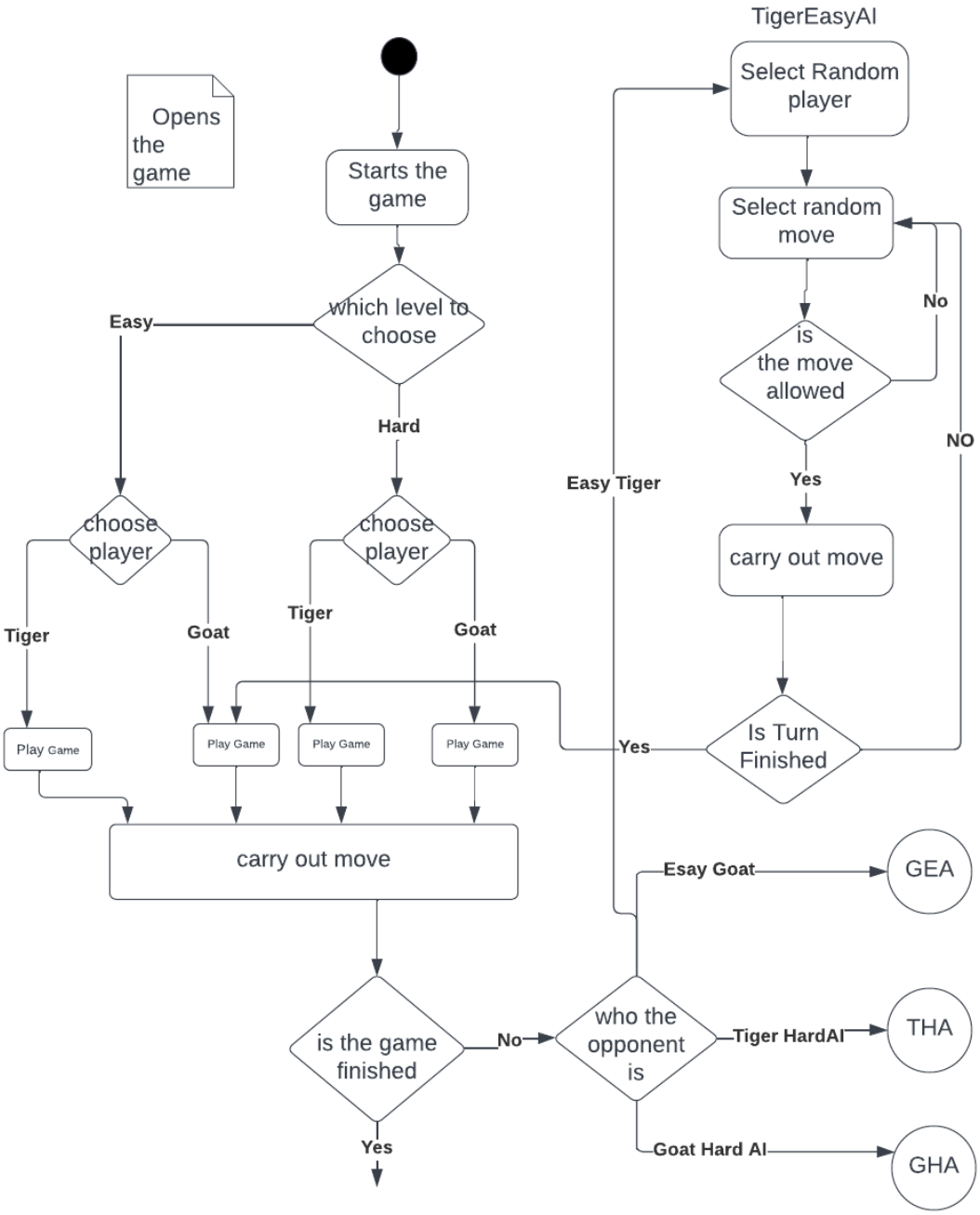


Figure 15: Refined sequence diagram


TigerEasyAI
Opens the game
Starts the game
which level to choose
Easy
Hard
choose player
choose player
Tiger
Goat
Tiger
Goat
Play Game
Play Game
Play Game
Play Game
carry out move
is the game finished
No
Yes
who the opponent is
Esay Goat
GEA
Tiger HardAI
THA
Goat Hard AI
GHA
Easy Tiger
Select Random player
Select random move
is the move allowed
No
Yes
NO
carry out move
Is Turn Finished
Yes

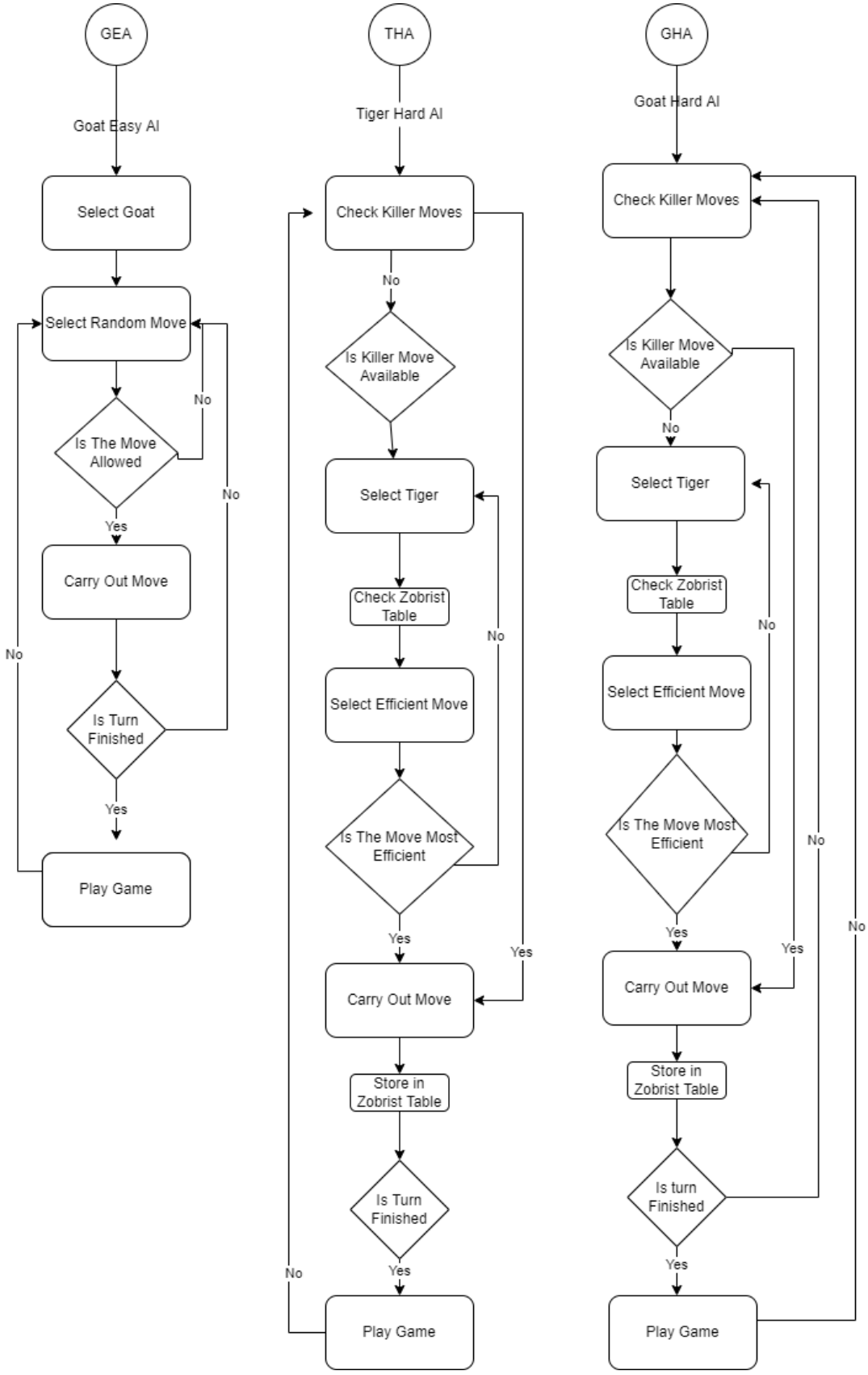


Figure 16: Refined activity diagram

ii. Component Diagram:

