

Snails Game Report (Artificial Intelligence)

Muhammad Sajid Hameed Khan (1802013)

November 2020

Abstract

Game development is very fascinating aspect of Computer Science world. It has many challenges on its path which makes it very interesting to work on. Nowadays AI based game are being developed on a large scale. These games help human to become expert after practicing on it. In this project, We made a "Snails" game in which there are two versions of game. One of them is "Human to Human" and other one is "Human to AI agent". Our major challenge was to provide user a good GUI and make the character (AI based) more intelligent. We used Arcade module ad Python in this project. We used the strategy that player occupying more spaces will be the winner.

Contents

1	Introduction	4
2	Game Overview	4
2.1	Versions	4
2.1.1	Human To Human	4
2.1.2	Human To AI agent	4
2.2	Game Rules and Goals	4
2.3	Game Requirements and Installation	5
2.4	How to Start the Game	5
2.5	Terminologies Used in Game	6
2.6	Assumptions and Constraints	6
2.7	User Interface	6
2.8	Illegal Moves	7
2.9	Winning Criteria	7
3	Literature Review	7
3.1	Movement	7
3.2	Decisions	7
3.3	AI Agent	8
3.4	Algorithms	8
3.4.1	Minimax Algorithm	8
3.4.2	Alpha Beta Pruning	9
3.4.3	Heuristics	9
4	METHODOLOGY AND IMPLEMENTATION	10
4.1	Views	10
4.2	Main Functions of Game Play	12
4.2.1	Constructor of StartGame	12
4.2.2	Representation at Back-end	13
4.2.3	On-draw()	13
4.2.4	Calculations()	13
4.2.5	HumanMove()	13
4.2.6	BotMove()	13
4.2.7	isConsecutiveMoveforHumanLeft()	14
4.2.8	isConsecutiveMoveforHumanRight()	14
4.2.9	Collision()	14
4.2.10	EvalBoard()	14
4.2.11	Minimax()	14
4.2.12	Heuristics()	15
4.2.13	isZeroPresent()	16
5	Conclusion	16
6	References	17

1 Introduction

Game development is very fascinating aspect of Computer Science world. It has many challenges on its path which makes it very interesting to work on. Nowadays AI based game are being developed on large scale. These games help humans to become expert after practicing on it. In this project, We made a "Snails" game in which there are two versions of game. One of them is "Human to Human" and other one is "Human to AI agent". Our major challenge was to provide user a good GUI and make the character (AI based) more intelligent. We used Arcade module ad Python in this project. We used the strategy that player occupying more spaces will be the winner.

2 Game Overview

Snails game is two players game. Each player tries to get more scores than the other player. To get more score players have to occupy more places in grid. Illegal moves and slippery surfaces are available to make the game more interesting.

2.1 Versions

This game has two versions, one is human to human and second is human to AI agent. This was decided at the start of project that our game will have two versions. Firstly we implemented the human to human versions and then modified first version to Human to AI agent by making AI agent more and more intelligent.

2.1.1 Human To Human

In human to human version, both players are human and they select their move and compete with each other. Both human players have to play manually by selecting their moves and GridSquares.

2.1.2 Human To AI agent

In this version of game, human have to play with the AI agent designed by developers. AI agent is actually an intelligent player and play the optimal moves every time.

2.2 Game Rules and Goals

- The main object of game is to move around and occupy more place on the GridSquare to get more scores while trying to minimize the score of opponent.
- One more objective is to provide users a good Graphical User Interface.

- Each player can move only in Horizontal and vertical directions. if player move to diagonal, it will be illegal move.
- Each player can move onto consecutive block , moving into non-consecutive block is considered as illegal move.
- Player can also move onto its own trail or slime by slipping.
- A player can not move onto the opponent's trail or slime.
- And clicking out side the GridWorld is illegal.
- For any illegal move, player can be penalized.

2.3 Game Requirements and Installation

- This game is designed and developed using **Arcade Library**.
- Version of Python language 3.8
- Once the platform is installed, you can start the game to play.

