



Namal Institute Mianwali

SNAIL GAME

Muhammad Omair

BSCS-2019-58

Tariq Shehzad

BSCS-2019-12

Fareeha Bano

BSCS-2019-23

Kiran Zafar

BSCS-2019-36

Acknowledgement:

We wish to express our sincere gratitude to **Sir Faisal Zeeshan** for helping in game development using Artificial Intelligence. Furthermore, we sincerely thank their guidance and encouragement in carrying out this project work and helping us to put the label of "Scientist" outside of our box.

Abstract:

AI-Game creation is an essential component of Artificial Intelligence. Today's gaming industry is dominated by AI-based games. AI agents will be utilized to play games with machines or computers in this case. An AI agent will discover and follow a conceptual flow. The AI agent strives to defeat the opponent or win the game. This paper focuses on AI Agent-based gaming. AI gaming aids in the development of psychosomatic abilities. Making a computer/program intelligent such that it can move like a person. AI Agents may be built using a variety of techniques such as path finding, trees, and heuristic functions. To move an AI agent, this project employs minimax and heuristic functions. In this project, an AI bot is created to compete with humans. AI agents should be sophisticated enough to compete with humans. The AI bot might aim to prevent the opponent from scoring or it can strive to score the most points possible. The AI agent's primary goal is to find the best answer. To make the optimal decision, A.I Agent will think like something of a human brain.

Table of Contents:

Contents

Acknowledgement:	2
Abstract:	3
1. Introduction:	6
2. How to Play Game?	6
2.1 Game overview:	6
2.2 Requirements:	6
2.2.1 Python with Arcade setup:	6
2.2.2 Images:	6
2.3 Start a New Game:	7
2.3.1 Welcome Screen:	7
2.3.2 Game Environment:	7
2.3.3 Game Play Instructions:	8
2.4 Game Rules & Regulations:	8
2.5 Restrictions:	8
3. Literature Review:	8
3.1 Movement:	8
3.2 Path Finding:	8
3.3 Decision Making:	8
3.4 AI Agent:	9
3.5 MINIMAX:	9
3.6 Heuristic:	9
4. Methodology & Implementation:	9
4.1 Core Procedures	9
4.1.1 Arcade Library	10
4.1.2 On_Mouse_Press ()	10
4.1.3 StuckCondition ()	10
4.1.4 Initialization	10
4.1.5 On Draw ()	11
4.1.6 Slip Function ()	11
4.1.7 A.I Agent move ()	14
5. Testing:	14
6. Result & Analysis:	16

7. Conclusion:	16
----------------------	----

1. Introduction:

Artificial intelligence is the process of making a machine or software intelligent in the same way that people are. Improving the machine's ability to think like a human. AI is the study of human cognitive processes. Artificial intelligence is becoming increasingly significant in the game business. Many games require AI or bot-based gaming. AI-assisted game design is becoming increasingly popular. This report will go through how we implemented AI in our snail game.

2. How to Play Game?

2.1 Game overview:

We created 2p game (human vs human) by initializing array-backed 10x10 grid. The next step was to add AI agent as another player with human player. This was a tricky experience.

2.2 Requirements:

2.2.1 Python with Arcade setup:

The basic step of snail game development was arcade installation which is a built in python library for creating 2d games with fascinating graphics and sound. So, to play a game you should have any python IDE with arcade installed in it. You can download python by following link given below:

[Download Python | Python.org](https://www.python.org/)

Enter the following command in python terminal for arcade setup:

Pip install arcade

2.2.2 Images:

Some images are used in a game as sprites, splashes representation and for background. Before playing a game, make sure that you have all these images in a single directory.