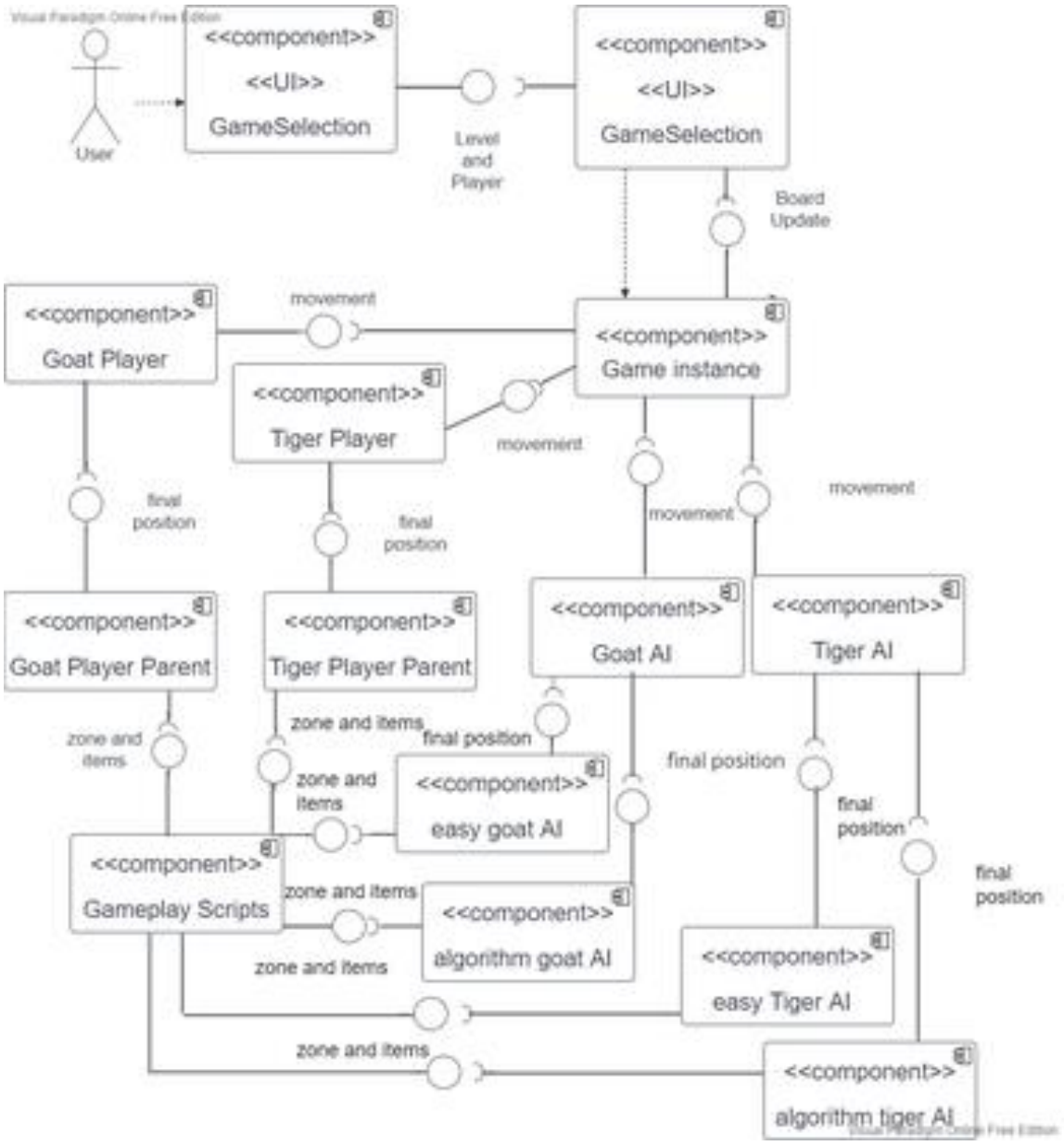


Figure 17: Component Diagram

iii. Deployment Diagram:

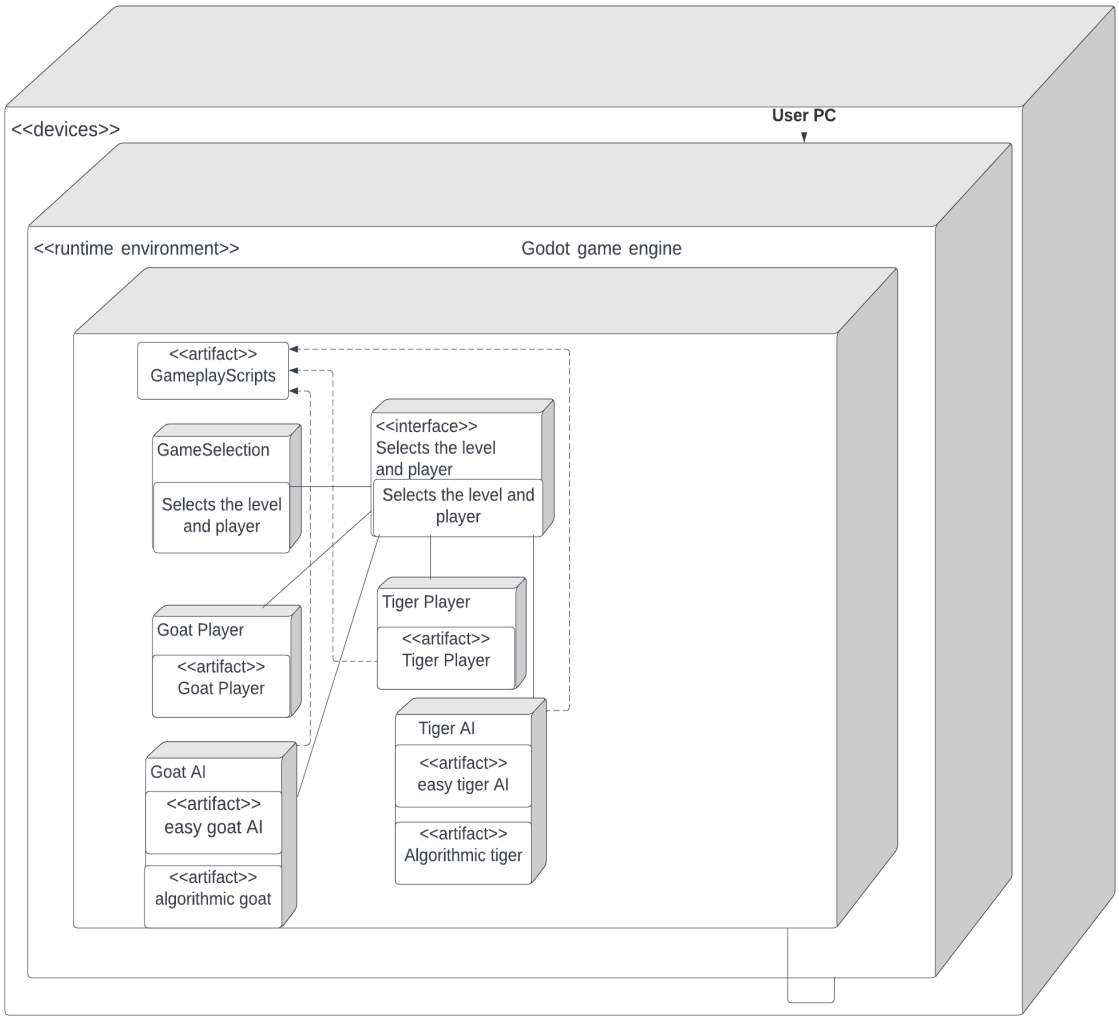


Figure 18: Deployment diagram

4.2 ALGORITHM DETAILS:

The final product depends on four different algorithms negamax algorithm, alpha-beta pruning, Zobrist hashing and killer heuristic. negamax algorithm is a decision-making algorithm that is used to predict effective move for the computer AI at that state such that the computer is in the winning condition, alpha-beta pruning is used to increase the efficiency of negamax algorithm. Zobrist hashing is used to implement transposition tables, whereas killer heuristic is a move-ordering method for alpha beta pruning.