2.4 How to Start the Game

We were using VSCode during the development of game. To run this game, you have to place the Folder of game as work space in VSCode. And then type "python Gamefile.py" in the terminal and it will run. This is also shown below:

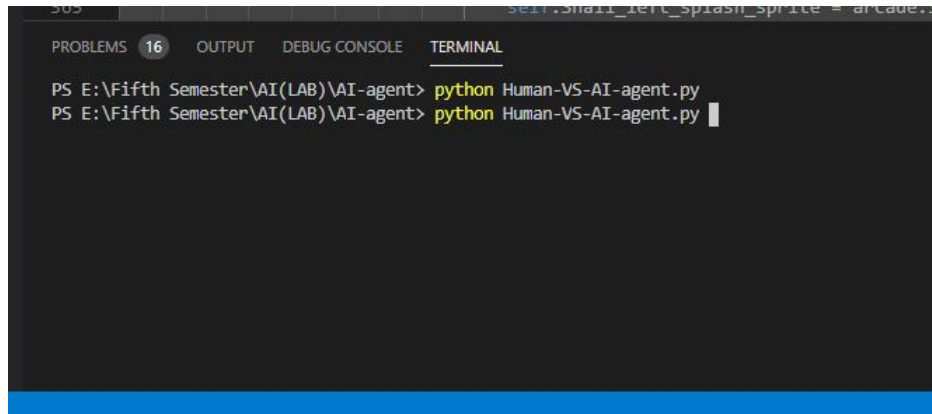
A screenshot of a Visual Studio Code (VS Code) terminal window. The terminal is dark-themed with a blue header bar. The header bar contains tabs for 'PROBLEMS' (with a count of 16), 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active. The terminal shows two lines of text: 'PS E:\Fifth Semester\AI(LAB)\AI-agent> python Human-VS-AI-agent.py' and 'PS E:\Fifth Semester\AI(LAB)\AI-agent> python Human-VS-AI-agent.py'. The second line is followed by a white cursor. The background of the terminal window shows some code from a file named 'Human-VS-AI-agent.py'.

Figure 1: Terminal Command

2.5 Terminologies Used in Game

We were using different terminologies during the development of game. All these terminologies are given below:

- **GridSquare** is single cell in grid shown at front-end. It is the part of Graphical User Interface(GUI). We are using the Grid of 10x10.
- **Board** is 2D list at back-end. It helps us to keep track of Players movements.
- **Trail or Slime or Splash** is the mark placed when a player move forward and leave is spot at previous position. Each Snail has its own slime or splash.
- **Slippery Surface** is also available in this game. It means that when a snail moves to its own splash , it will slip from current position to the last splash present in that direction.
- **GridWorld** is the total area of GridSquare.
- **Sprites** is a two dimensional image that is part of the larger graphical scene.
- **AI agent** is artificially intelligent player.

2.6 Assumptions and Constraints

There are some assumptions and constraints in the game given below:

- We are using the Grid of 10x10.
- During game development, we also using some assumption for representation like human player, AI player, their splashes, etc.
- Output window of the game re-sizable but Grid is static on the output window. Output window is of size 732x 1070.

2.7 User Interface

Once you start the game, you will see following interfaces. This game include three Interfaces.

- Instruction View
- Game Play View
- Result Screen View

2.8 Illegal Moves

During the game play, a player can be penalized for illegal moves or a foul. It may include following cases:

- Clicking out of the GridWorld
- Non-Consecutive Moves by player

2.9 Winning Criteria

In order to win the game, player need to cover more GridSquares in the grid. Player who cover the more GridSquares will be the winner, the one having highest score.

3 Literature Review

Literature review is considered as a road block in the professional career of a researcher. It is very compulsory when we start a new project to do some research on it. In this section, literature review of the techniques which we were using during the game development. On path of game development, there are many challenges and technical issues that faced to by the developer, which also makes it interesting to work on it. In computer science world, there is, nowadays, task to develop games on AI based. Humans can practice enough on it by playing with a artificially intelligent player. AI based game's main purpose is to make game character enough intelligent so that it can compete with Human. There are some techniques which can be put together to make player artificially intelligent.

3.1 Movement

Movement is the important feature for our game because player have to move around the Grid to occupy GridSquares. This is low level technique and it is necessary because we have to move the player on grid.