2.3 Start a New Game:

2.3.1 Welcome Screen:

After the successful installation of arcade and python, you can run .py file to play a game. The game will be start with welcome screen as shown below:



Figure 1: Welcome Screen

2.3.2 Game Environment:

By pressing any key, we moved to game environment having two sprites at their fixed starting positions. One at the top right corner and other at the bottom left corner as shown below:

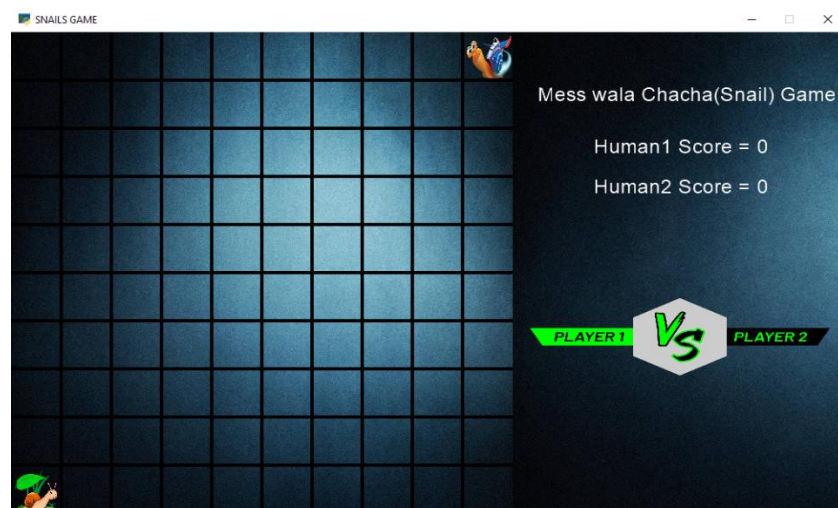


Figure 2: Main Game Environment

2.3.3 Game Play Instructions:

- By pressing the left key of the mouse on the boxes next to snail, you can move your snail.
- If a player click on his own splash next to him, He would be moved to the first splash on that route.

2.4 Game Rules & Regulations:

Some rules and regulations are given below:

- One player can't take more than one turn at a time.
- One each turn, one score will be added to the player scores.
- To win a game, player has to occupy more empty grids.
- One player can't click on the other player's splash.

2.5 Restrictions:

Besides rules and regulations game have also some restrictions which are given below.

- One player will win if he has gain more scores than his opponent player otherwise it will lose game.
- In case of equal scores of both players, game will withdraw.

3. Literature Review:

3.1 Movement:

The movement of players and their behaviours are fundamental elements of AI-based games. Players can move in such a way that their actions have no effect on the game's rules. Distinct sorts of AI games, such as chess, include different types of movements for each player's king or sprite. In a chess game, for example, the crown, unlike the knight and queen, cannot move.

3.2 Path Finding:

During the game, both the human and the bot must establish a path that will allow them to defend against the opponent's attacks. Similarly, for the victory state, the player might find an offensive path. AI Agents must be very clever in order to determine the correct path to take and to be able to modify the course in different game scenarios. As in AI-based games, your sprite should not become stuck at any point so that it may attack.

3.3 Decision Making:

Players' movements can be claimed to be fixed. How they are able to move and how they are not able to move. However, when it comes to AI-based agents, the idea will shift at various points, such as the agent must make judgments. The AI bot must decide whether to continue attacking or to go into defence mode.

3.4 AI Agent:

The agent could be anything that identifies the environment via sensors and acts on it via actuators. Agents cycle via perception, thinking, and action. AI agents are gamers who can move in the same way that humans do. AI agents behave inside their own environment and against other agents (people).

3.5 MINIMAX:

The minimax calculation is exceptionally famous for showing AI specialists how to play turn-based system games. The explanation being is that it considers every one of the potential moves that players can require some investment during the game. With this data, it then, at that point, endeavours to limit the adversary player's benefit while boosting the specialists every step of the way the AI specialist gets to play. There are two player mina and maxi they play for their own viewpoints.