1. negamax algorithm:

It is a common way of implementing minimax and derived algorithms. Instead of using two separate subroutines for the min player and the max player, it passes on the negated score due to following mathematical relation: [9]

max(a, b) == -min(-a, -b)

This algorithm relies on the fact that max(a,b)= -min(-a,-b) to simplify the implementation of the minimax algorithm. More precisely, the value of a position to player A in such a game is the negation of the value to player B. Thus, the player on move looks for a move that maximizes the negation of the value resulting from the move:

Pseudo-code for negamax base algorithm [10]

```
function negamax (node, depth, color) is
    if depth = 0 or node is a terminal node then
        return color × the heuristic value of node
    value:= −∞
    for each child of node do
        value: = max (value, −negamax (child, depth − 1, −color))
    return value
(* Initial call for Player A's root node *)
negamax (rootNode, depth, 1)
(* Initial call for Player B's root node *)
negamax (rootNode, depth, −1)
```

2. alpha-beta pruning:

alpha-beta pruning is a modified version of the minimax algorithm. [11]. alpha-beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm. It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. It is called alpha-beta pruning because it passes 2 extra parameters in the minimax function, namely alpha and beta.

Let's define the parameters alpha and beta.

alpha is the best value that the maximizer currently can guarantee at that level or above.

beta is the best value that the minimizer currently can guarantee at that level or above. [12]

Pseudo-code for negamax algorithm using alpha-beta pruning [13]

```
function negamax (node, depth, α, β, color) is
   if depth = 0 or node is a terminal node then
      return color × the heuristic value of node

   childNodes := generateMoves(node)
   childNodes := orderMoves(childNodes)
   value := −∞
   foreach child in childNodes do
      value := max (value, −negamax (child, depth − 1, −β, −α, −color))
      α := max (α, value)
      if α ≥ β then
         break (* cut-off *)
   return value
(* Initial call for Player A's root node *)
negamax (rootNode, depth, −∞, +∞, 1)
```

3. Zobrist hashing:

Zobrist hashing is a hashing function that is widely used in 2 player board games. It is the most common hashing function used in transposition table. Transposition tables basically store the evaluated values of previous board states, so that if they are encountered again, we simply retrieve the stored value from the transposition table. [14] It has also been applied as a method for recognizing substitutional alloy configurations in simulations of crystalline materials.

Pseudo-Code for Zobrist hashing [14]

```
// A matrix with random numbers initialized once

Table[#ofBoardCells] [#ofPieces]

// Returns Zobrist hash function for current conf-

// iguration of board.

function findhash(board):

   hash = 0

   for each cell on the board:

      if cell is not empty:

         piece = board[cell]

         hash ^= table[cell][piece]

   return hash
```

4. Killer heuristic:

In competitive two-player games, the killer heuristic is a move-ordering method based on the observation that a strong move or small set of such moves in a particular position may be equally strong in similar positions at the same move (ply) in the game tree. This technique improves the efficiency of alpha–beta pruning, which in turn improves the efficiency of the minimax algorithm. [15]

Pseudo-Code for Implementing Killer Heuristic in Alpha-Beta Search [16]

```
for (int i = killerMoves[ply].Length - 2; i >= 0; i--)
   killerMoves[ply][i + 1] = killerMoves[ply][i];
killerMoves[ply][0] = move;
```

Note: Now when you are performing move ordering (before iterating through the move list), you can determine whether a move is a killer move:

```
for (int slot = 0; slot < killerMoves[ply].Length; slot++) {
   int killerMove = killerMoves[ply][slot];
   for (int i = 0; i < movesCount; i++)
      if (moves[i] == killerMove) {
         // moves[i] is a killer move so move it up the list
         break;
   }}
```

CHAPTER 5: IMPLEMENTATION AND TESTING

5.1 IMPLEMENTATION

The implementation of the project is done using Godot game engine framework and the scripting language used is gdscript (python- like dynamic programming language). Still graphics creation was done using the help of graphics designing tools such as photoshop. Accordingly, input form the user is taken twofold:

i. Button input options that allow the system select the desired level (hard or easy) and the player with which the user wants to play with (tiger or goat)
ii. Mouse drag and drop input that moves the player from its existing position to the designated and allowed position

Input from (i) chooses the opponent (the level of the opponent and the player) for the human user. Level of the opponent ranges from easy (the opponent carries a valid but random in the board) to hard (the opponent uses the algorithmic implementations to carry out the most efficient moves). The player ranges between tiger (one that eats the goat) and goat (one that captures the tiger). The algorithmic implementation was done using negamax algorithm and alpha beta pruning and optimized using Zobrist hashing and killer heuristic. The input provided in these algorithms was the present position of the player and the present state of the board (i.e., the players allies and opponent). Similarly input from (ii) sets a valid final position for the player with which a valid move is carried out.

5.1.1 TOOLS USED

a. CASE Tools:

Computer-Aided Software Engineering (CASE) tool is a type of software tool which is primarily used to design and to implement applications. It is the implementation of computer facilitated tools and methods in software development. Some of the case tools used in this project are:

i. **Diagram Tools**: Diagram tools are used to generate diagrams such as use case, activity diagram, class and object diagram, state and sequence diagram. The diagram tools used in this project are Microsoft Visio and lucid chart.

ii. **Design Tools:** Design tools are used to create 2D graphics. This graphics are used to increase the design value of the project. The design tool used in this project is Adobe Photoshop.

iii. **Programming Tools:** Programming tools are used for development of the baghchal game. The programming tool used in this project is game engine called Godot which is used to create 2D games. The game engine helps in writing codes for the baghchal game and also allows the user to optimize the game making by providing user friendly features and interfaces.

b. Programming language:

The programming language used in the project is gdscript. Gdscript is a high-level, dynamically typed programming language used to create content. It uses a syntax similar to python (blocks are indent-based and many keywords are similar). Its goal is to be optimized for and tightly integrated with Godot Engine, allowing great flexibility for content creation and integration. [17]. Gdscript is optimized for Godot's scene-based architecture and can specify strict typing of variables. Godot's developers have stated that many alternative third-party scripting languages such as Lua, Python, and Squirrel were tested before deciding that using a custom language allowed for superior optimization and editor integration. [18]

c. Database platforms:

The project does not require the storing of huge amount of data as much as many other projects require. Although the project does not require storing and management of huge amount of data items, the product does require storing of some data related to the position of each Zone (point in the baghchal board), the hash table (used in Zobrist hashing), list of allowed moves for each zone, present item in each zone. the position of players in the board etc.so rather than using any database technology this project uses the concept of dictionaries to store and manage the required data.

Dictionary:

A dictionary is an associative container which contains values referenced by unique keys. Gdscript dictionary are used to store data as key: value pairs. Each key and value may be numbers, strings, or objects.

5.1.2 IMPLEMENTATION DETAILS OF MODULES

The project incorporates the use of various modules which work with each other in order to carry out the designated tasks. Each of the module contains its own list of methods and algorithmic implementations used to carry out various operations.

The following list defines the modules used in the system:

a. Easy tiger AI module
b. Algorithmic tiger AI module
c. Easy goat AI module
d. Algorithmic goat AI module
e. Goat player module
f. Tiger player module
g. Dropzone module
h. Selection module
i. Defeat case module

a. Easy tiger AI module

The easy tiger AI module basically generates a tiger AI that is relatively easier to defeat and manages the generated tiger AI. This module randomly decides a tiger instance and a random move for the randomly selected tiger instance (see Appendix 1). Then the module checks whether the selected move is valid (according to the allowed moves describes by the baghchal game rules) or not. If yes, that particular move is carried out and if no the deciding and selection is performed again until a valid move is not generated. The move is carried out using the tiger's initial location, the final valid move generated with the help of lerp() function (see Appendix 2)

Additionally, the modules contain functions that are invoked when the opponent has carried out its move and its turn is finished (which in this case is achieved using a signal). The invoked function calls the process of random selection of tiger and moves. Also, once the AI carries out its move it triggers a signal that says that the turn go tiger AI has finished and mow it's the turn of the opponent i.e., human

b. Algorithmic tiger AI module

The algorithmic tiger AI module basically generates a tiger AI that uses the algorithmic implementations for predicting and carrying out the moves. Due to this the generated AI is harder to defeat. This module checks the best move for all the tiger instances along with the evaluation value. The tiger instance with the highest evaluation value is selected and the respective move is carried out. It two or more tiger instances have the same evaluation value the tiger instance the latter one selected. The checking of the move is carried out using negamax algorithm, alpha-beta pruning, Zobrist hashing and killer heuristic (see Appendix 3).