3.2 Decisions

In this game AI player has to decide smartly that where to move at very turn. Decision taken by AI agent are intelligent. There might some condition in the game, that game reaches the state in which there are some GridSquares are left but both players can not move into that place. In that condition, we have to observe that if there is no continuous change in score of any player, we will decide the winner on the base of current scores.

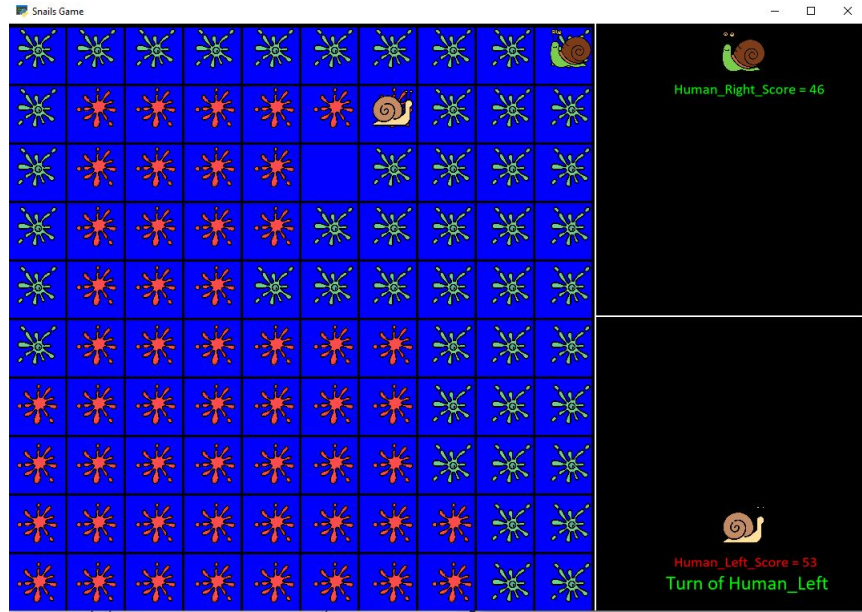


Figure 2: Decision

3.3 AI Agent

An AI agent can be developed by using some specific heuristics and techniques. Our purpose was to make the AI agent as intelligent as possible for us so that it can compete with the human. In this way game becomes more interesting.

3.4 Algorithms

The major algorithms we were using during the game development are given below:

3.4.1 Minimax Algorithm

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.

Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by

some heuristics which are unique for every type of game. A game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level.

- Maximizer goes LEFT: It is now the minimizers' turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3
- Maximizer goes RIGHT: It is now the minimizers' turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

And then the game tree looks like:

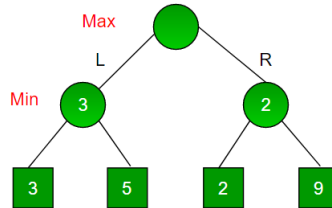


Figure 3: Minimax

3.4.2 Alpha Beta Pruning

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.

3.4.3 Heuristics

Heuristics helps us to make our AI agent an intelligent player. In minimax we have to traverse the all possibilities which is very time consuming. that's why we used heuristics in the implementation of our AI agent. And the implementation of heuristic function is described in the methodology section.

4 METHODOLOGY AND IMPLEMENTATION

In order to efficiently structure, plan, and control the process of proposed game development, Agile based development methodology is applied. In this section, full detail of our work is given with detail, for methodology, techniques and algorithms which we used for implementation of the game.

4.1 Views

To provide a good graphical user interface to users, we used different views in snails game. Views is class in arcade library, which provide better interface.