3.6 Heuristic:

Video games are no longer just for fun and pleasure; they also play an essential part in education, the military, medicine, and business. The increasing number of mobile phones is the cause for the expansion of mobile games. Heuristics are solutions-finding shortcuts. The concept of heuristic approaches in artificial intelligence is linked to cognitive sciences, or the research of how people think. Heuristics are used by humans all the time to make judgments and solve issues. Similarly, heuristic algorithms are frequently employed in AI to instruct a computer to discover an approximate answer rather than a precise solution.

4. Methodology & Implementation:

This is the most important part of a program or software development. In methodology methods are constructed against every process or component of a program. Then every component or function is tested and implemented in actual program. A single mistake can violate the whole functioning of a software program. That's why every programmer needs more attention towards this step.

4.1 Core Procedures

Our snail game is working on two boards. One board is graphical in grids where both snails Human and Bot can move on their turns. Second board is working at the backend of the graphical board in form of matrix values. Every time a turn is played and score is either added to the scoreboard of that player or deducted on regulating rules or the other possibility is that nothing is added instead player perform slip operation. These all functionalities are performed in the code via some functions. Some of the important functions, procedures and methodologies are discussed

below:

4.1.1 Arcade Library

Arcade is an Object-Oriented Library. We used this to draw grids of visual board, to draw window of game view. Already discussed in section 2.2.1.

4.1.2 On_Mouse_Press ()

Similarly, like key press, Mouse press is working. Whenever a click is performed on the mouse a turn is played and a value is generated against every click in the matrix board.

4.1.3 StuckCondition ()

Stuck condition occurs when a player is stuck somewhere between its own splashes and opponent splashes. Player does not have any empty path to play its turn smoothly.

4.1.4 Initialization

We used a built-in function 'init function' to initialize different values. Specifically, initialization is done for Board, Turns, Player Scores, Win state, Scores record.

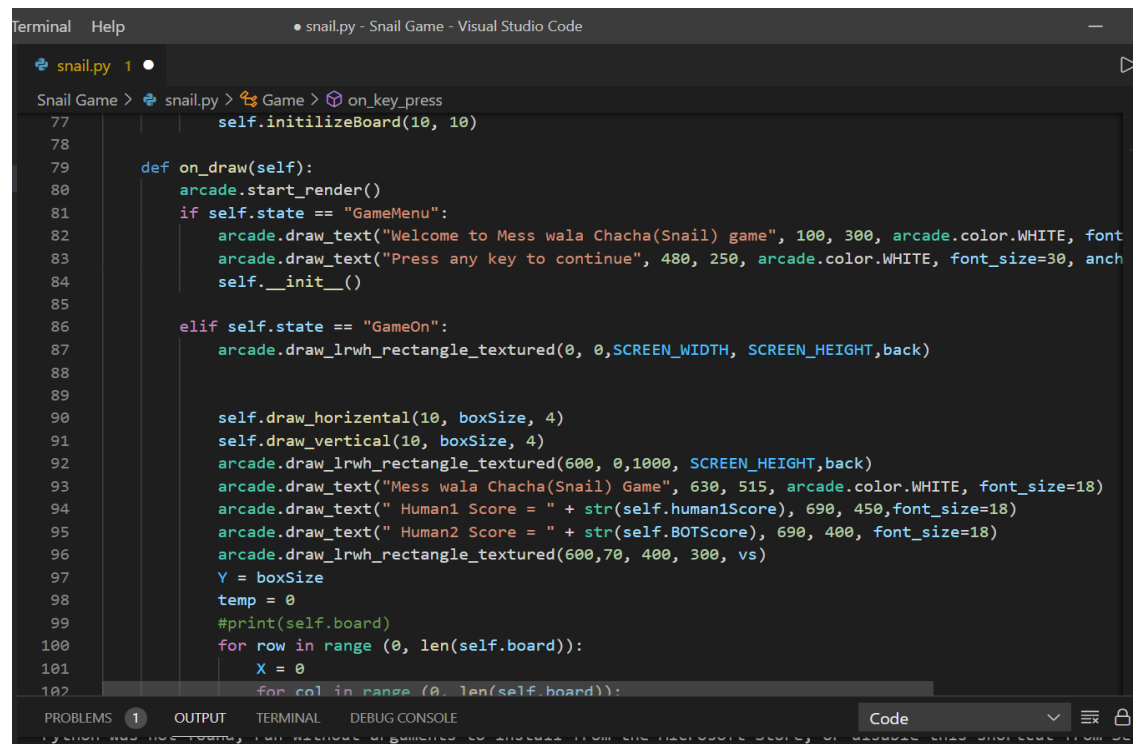
Initialize Board ()