During the checking of the best move first of all killer moves are checked. If the killer moves are present and the condition for killer evaluation are fulfilled, that particular move is selected and is then carried out.

If the killer evaluation condition is not satisfied, the module then chooses a tiger instance and then calls the negamax algorithm for that tiger instance. Initially in the negamax method the module first checks the Zobrist hash table to check for any previously visited moves. If present the negamax method returns that particular value and move as the best move for the tiger instance. if not, evaluation function is called for the child at depth 2. Evaluation function is carried out on all the child of the node generated until level 2 which is optimized using alpha-beta pruning. Once the evaluation value and the best move is determined, that values are stored in the Zobrist hash table at that particular hash value. The move is carried out using the tiger's initial location, the final valid move evaluated with the help of lerp() function (see Appendix 2).

Once the AI carries out its move it triggers a signal that says that the turn go tiger AI has finished it's turn and mow it's the turn of the opponent i.e., human. Additionally, the

modules contain functions that are invoked when the opponent has carried out its move and its turn is finished (which in this case is achieved using a signal). The invoked function calls the process of random selection of tiger and moves

c. Easy goat AI module

The easy goat AI module basically generates a goat AI that is relatively easier to defeat and manages the generated goat AI. If all goats are placed, then the goat instance is selected randomly and if not, a goat instance is selected in the decreasing order. This module decides a goat instance and a random move for the selected goat instance (see Appendix 4). Then the module checks whether the selected move is valid (according to the allowed moves describes by the baghchal game rules) or not. If yes, that particular move is carried out and if no the deciding and selection is performed again until a valid move is not generated. The move is carried out using the goat's initial location, the final valid move generated with the help of lerp() function (see Appendix 2).

Additionally, the module contains functions that are invoked when the opponent has carried out its move and its turn is finished (which in this case is achieved using a signal). The invoked function calls the process of random selection of goat and moves. Also, once the AI carries out its move it triggers a signal that says that the turn go tiger AI has finished and mow it's the turn of the opponent i.e., human

d. Algorithmic goat AI module

The algorithmic goat AI module basically generates a goat AI that uses the algorithmic implementations for predicting and carrying out the moves. Due to this the generated AI is harder to defeat. If all goats are placed, then evaluation for all goat instance are carried out and if not, a goat instance is selected in the decreasing order. This module checks the best move for the selected/allocated goat instances along with the evaluation value. The goat instance with the highest evaluation value is selected and the respective move is carried out. It two or more goat instances have the same evaluation value the goat instance the latter one selected. The checking of the move is carried out using negamax algorithm, alpha-beta pruning, Zobrist hashing and killer heuristic (see Appendix 5).

During the checking of the best move first of all killer moves are checked. If the killer moves are present and the condition for killer evaluation are fulfilled, that particular move is selected and is then carried out.

If the killer evaluation condition is not satisfied, the module then chooses a goat instance and then calls the negamax algorithm for that goat instance. Initially in the negamax method the module first checks the Zobrist hash table to check for any previously visited moves. If present the negamax method returns that particular value and move as the best move for the goat instance. if not, evaluation function is called for the child at depth 2. Evaluation function is carried out on all the child of the node generated until level 2 which is optimized using alpha-beta pruning. Once the evaluation value and the best move is determined, that values are stored in the Zobrist hash table at that particular hash value.

The move is carried out using the goat's initial location, the final valid move evaluated with the help of lerp() function (see Appendix 2).

Once the AI carries out its move it triggers a signal that says that the turn goat AI has finished its turn and mow it's the turn of the opponent i.e., human. Additionally, the modules contain functions that are invoked when the opponent has carried out its move and its turn is finished (which in this case is achieved using a signal). The invoked function calls the process of random selection of goat and moves.

e. Goat player module

The goat player module generates a goat player that the user can control and play with. The module allows the human user to use and control the goats such that the goat can play a game of baghchal against the tiger AI.

After the generation of the goat player, the module checks if all of the goats are placed in the board or not. After that the module checks for any kind of movement done by the user. Once the movement is done, it is checked whether the movement done is valid or not. It also checks whether the goat instance moved is valid or not (for e.g., all the goats present outside the board must first be placed inside the board and then only the goat can e moved from one position of the board to another). If both the goat instance and the move is valid, the modules carry out the movement (see Appendix 6).

The move is carried out using the goat's initial location (incase all goats are placed in the board), the final valid move carried out with the help of lerp() function (see Appendix 2).

Once the player carries out its move it triggers a signal that says that the turn goat player has finished it's turn and mow it's the turn of the opponent i.e., tiger AI. Additionally, the modules contain functions that are invoked when the opponent has carried out its move and its turn is finished (which in this case is achieved using a signal). The invoked function calls the process of random selection of goat and moves

f. Tiger player module

The tiger player module generates a tiger player that the user can control and play with. The module allows the human user to use and control the tigers such that the tiger can play a game of baghchal against the goat AI.

After the generation of the tiger player, the module checks whether the tiger can eat a goat or not. If it can, it will carry out movement accordingly and if not, the module then checks for any kind of movement done by the user. Once the movement is done, it is checked whether the movement done is valid or not. If the move is valid, the module carries out the movement (see Appendix 7).

The move is carried out using the tiger's initial location, the final valid move carried out with the help of lerp() function (see Appendix 2).

Once the player carries out its move it triggers a signal that says that the turn tiger player has finished it's turn and mow it's the turn of the opponent i.e., goat AI. Additionally, the modules contain functions that are invoked when the opponent has carried out its move and its turn is finished (which in this case is achieved using a signal). The invoked function calls the process of random selection of tiger and moves.

g. Dropzone module

There are basically two purposes of the dropzone module:

a. To provide graphics generation for a board of baghchal. This means this module constructs the circles and lines and connects them forming ma board of baghchal where each circle determines a position in the baghchal board (node) and the line determines the path between two positions (links).
b. To define which player (tiger or goat) has entered has exited or is present in the position. This data is sent to another modules which provides them in which position are the players are currently located and where they are moved

(See Appendix 8)

h. Selection module

The selection module allows the player choose the level of difficulty and the type of player the user wants to play a game of baghchal with. It is basically a UI with multiple buttons that allows the user choose form different options and combination.

Additionally, the module allows the user to view the list of rules for both goat and tiger for baghchal game

i. Defeat case module:

This module checks whether the player is defeated or has won the game. It is present in another modules as a single function or a part of function which checks whether the associated player has won the game or is defeated. The winning condition for goat is the none of the tigers have any valid moves. Whereas winning condition for tiger is that 5 of the goats has been eaten. If the conditions have been met a scene is triggered indicating that the game has finished and the game has been won (see Appendix 9).

5.2 TESTING

5.2.1. Test Cases for Unit Testing

Unit testing includes the testing of each individual components present in the system. In order to carry out unit testing in the system, each of the system modules are tested. Unit testing was done to check whether the individual modules are properly working or not i.e., do they carry out their operations properly or not.