- Instruction View
- Game Play View
- Result View



Figure 4: Instruction View

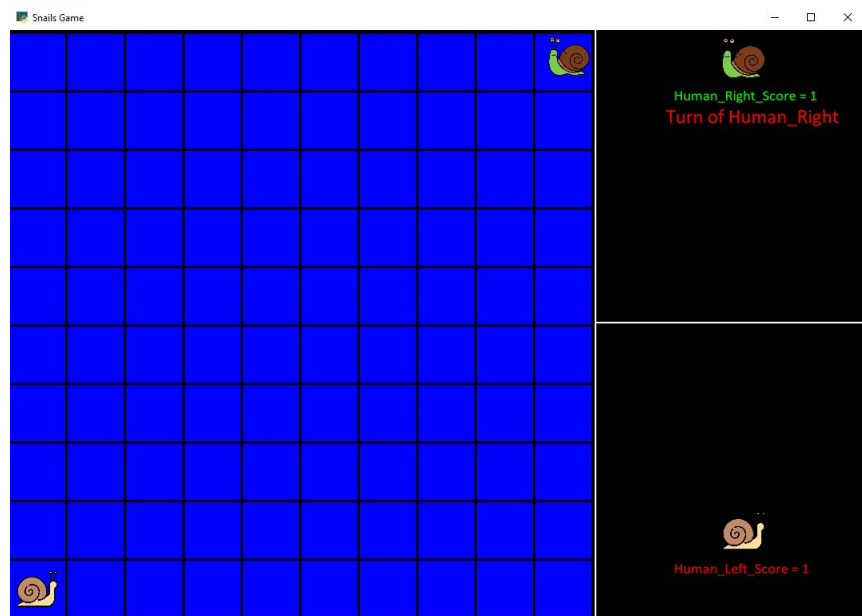


Figure 5: Game Play-1

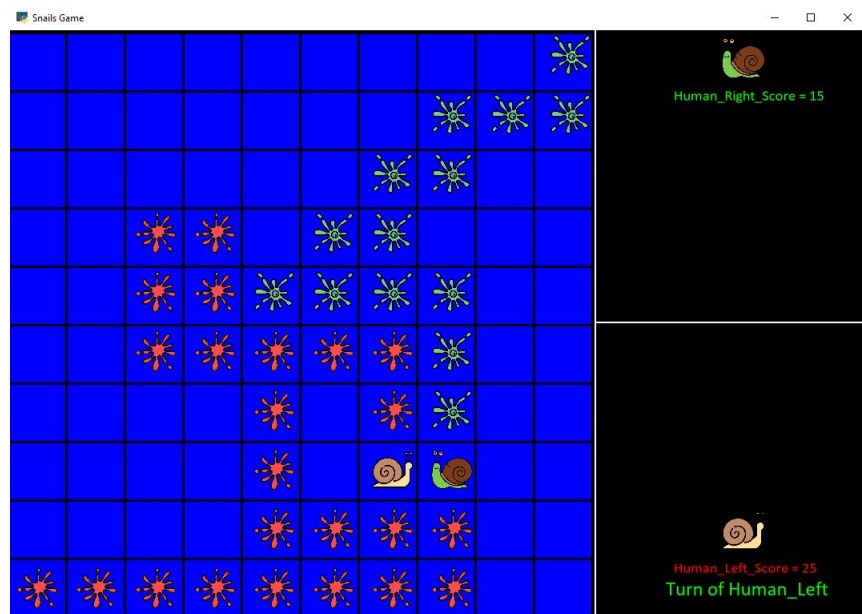


Figure 6: Game Play-2

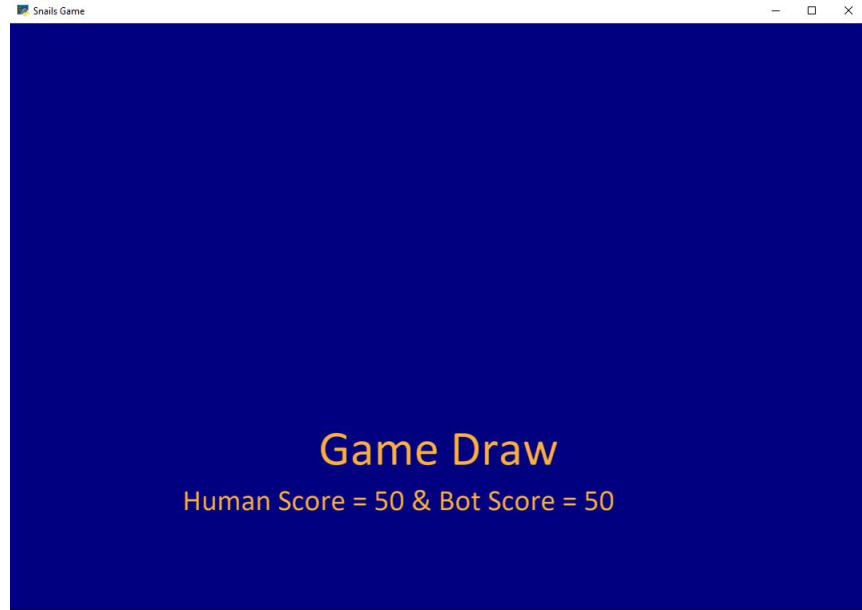


Figure 7: Result View

4.2 Main Functions of Game Play

There is a class named as **StartGame** which handles with the game play and performs all things during the game. Two more classes, one named as **InstructionMenu** which shows the instruction view to user, and the second one is **GameOver** which shows the Result view of game that whether the game is win and lose by which player or the game is draw.

4.2.1 Constructor of StartGame

When object of StartGame is made, default constructor is called and following class attributes are created which will be used during game play.

- self.Board ===== Back-end grid (10x10)
- self.status = 0 ===== For status of game that whether continue, win,lose, draw
- self.Snail-left-list ===== To store sprites of Human
- self.Snail-right-list ===== To store sprites of AI agent

- self.Snail-left-splash-list ===== To store splash of Human
- self.Snail-right-splash-list ===== To store splash of AI agent

4.2.2 Representation at Back-end

There were some back-end representations which decide during the planning activity before implementation of the game. These representations helps us to manipulate the back-end board. All representations are given below:

- Human player ===== 1
- Human player's slime ===== 10
- AI agent or Second Human player ===== 2
- AI agent or second Human player's slime ===== 20

4.2.3 On-draw()

On-draw function draw the grid on front end and update the grid every time when move is played. All lists of sprites are drawn on the grid in this function.

4.2.4 Calculations()

This function is getting the x and y axes every when clicked on the screen and gives the coordinates of desired GridSquare in grid. And then human and AI agent play their moves by calling humanMove() botMove() respectively.