In the actual program coding we used some classes of **arcade** like **views** to draw the board. Our's board is basically initialized with a grid size of (10*10) with starting value set to Zero. The backend board is a 2D array board. This backend board has initial values of players (1 for player1 & 2 for player2) and the rest of the values are all Zeros. These values change as players play their turns by mouse press.

0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0

4.1.5 On Draw ()

On Draw function is used for generating Game start view and game menu by using views of Arcade Library. All the visual graphics of snail game are created inside On Draw function.



```

77         self.initializeBoard(10, 10)
78
79     def on_draw(self):
80         arcade.start_render()
81         if self.state == "GameMenu":
82             arcade.draw_text("Welcome to Mess wala Chacha(Snail) game", 100, 300, arcade.color.WHITE, font
83             arcade.draw_text("Press any key to continue", 480, 250, arcade.color.WHITE, font_size=30, anch
84             self.__init__()
85
86         elif self.state == "GameOn":
87             arcade.draw_lrwh_rectangle_textured(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, back)
88
89
90             self.draw_horizontal(10, boxSize, 4)
91             self.draw_vertical(10, boxSize, 4)
92             arcade.draw_lrwh_rectangle_textured(600, 0, 1000, SCREEN_HEIGHT, back)
93             arcade.draw_text("Mess wala Chacha(Snail) Game", 630, 515, arcade.color.WHITE, font_size=18)
94             arcade.draw_text(" Human1 Score = " + str(self.human1Score), 690, 450, font_size=18)
95             arcade.draw_text(" Human2 Score = " + str(self.BOTScore), 690, 400, font_size=18)
96             arcade.draw_lrwh_rectangle_textured(600, 70, 400, 300, vs)
97             Y = boxSize
98             temp = 0
99             #print(self.board)
100             for row in range (0, len(self.board)):
101                 X = 0
102                 for col in range (0, len(self.board)):

```

4.1.6 Slip Function ()

Slip function works when a player clicks on its own splash then its snail will jump to all the splashed and stop at the end of splashes.

In our snail game player can perform slip operation in four directions. Left

Slip, Right Slip, Down Slip and Up Slip. In left right slip we will change column numbers and in up down slips we will change row numbers.

LSlip ()

```
terminal Help • snail.py - Snail Game - Visual Studio Code
snail.py 1 •
Snail Game > snail.py > Game > on_key_press
142
143 def LSlip(self, Player):
144     if Player == 2:
145         x, y = self.getBOTPosition()
146         x1, y1 = x, y
147         while(True):
148             if self.board[x][y] == 0 or self.board[x][y] == 22 or self.board[x][y] == 2:
149                 self.board[x1][y1] = 11
150                 self.board[x][y+1] = 1
151                 # self.turn = "human1"
152                 break
153             elif y == 0:
154                 self.board[x1][y1] = 11
155                 self.board[x][y] = 1
156                 # self.turn = "human1"
157                 break
158             else:
159                 y = y-1
160         elif Player == 1:
161             x, y = self.gethuman1Position()
162             x1, y1 = x, y
163             while(True):
164                 if self.board[x][y] == 0 or self.board[x][y] == 11 or self.board[x][y] == 1:
165                     self.board[x1][y1] = 22
166                     self.board[x][y+1] = 2
167                     # self.turn = "BOT"
168                     break
169                 elif y == 0:
170                     self.board[x1][y1] = 22
171                     self.board[x][y] = 2
```