The following shows the results of unit testing performed in the system modules:

Table 1: Unit testing for Easy tiger AI module

s.n.	Initial Position of the AI	Is there a goat that can be eaten	Position of the goat	Final position of the AI	Is the move valid
1	Zone1	no	null	Zone2	yes
2	Zone4	no	null	Zone5	yes
3	Zone19	no	null	Zone13	yes
4	Zone13	yes	Zone19	Zone25	yes
5	Zone23	no	null	Zone24	yes
6	Zone15	yes	Zone10	Zone5	yes

Table 2: User testing for Algorithmic tiger AI module

s.n.	Initial position	Is goat in eatable position (if yes what is the position)	Final position	Where was the final position accessed from	Is the move valid??
1	Zone5	no	Zone10	Negamax evaluation	yes
2	Zone8	no	Zone2	Negamax evaluation	yes
3	Zone5	no	Zone10	Zobrist hashing	yes
4	Zone13	no	Zone2	Negamax evaluation	yes
5	Zone1	no	Zone2	Negamax evaluation	yes
6	Zone2	Zone7	Zone12	Killer heuristic	yes

Table 3: User testing for easy goat AI module

s.n.	Are all the goat placed in the board	Initial position	Final position	Is the move valid?
1	no	null	Zone2	yes
2	yes	Zone5	Zone10	yes
3	no	null	Zone1	yes
4	yes	Zone7	Zone13	yes
5	yes	Zone15	Zone24	yes

6	no	null	Zone23	yes

Table 4: User testing for algorithmic goat AI module

s.n.	Are all goat placed in the board	Initial position	Final position	Where was the final position accessed from	Is the move valid??
1	no	null	Zone2	Negamax evaluation	yes
2	yes	Zone24	Zone25	Negamax evaluation	yes
3	no	null	Zone5	Negamax evaluation	yes
4	yes	Zone2	Zone1	Negamax evaluation	yes
5	no	null	Zone7	Negamax evaluation	yes
6	yes	Zone23	Zone24	Negamax evaluation	yes
7	yes	Zone7	Zone8	Negamax evaluation	yes

Table 5: User testing for goat player module

s.n.	Are all goat placed	Initial position	Final position	Was the movement accepted/rejected	Was the acceptation/rejection valid?
1	no	null	Zone5	accepted	yes

2	no	null	Zone16	accepted	yes
3	yes	Zone7	Zone9	rejected	yes
4	no	null	Zone23	accepted	yes
5	yes	Zone25	Zone20	rejected	yes
6	yes	Zone1	Zone1	accepted	yes

Table 6: User testing for tiger player module

s.n.	Initial Position	Is there a goat that can be eaten (if yes give the position)	Final position	Is the move accepted or rejected	Is the acceptance/rejection valid
1	Zone1	no	Zone2	accepted	yes
2	Zone4	no	Zone10	rejected	yes
3	Zone19	no	Zone12	rejected	yes
4	Zone13	Zone19	Zone25	accepted	yes
5	Zone23	no	Zone1	rejected	yes
6	Zone15	Zone10	Zone5	accepted	yes

5.2.2. Test Cases for System Testing

System testing is done to check the overall efficiency of the system. System testing is done to test whether the system does the work efficiently or not. In this project system testing is done to check whether the created computer AI accurately predicts the best moves for victory or not. This means that the created computer AI gives the human user a tough competition and compels the human to use its brain to win the game of baghchal

The following table shows test cases for system testing

Table 7: System testing for baghchal game

s.n.	Human player	Opponent AI	Who won??	Level of difficulty while playing the game
1	tiger	Easy goat AI	human	easy
2	tiger	Easy goat AI	human	easy
3	tiger	Easy goat AI	human	easy
4	tiger	Easy goat AI	human	easy
5	tiger	Algorithmic goat AI	human	medium
6	tiger	Algorithmic goat AI	human	medium
7	tiger	Algorithmic goat AI	human	medium
8	tiger	Algorithmic goat AI	AI	hard
9	goat	Easy tiger AI	human	easy
10	goat	Easy tiger AI	human	easy

11	goat	Easy tiger AI	human	easy
12	goat	Easy tiger AI	human	easy
13	goat	Algorithmic tiger AI	AI	hard
14	goat	Algorithmic tiger AI	human	hard
15	goat	Algorithmic tiger AI	AI	medium
16	goat	Algorithmic tiger AI	human	medium

5.3. RESULT ANALYSIS

The test results shows that the individual modules operate properly each of them efficiently carrying out their allocated operation. These modules independently operate without any chance of error. The algorithmic AI generated provides a decent job of predicting the most efficient move (move that takes the AI near to the winning state) during the game of baghchal. While the easy AI carries out allowed moves at random, the algorithmic implementation of the AI properly and efficiently checks and evaluates the best moves on the current scenario. The implementation is simpler and hence the system opponent is not as strong as it should be. The AI is easier to defeat then the other ones but nonetheless the whole gaming experience was fun and enjoyable. Even though the implemented AI is termed as comparatively weak, it was still hard to defeat at times, giving the user a sort of challenge.

Overall, the game even though having a comparatively weaker AI, the experience is fun and the game is a lot challenging. Each of the modules work perfectly well and the system as a complete program provides efficient working.

CHAPTER 6: CONCLUSION AND FUTURE RECOMMENDATION

6.1 CONCLUSION:

Minimax algorithm is the most used algorithm when it comes to implementing strategic games. But according to our research, negamax algorithm is a good and relatively more effective alternative for minimax. Negamax, when combined with alpha-beta pruning works as much effectively as the combination of minimax-alpha-beta pruning implementation if not better. On combining the existing implementation with Zobrist hashing (for transposition table implementation) and killer heuristic (making more effective move to be execute before). The final product becomes more effective for determining the more effective move and reduces any stress to the system regarding computational power and computational time

With the help of excessive research and utilization of different programming methodologies, we have been able to implement the decision-making algorithms and its various optimization techniques as described above. We found the implementation of these techniques helps create a competitive AI that efficiently predicts the next move for the AI such that the AI can win the game of baghchal. Similarly, our implementation has resulted simpler AI which results in the reduction in the execution time and computational power since only the allowed moves of the root node is accessed for evaluation of the best move and not the whole board combination.

The implementation even though is not that complex and results in reduction of computational time and power the created AI is not as strong as any AI should be. The AI is relatively easier to defeat and the moves it predicts to be effective is not that effective. Also, the implementation narrows the view of the AI.

Even with some shortcomings of the final product, the product is an efficient implementation of negamax algorithm, alpha-beta pruning, Zobrist hashing and killer heuristic in the game of baghchal to create an intelligent AI.

6.2 FUTURE RECOMMENDATION

The constructed AI is simple in complexity and hence is less difficult to tackle. So, there is a room for improvement and hence some future. Some of the possible recommendation for the future can be listed as follows:

a. The implementation of the algorithm can be done in such a way that the combination of components in the whole board are evaluated rather than the allowed moves of the root node.
b. An actual AI can be trained using any of the supervised or unsupervised training methods rather than using the normally used decision making algorithms and optimization techniques
c. Similar implementations can be done in other games such as shogi and checkers

REFERENCES

[1] Wikipedia, "Strategic game," [Online]. Available: https://en.wikipedia.org/wiki/Strategy_game.

[2] J. Point, "Minimax algorithm in ai," [Online]. Available: https://www.javatpoint.com/mini-max-algorithm-in-ai.

[3] S. overflow, "MIn max algorithm advantages and disadvantafes," [Online]. Available: https://stackoverflow.com/questions/36892813/minimax-algorithm-advantages-disadvantages.

[4] P. ai, "alpha beta pruining," [Online]. Available: https://www.professional-ai.com/alpha-beta-pruning.html.

[5] ludoteka, "Bagh chal," [Online]. Available: https://www.ludoteka.com/clasika/bagh-chal-en.html.

[6] Wikipedia, "Bagh-chal," [Online]. Available: https://en.wikipedia.org/wiki/Bagh-Chal.

[7] Chess.com, "Chess Engine," [Online]. Available: https://www.chess.com/terms/chess-engine.

[8] S. blogs, "Mastering the Nepali board game of Bagh chal with self-learning AI," [Online]. Available: https://soyuj.com.np/blog/mastering-bagh-chal-with-self-learning-ai#prior-work-on-bagh-chal.

[9] ChessProgramming, "Negamax algorithm," [Online]. Available: https://www.chessprogramming.org/Negamax.

[10] Wikipedia, "Negamax," [Online]. Available: https://en.wikipedia.org/wiki/Negamax.

[11] JavaPoint, "Alpha beta purning," [Online]. Available: https://www.javatpoint.com/ai-alpha-beta-pruning.