4.2.5 HumanMove()

This function is a large function which plays the human as the human wishes to be played. If a move is illegal, human have to be penalized in this case and no score in this case. More in this **slippery surface** is also implemented. Up, down , left and right slips are implemented through coding in this function. Using slips, a player can move onto its own splash or slime until the last slime reached in that direction.

4.2.6 BotMove()

This function is designed for AI agent to play its move intelligently by using minimax algorithm and Alpha Beta pruning and some heuristics. AI agent suppose that human player is also playing the optimal move every time and decide to make intelligent moves. Its also supports the feature of slippery surface.

4.2.7 isConsecutiveMoveforHumanLeft()

This function checks that the move played by human is consecutive or not. If move is consecutive, it's good; otherwise, it's an illegal move.

4.2.8 isConsecutiveMoveforHumanRight()

This function is to check move of AI agent. This function was actually for Human to Human version of game. In AI agent Vs Human, AI agent always makes intelligent moves so will automatically be consecutive moves.

4.2.9 Collision()

In this function, we are checking the collision of sprites because we want to place only one sprite at one GridSquare to provide a good GUI. If there are more than one sprites in one GridSquare, then it removes the bottom sprites.

4.2.10 EvalBoard()

This function takes a board as a parameter and evaluates it and tells us that whether which status is currently on the board. This function simply returns the status of board.

```
def evaluateBoard(self, board):
    Zero = self.is_zero_present(board)
    if Zero == 1:
        return 0 # Continue state
    else:
        if board.count(10) + 1 == board.count(20) + 1:
            return 50 # Draw state
        elif board.count(10) + 1 >= board.count(20) + 1:
            return 100 # Human Win
        else:
            return 200 # Bot Win
```