[12] G. f. geeks, "Minimax algorithm in game-theory set-4 Alpha beta Pruning," [Online]. Available: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/.

[13] Wikipedia, "negamax," [Online]. Available: https://en.wikipedia.org/wiki/Negamax.

[14] G. f. geeks, "Minimax algorithm in game theory set-5- Zobrist hashing," [Online]. Available: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-5-zobrist-hashing/.

[15] Wikipedia, "Killer Heuristic," [Online]. Available: https://en.wikipedia.org/wiki/Killer_heuristic.

[16] S. overflow, "Implementing Killer Heuristic in Alpha-Beta Search in Chess," [Online]. Available: https://stackoverflow.com/questions/17692867/implementing-killer-heuristic-in-alpha-beta-search-in-chess/17706147#17706147.

[17] G. engine, "gdscript basics," [Online]. Available: https://docs.godotengine.org/en/stable/tutorials/scripting/gdscript/gdscript_basics.html.

[18] Wikipedia, "GDScript," [Online]. Available: https://en.wikipedia.org/wiki/Godot_(game_engine)#GDScript.

[19] G. f. geeks, "Minimax Algorithm in Game Theory | Set 5 (Zobrist Hashing)," [Online]. Available: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-5-zobrist-hashing/.

APPENDICES

Appendix 1:

#Randomly selecting a tiger from all available tigers

```
func tiger_selection():
	var n = get_child_count()
	var rng = RandomNumberGenerator.new()
	rng.randomize()
	var rand = rng.randi_range(0, n)
	var child = parent.get_child(rand)
	defeat_case()
	move_child(child,0)
```

#Moving the easy tiger AI while determining whether the move is valid or not

```
func move():
	var n = get_parent().get_parent().allowed_moves[initial_position].size()
	var rng = RandomNumberGenerator.new()
	rng.randomize()
	var rand = rng.randi_range(0, n-1)
	final_position= get_parent().get_parent().allowed_moves[initial_position][rand]
	final_location = get_parent().get_parent().gameplay_position[final_position]
	initial_location = get_parent().get_parent().gameplay_position[initial_position]
```

```
if get_parent().get_parent().item_in_zone[final_position][0] == null:
        move_Position(final_position)
elif get_parent().get_parent().item_in_zone[final_position][0] == "tiger":
        final_position = initial_position
        final_location = get_parent().get_parent().gameplay_position[final_position]
elif get_parent().get_parent().item_in_zone[final_position][0] == "goat":
        if initial_location.x == final_location.x:
                if initial_location.y > final_location.y:
                        var new_Location=Vector2(final_location.x,final_location.y-100)
                        set_newPosition(new_Location)
                elif initial_location.y < final_location.y:
                        var new_Location=Vector2(final_location.x,final_location.y+100)
                        set_newPosition(new_Location)
        elif initial_location.y==final_location.y:
                if initial_location.x > final_location.x:
                        var new_Location = Vector2(final_location.x-100,final_location.y)
                        set_newPosition(new_Location)
                elif initial_location.x < final_location.x:
                        var new_Location = Vector2(final_location.x+100,final_location.y)
                        set_newPosition(new_Location)
        else:
```

```
                if initial_location.x < final_location.x and initial_location.y > final_location.y:
                    var new_Location = Vector2(final_location.x+100,final_location.y-100)
                        set_newPosition(new_Location)
                elif initial_location.x < final_location.x and initial_location.y < final_location.y:
                    var new_Location = Vector2(final_location.x+100,final_location.y+100)
                        set_newPosition(new_Location)
                elif initial_location.x > final_location.x and initial_location.y > final_location.y:
                    var new_Location = Vector2(final_location.x-100,final_location.y-100)
                        set_newPosition(new_Location)
                elif initial_location.x < final_location.x and initial_location.y < final_location.y:
                    var new_Location = Vector2(final_location.x+100,final_location.y+100)
                        set_newPosition(new_Location)
func set_newPosition(new_Location):
    for i in get_parent().get_parent().gameplay_position:
        if get_parent().get_parent().gameplay_position[i] == new_Location:
            var new_Position = i
            if get_parent().get_parent().item_in_zone[new_Position][0] == null:
                move_Position(new_Position)
```

```
func move_Position(final_pos):
	if get_parent().get_parent().item_in_zone[final_position][0] == "goat":
		var goat = get_node(get_parent().get_parent().item_in_zone[final_position][1])
		goat.queue_free()
		get_parent().goat_eaten = get_parent().goat_eaten+1
		if get_parent().goat_eaten == 5:
			get_parent().goat_eaten = 0
			get_tree().change_scene("res://Main Project/Levels and interface/Tiger won.tscn")
		get_parent().get_parent().item_in_zone[initial_position][1] = null
	get_parent().get_parent().item_in_zone[initial_position][0] = null
	initial_position = final_pos
	get_parent().get_parent().item_in_zone[final_pos][0] = "tiger"
	location = get_parent().get_parent().gameplay_position[initial_position]
	selected = false
	get_parent().player1 = false
	emit_signal("dragandfropturnfinished")
```

Appendix 2:

#Moving the player using lerp() function. Here the player can be either the human player or the computer AI

```
func _physics_process(delta):
        if get_parent().player1==true:
            move()
            global_position = lerp(global_position, final_location,50*delta)
        else:
            global_position = lerp(global_position, location,10*delta)
```

Appendix 3:

#Tiger selection while implementing negamax, killer heuristic and Zobrist hashing algorithm

```
func tiger_selection():
    var child
    final_position = null
    var is_killer = killer_evaluation()
    if is_killer == false:
        tiger_value = 0
        final_moves = [null,null,null,null]
        which_tiger = [-1000,null,null]
```

```
            hash_value = hash_indexing()
            for tiger in self.get_children ():
                var tiger_location = tiger.get_position()
                tiger_location = tiger_location.round()
                for pos in get_parent().gameplay_position:
                    if get_parent().gameplay_position[pos] == tiger_location:
                        var tiger_position = pos
                        root_position = tiger_position
                        var negamax_val = negamax(tiger_position)
                 zorbrist_table[hash_value] = [tiger_position,final_position,negamax_val]
                        final_moves[tiger_value] = final_position
                        if negamax_val >= which_tiger[0]:
                    which_tiger = [negamax_val,tiger_value,final_moves[tiger_value]]
                tiger_value = tiger_value+1
            child = parent.get_child(which_tiger[1])
            final_position = which_tiger[2]
    else:
            child = parent.get_child(which_tiger)
    if goat_eaten >= 5:
            goat_eaten = 0
            get_tree().change_scene("res://Main Project/Levels and interface/Tiger won.tscn")
```

```
        if final_position ==null:
                print(" Tiger for AI algorithm Defeated true vayo")
                get_tree().change_scene("res://Main Project/Levels and interface/Goat won.tscn")
        print(final_moves)
        move_child(child,0)
```

#Negamax algorithm implementation

```
func negamax(position, depth = 0, alpha = -1000, beta = 1000, color = 1):
        for val in zorbrist_table:
                if hash_value == val and position == zorbrist_table[val][0] and depth == 2:
                        final_position = zorbrist_table[val][1]
                        #print("table use vayo")
                        return zorbrist_table[val][2]
        var max_val = -1000
        if depth == 3:
                return evaluation_function(position)
        else:
                for move in get_parent().allowed_moves[position]:
                        if get_parent().item_in_zone[move][0] ==null:
                                if move != root_position:
                                        max_val = max(max_val,-negamax(move,depth+1,-beta,-
alpha,-color))
```

```
                                        alpha = max(alpha, max_val)
                                        final_position = move
                                        if alpha>= beta:
                                            return alpha
                    else:
                        pass
        return max_val
```

#Evaluaton function implementation

```
func evaluation_function(pos):
    if get_parent().item_in_zone[pos][0] == null:
        return 5
    elif get_parent().item_in_zone[pos][0] == "goat":
        return 10
    else:
        return 0
```

#Killer evaluation implementation

```
func killer_evaluation():
    var tiger_num = 0
    for tiger in self.get_children ():
```

```
		var tiger_location = tiger.get_position()
		var tiger_position
		tiger_location = tiger_location.round()
		for pos in get_parent().gameplay_position:
			if get_parent().gameplay_position[pos] == tiger_location:
				tiger_position = pos
				root_position = tiger_position
				for i in range(0,killer_moves.size(),1):
					if killer_moves[i][0] == root_position:
						if get_parent().item_in_zone[killer_moves[i][1]][0] =="goat" and get_parent().item_in_zone[killer_moves[i][2]][0] ==null:
							final_position = killer_moves[i][2]
							which_tiger = tiger_num
							var goat_path = get_parent().item_in_zone[killer_moves[i][1]][1]
							var goat = get_node(goat_path)
							goat.queue_free()
							goat_eaten = goat_eaten+1
						get_parent().item_in_zone[killer_moves[i][1]][0] = null
							return true
		tiger_num = tiger_num+1
	return false
```

#Hash indexing implementation

```
func hash_indexing():
	var hash_val = 0
	for hash_VAL in get_parent().zones:
		if get_parent().item_in_zone[hash_VAL][0] == "goat":
			hash_val = hash_val ^ zorbrist_hashing[hash_VAL][1]
		elif get_parent().item_in_zone[hash_VAL][0] == "tiger":
			hash_val = hash_val ^ zorbrist_hashing[hash_VAL][0]
		else:
			pass
	return hash_val
```

Appendix 4:

#Selecting random goat using function

```
func goat_selection():
	if all_Placed<20:
		var n = parent.get_child_count()
		var child = parent.get_child(n-1)
		move_child(child,0)
	else:
		var n = parent.get_child_count()
```

```
        var rng = RandomNumberGenerator.new()
        rng.randomize()
        var rand = rng.randi_range(0, n-1)
        var child = parent.get_child(rand)
        move_child(child,0)
```

#Moving easy goat AI

```
func move():
    is_All_Placed = get_parent().all_Placed
    if  get_parent().all_Placed< 20 and is_placed == false:
        var n = get_parent().get_parent().gameplay_position.size()
        var rng = RandomNumberGenerator.new()
        rng.randomize()
        var rand = rng.randi_range(0, n-1)
        final_position= get_parent().get_parent().zones[rand]
        final_location = get_parent().get_parent().gameplay_position[final_position]
        if get_parent().get_parent().item_in_zone[final_position][0] == null:
            initial_position = final_position
            get_parent().get_parent().item_in_zone[final_position][0] = "goat"
            get_parent().get_parent().item_in_zone[final_position][1] = get_path()
            location = get_parent().get_parent().gameplay_position[final_position]
```

```
                get_parent().player2 =false
                get_parent().all_Placed = get_parent().all_Placed+1
                is_placed = true
                emit_signal("goataiturnfinished")
        else:
                move()
    elif is_All_Placed >= 20 and is_placed:
        var n = get_parent().get_parent().allowed_moves[initial_position].size()
        var rng = RandomNumberGenerator.new()
        rng.randomize()
        var rand = rng.randi_range(0, n-1)
        final_position= get_parent().get_parent().allowed_moves[initial_position][rand]
        final_location = get_parent().get_parent().gameplay_position[final_position]
        initial_location = get_parent().get_parent().gameplay_position[initial_position]
        if get_parent().get_parent().item_in_zone[final_position][0] == null:
                for i in get_parent().get_parent().gameplay_position:
                    if get_parent().get_parent().gameplay_position[i] == final_location:
                            var new_Position = i
                        if   get_parent().get_parent().item_in_zone[new_Position][0]
== null:
                get_parent().get_parent().item_in_zone[initial_position][0] = null
                                    initial_position = final_position
```

```
get_parent().get_parent().item_in_zone[final_position][0] = "goat"
                location = get_parent().get_parent().gameplay_position[initial_position]
                                                get_parent().player2 = false
                                                emit_signal("goataiturnfinished")
```

Appendix 5:

#Selecting goat instance

```
func goat_selection():
    var child
    var killer_move = killer_evaluation();
    if killer_move:
        var n = parent.get_child_count()
        child = parent.get_child(n-1)
        move_child(child,0)
    else:
        neighbouring_nodes = [null,null,null]
        neighbouring_values = [null,null,null]
        var present_negamax_value = null
        var max_negamax_value= -100
        if all_Placed<20:
            var n = parent.get_child_count()
```

```
                child = parent.get_child(n-1)
                for demo_move in get_parent().zones:
                        if get_parent().item_in_zone[demo_move][0]==null:
                                present_negamax_value = abs(negamax(demo_move))
                                if present_negamax_value > max_negamax_value:
                                        max_negamax_value= present_negamax_value
                                        final_move = demo_move
                        else:
                                pass
                move_child(child,0)
        else:
                final_move2 =null
                root_move = null
                var goat_value = 0
                var final_moves = [null,null,null,null]
                var which_goat = [-1000,null,null]
                for goat in self.get_children ():
                        var goat_location = goat.get_position()
                        goat_location = goat_location.round()
                        for pos in get_parent().gameplay_position:
                                if get_parent().gameplay_position[pos] == goat_location:
```

```
                                        var goat_position = pos
                                        demo_move = goat_position
                                        var negamax_val = negamax(goat_position)
                                        if negamax_val >= which_goat[0]:
                                            root_move = demo_move
                                    which_goat = [negamax_val,goat_value,final_move2]
                            goat_value = goat_value+1
                    child = parent.get_child(which_goat[1])
                    final_move2 = which_goat[2]
                    move_child(child,0)
```

#Negamax implementation with alpha-beta pruning

```
func negamax(position, depth = 0, alpha = -1000, beta = 1000, color = 1):
    var max_val = -1000
    if depth == 3:
        return evaluation_function(position)
    else:
        if all_Placed<20:
            neighbouring_nodes[depth] = position
            neighbouring_values[depth] =get_parent().item_in_zone[position][0]
            for move in get_parent().allowed_moves[position]:
```

```
                var repeat = false
                for i in range(0,depth):
                    if move == neighbouring_nodes[i]:
                        repeat = true
                if repeat==false :
                  max_val = max(max_val,-negamax(move,depth+1,-beta,-
alpha,-color))
                    alpha = max(alpha, max_val)
                    if alpha>= beta:
                        final_move2 = move
                        return alpha
                else:
                    pass
        else:
            for move in get_parent().allowed_moves[position]:
                if get_parent().item_in_zone[move][0] ==null:
                    if move != root_move:
                      max_val = max(max_val,-negamax(move,depth+1,-
beta,-alpha,-color))
                        alpha = max(alpha, max_val)
                        final_move2 = move
                        if alpha>= beta:
```

return alpha

else:

pass

return max_val

#Implemeting evaluation function

func evaluation_function(pos):

if neighbouring_values[1] =="goat" and neighbouring_values[2] == "tiger":

return 15

elif neighbouring_nodes[0]=="Zone1" or neighbouring_nodes[0]=="Zone5" or neighbouring_nodes[0]=="Zone21" or neighbouring_nodes[0]=="Zone25":

if neighbouring_values[0] == null:

return 10

elif neighbouring_values[1] =="goat" and neighbouring_values[2] == "goat":

return 7

elif neighbouring_values[1] =="goat" and neighbouring_values[2] == null:

return 8

elif neighbouring_values[1] =="tiger" and neighbouring_values[2] == "tiger":

return 0

elif neighbouring_values[1] =="tiger" and neighbouring_values[2] == "goat":

return 0

elif neighbouring_values[1] =="tiger" and neighbouring_values[2] == null:

```
            return 0
        elif neighbouring_values[1] ==null  and  neighbouring_values[2] == "tiger":
            return 4
        elif neighbouring_values[1] ==null  and  neighbouring_values[2] == "goat":
            return 5
        elif neighbouring_values[1] ==null  and  neighbouring_values[2] == null:
            return 9
```

#Evaluating killer moves

```
func killer_evaluation():
    for move in killer_moves:
        if get_parent().item_in_zone[move[0]][0] == null:
            if get_parent().item_in_zone[move[1]][0] == "goat" and get_parent().item_in_zone[move[2]][0] == "tiger":
                final_move = move[0]
                return true
        else:
            pass
    return false
```