Figure 8: Evaluation of Board

4.2.11 Minimax()

This function, implementation of minimax algorithm, check for the best possible move of AI agent. And this function also include Alpha Beta pruning. Alpha beta pruning helps us to reduce the computational time by a large factor.

```

def minimax(self, board, depth, isAgentTurn, alpha, beta):
    if depth == 10:
        return
    win, draw, lose = 200, 50, 100
    result = self.evaluate_board(board)
    if (result == win or result == draw or result == lose):
        bestScore = result
        bestChild = board
        return
    if isAgentTurn:
        best = -MIN
        for w in range(Rows):
            for h in range(Columns):
                if board[h][w] == 0:
                    board[h][w] == 20
                    val, R1, C1 = self.minimax(board, depth + 1, True, alpha, beta)
                    best = max(best, val)
                    alpha = max(alpha, best)
                    R, C = w, h
                    if beta <= alpha:
                        break
            return best, R, C
    else:
        best = MAX
        for w in range(Rows):
            for h in range(Columns):
                if board[h][w] == 0:
                    board[h][w] == 10
                    val, R1, C1 = self.minimax(board, depth + 1, False, alpha, beta)
                    best = min(best, val)
                    beta = min(beta, best)
                    R, C = w, h
                    if beta <= alpha:
                        break
            return best, R, C

```

Figure 9: Code Snippet for Minimax

4.2.12 Heuristics()

In heuristic function, we have to decide the intelligent move for AI agent. We used our technique to make intelligent moves. As the goal of each player is to occupy more GridSquares, therefore, we moved the AI agent to the vacant places. We calculated the chances of movement from current position to all other directions (horizontal leftward and rightward and vertical upward and downward). And where chance were maximum, we moved AI agent to that direction. Implementation of heuristics function is given below:

```

def heuristics(self,board):
    # Move Chance for every direction
    self.MoveChanceRight , self.MoveChanceLeft = 0,0
    self.MoveChanceUp , self.MoveChanceDown = 0,0
    # x and y coordinates of AI agent
    self.x = self.Previous_Backend_grid_row_H2
    self.y = self.Previous_Backend_grid_col_H2
    # count the MoveChanceDown
    for a in range(9, 0, -1):
        if self.y-a <= 9:
            if board[a - 1][self.y] == 0:
                self.MoveChanceDown += 1
            elif board[a - 1][self.y] == 1 or board[a - 1][self.y] == 10:
                break
        else:
            break
    # count the MoveChanceLeft
    for b in range(9, 0, -1):
        if self.x-b <= 8:
            if board[self.x][b - 1] == 0:
                self.MoveChanceLeft += 1
            elif board[self.x][b - 1] == 1 or board[self.x][b - 1] == 10:
                break
        else:
            break
    # count the MoveChanceRight
    for d in range(0,9):
        print(f'd={d}')
        if self.y+d<=8:
            if board[self.x][self.y+d] == 0:
                self.MoveChanceUp += 1
            elif board[self.x][self.y+d] == 1 or board[self.x][self.y+d] == 10:
                break

```

Figure 10: Code Snippet for Heuristics-Part1

```

        else:
            break
    # count the MoveChanceUp
    for c in range(0,9):
        if self.y+c <= 9:
            if board[self.x][self.y+c] == 0:
                self.MoveChanceRight += 1
            elif board[self.x][self.y+c] == 1 or board[self.x][self.y+c] == 10:
                break
        else:
            break
    self.maxChances = max(self.MoveChanceRight, self.MoveChanceLeft, self.MoveChanceUp, self.MoveChanceDown)
    if self.maxChances == self.MoveChanceRight:
        return self.x,self.y+1
    elif self.maxChances == self.MoveChanceLeft:
        return self.x, self.y-1
    elif self.maxChances == self.MoveChanceUp:
        return self.x+1,self.y
    elif self.maxChances == self.MoveChanceDown:
        return self.x-1,self.y

```

Figure 11: Code Snippet for Heuristics-Part2

4.2.13 isZeroPresent()

This function is used in the evaluation of board. Its check that is there any zero (0) present in the board. If a single zero is present in board, then it returns True otherwise False.

5 Conclusion

Finally after spending a lot of time we were able to reach some results for the game. And in this project we learned to develop the AI based games using some libraries. Project was properly planned and agile process flow was followed

during the working of project. In this journey, we faced many issue during the implementation and we learned first time that how to resolve them. We used heuristics and many other algorithms in this development of game. After this project, we gained some confidence to work in the world of game development.

6 References

- www.flaticon.com
- www.youtube.com
- "Alpha Beta pruning and Minimax" from www.geeksforgeeks.com