Appendix 6:

#Implementing goat player movement

```
func _on_Area2D_input_event(viewport, event, shape_idx):

    if event.is_action("movement_click") and Input.is_action_just_pressed("movement_click") and not event.is_echo():

        selected = true;

    if event.is_action("movement_click") and Input.is_action_just_released("movement_click") and not event.is_echo():

        if get_parent().all_Placed<20 and not placed and is_in_area:

            selected = false;

            get_parent().player2 = false;

            location = get_parent().get_parent().gameplay_position[final_position]

            initial_position = final_position

            get_parent().get_parent().item_in_zone[final_position][0] = "goat"

            get_parent().get_parent().item_in_zone[final_position][1] = get_path()

            placed = true

            get_parent().all_Placed= get_parent().all_Placed+1

            emit_signal("demoaiturnfinished")

        elif get_parent().all_Placed >=20 and placed and is_in_area:

            selected = false;

            get_parent().player2 = false;

            for i in get_parent().get_parent().allowed_moves[initial_position]:
```

```
                    if i == final_position:

        get_parent().get_parent().item_in_zone[final_position][0] = "goat"

        get_parent().get_parent().item_in_zone[final_position][1] = get_path()
            get_parent().get_parent().item_in_zone[initial_position][0] = null
                get_parent().get_parent().item_in_zone[initial_position][1] = null
                        #position mane zone ko naam
                        initial_position = final_position
                    location = get_parent().get_parent().gameplay_position[final_position]
                        emit_signal("demoaiturnfinished")
                get_parent().player2 = true
            else:
                get_parent().player2 = true
```

Appendix 7:

#Implemeting tiger player movement

```
func _on_TigerArea_input_event(viewport, event, shape_idx):

    if event.is_action("movement_click") and Input.is_action_just_pressed("movement_click") and not event.is_echo():

        selected = true;

    if event.is_action("movement_click") and Input.is_action_just_released("movement_click") and not event.is_echo():
```

```
selected = false;
initial_location = get_parent().get_parent().gameplay_position[initial_position]
final_location = get_parent().get_parent().gameplay_position[final_position]
if goat_condition == true:
    var allowed;
    goat_location = goat_path.get_position()
    goat_location.x = round(goat_location.x)
    goat_location.y = round(goat_location.y)
    for i in get_parent().get_parent().gameplay_position:
        if get_parent().get_parent().gameplay_position[i] == goat_location:
            goat_position = i
    for i in get_parent().get_parent().allowed_moves[initial_position]:
        if get_parent().get_parent().gameplay_position[i] == goat_location:
            allowed = true
    if get_parent().get_parent().item_in_zone[final_position][0] == null and allowed ==true:
        if initial_location.x == goat_location.x:
            if initial_location.y > goat_location.y:
                if final_location.x == goat_location.x and final_location.y == goat_location.y-100:
                    eat_goat_movement()
                else:
```

```
                movement()
            else:
                if goat_location.x == final_location.x and final_location.y == goat_location.y+100:
                    eat_goat_movement()
                else:
                    movement()
        elif initial_location.y == goat_location.y:
            if initial_location.x > goat_location.x:
                if final_location.x == goat_location.x-100 and goat_location.y ==final_location.y:
                    eat_goat_movement()
                else:
                    movement()
            else:
                if final_location.x == goat_location.x+100 and goat_location.y == final_location.y:
                    eat_goat_movement()
                else:
                    movement()
        else:
            if initial_location.x < final_location.x and initial_location.y > final_location.y:
```

```
                    if final_location.x == goat_location.x+100 and
final_location.y == goat_location.y-100:
                        eat_goat_movement()
                    else:
                        movement()
                elif initial_location.x < final_location.x and
initial_location.y < final_location.y:
                    if final_location.x == goat_location.x+100 and
final_location.y == goat_location.y+100:
                        eat_goat_movement()
                    else:
                        movement()
                elif initial_location.x > final_location.x and
initial_location.y > final_location.y:
                    if final_location.x == goat_location.x-100 and
final_location.y == final_location.y-100:
                        eat_goat_movement()
                    else:
                        movement()
            elif initial_location.x < final_location.x and initial_location.y <
final_location.y:
                if final_location.x == goat_location.x+100 and
final_location.y == goat_location.y+100:
                    eat_goat_movement()
```

```
                                        else:
                                                movement()
                                else:
                                        pass
                        else:
                                selected = false
                                get_parent().player1 = true
                else :
                        for i in get_parent().get_parent().allowed_moves[initial_position]:
                                if i == final_position:
                          get_parent().get_parent().item_in_zone[final_position][0] = "tiger"
                        get_parent().get_parent().item_in_zone[initial_position][0] = null
                                        initial_position = final_position
                                        location                                        =
get_parent().get_parent().gameplay_position[final_position]
                                        get_parent().player1 = false
                                        emit_signal("tigerplayerturnfinished")
                        location = get_parent().get_parent().gameplay_position[initial_position]
                        get_parent().player1 = true
func eat_goat_movement():
        goat_path.queue_free()
        goat_condition = false
```

```
		get_parent().get_parent().item_in_zone[goat_position][0] = null
		get_parent().get_parent().item_in_zone[initial_position][0] = null
		get_parent().goat_eaten = get_parent().goat_eaten +1
		if get_parent().goat_eaten == 5:
			get_tree().change_scene("res://Main Project/Levels and interface/Tiger won.tscn")
		initial_position = final_position
		location = get_parent().get_parent().gameplay_position[final_position]
		get_parent().get_parent().item_in_zone[final_position][0] = "tiger"
		get_parent().player1 = false
		emit_signal("tigerplayerturnfinished")
func movement():
		for i in get_parent().get_parent().allowed_moves[initial_position]:
			if i == final_position:
				get_parent().get_parent().item_in_zone[final_position][0] = "tiger"
				get_parent().get_parent().item_in_zone[initial_position][0] = null
				initial_position = final_position
				location = get_parent().get_parent().gameplay_position[final_position]
				get_parent().player1 = false
				emit_signal("tigerplayerturnfinished")
		location = get_parent().get_parent().gameplay_position[initial_position]
		get_parent().player1 = true
```

Appendix 8:

#Dropzone implementation

```
func _draw():
        draw_circle(Vector2.ZERO,30,Color.pink)
func _on_dropzone_body_entered(body):
        # body.get_name() le body ko naam dinxa
        # get_name() le position ko naam dinxa
        if body.get_parent().get_name() =="Goat":
                body.is_in_area = true
                body.final_position = get_name()
        if body.get_parent().get_name() =="Tiger":
                body.final_position = get_name()
func _on_dropzone_body_exited(body):
        if body.get_parent().get_name() =="Goat":
                body.is_in_area = false
```

Appendix 9:

#An instance of defeat case implementation

```
func defeat_case():
        defeated = true
        for pos in get_parent().item_in_zone:
```

```
var tiger_position
if get_parent().item_in_zone[pos][0] == "tiger":
    tiger_position = pos
    for nodes in get_parent().allowed_moves[tiger_position]:
        if get_parent().item_in_zone[nodes][0] == null:
            defeated = false
            print(" Tiger for easy AI Defeated false vayo")
            return false
if defeated == true:
    print(" Tiger for easy AI Defeated True vayo")
    get_tree().change_scene("res://Main Project/Levels and interface/Goat won.tscn")
```

Log of visits to supervisor

Meeting no 1	
Topic	Project Proposal Discussion
Date of meeting	2078/10/18
Day of meeting	Tuesday
Starting time	1:44 PM
Ending time	2:30 PM

Meeting no 2	
Topic	Project Proposal Evaluation
Date of meeting	2078/10/24
Day of meeting	Monday
Starting time	2:00 PM
Ending time	3:30 PM

Meeting no 3	
Topic	Mid Term project defense
Date of meeting	2078/10/27
Day of meeting	Thursday
Starting time	1:00 PM
Ending time	3:00 PM

Meeting no 4	
Topic	Final project defense evaluation
Date of meeting	2079/01/03
Day of meeting	Saturday
Starting time	2:00 PM
Ending time	4:00 PM