RSlip ()

```
terminal Help • snail.py - Snail Game - Visual Studio Code
snail.py 1 •
Snail Game > snail.py > Game > on_key_press
178
179 def RSlip(self, Player):
180     if Player == 2:
181         x, y = self.getBOTPosition()
182         x1, y1 = x, y
183         while(True):
184             if self.board[x][y] == 0 or self.board[x][y] == 22 or self.board[x][y] == 2:
185                 self.board[x1][y1] = 11
186                 self.board[x][y-1] = 1
187                 # self.turn = "human1"
188                 break
189             elif y == 9:
190                 self.board[x1][y1] = 11
191                 self.board[x][y] = 1
192                 # self.turn = "human1"
193                 break
194             else:
195                 y = y+1
196         elif Player == 1:
197             x, y = self.gethuman1Position()
198             x1, y1 = x, y
199             while(True):
200                 if self.board[x][y] == 0 or self.board[x][y] == 11 or self.board[x][y] == 1:
201                     self.board[x1][y1] = 22
202                     self.board[x][y-1] = 2
203                     # self.turn = "BOT"
204                     break
205                 elif y == 9:
206                     self.board[x1][y1] = 22
207                     self.board[x][y] = 2
```

DSlip()

```
Terminal  Help  • snail.py - Snail Game - Visual Studio Code

snail.py 1 •
Snail Game > snail.py > Game > on_key_press

212 def DownSlip(self, Player):
213     if Player == 2:
214         x, y = self.getBOTPosition()
215         x1, y1 = x, y
216         while(True):
217             if self.board[x][y] == 0 or self.board[x][y] == 22 or self.board[x][y] == 2:
218                 self.board[x1][y1] = 11
219                 self.board[x-1][y] = 1
220                 # self.turn = "human1"
221                 break
222             elif x == 9:
223                 self.board[x1][y1] = 11
224                 self.board[x][y] = 1
225                 # self.turn = "human1"
226                 break
227             else:
228                 x = x+1
229         elif Player == 1:
230             x, y = self.gethuman1Position()
231             x1, y1 = x, y
232             while(True):
233                 if self.board[x][y] == 0 or self.board[x][y] == 11 or self.board[x][y] == 1:
234                     self.board[x1][y1] = 22
235                     self.board[x-1][y] = 2
236                     # self.turn = "BOT"
237                     break
238                 elif x == 9:
239                     self.board[x1][y1] = 22
240                     self.board[x][y] = 2
```

USlip()

```
Terminal  Help  • snail.py - Snail Game - Visual Studio Code

snail.py 1 •
Snail Game > snail.py > Game > on_key_press

245 def UPSlip(self, Player):
246     if Player == 2:
247         x, y = self.getBOTPosition()
248         x1, y1 = x, y
249         while(True):
250             if self.board[x][y] == 0 or self.board[x][y] == 22 or self.board[x][y] == 2:
251                 self.board[x1][y1] = 11
252                 self.board[x+1][y] = 1
253                 # self.turn = "human1"
254                 break
255             elif x == 0:
256                 self.board[x1][y1] = 11
257                 self.board[x][y] = 1
258                 # self.turn = "human1"
259                 break
260             else:
261                 x = x-1
262         elif Player == 1:
263             x, y = self.gethuman1Position()
264             x1, y1 = x, y
265             while(True):
266                 if self.board[x][y] == 0 or self.board[x][y] == 11 or self.board[x][y] == 1:
267                     self.board[x1][y1] = 22
268                     self.board[x+1][y] = 2
269                     # self.turn = "BOT"
270                     break
271                 elif x == 0:
272                     self.board[x1][y1] = 22
273                     self.board[x][y] = 2
```

4.1.7 A.I Agent move ()

In this function minimax will be called to find best move so that AI agent will move according to the best path extracted in minimax. Before moving to next AI agent will check all the invalid conditions to its left right and up down.

Minimax ()

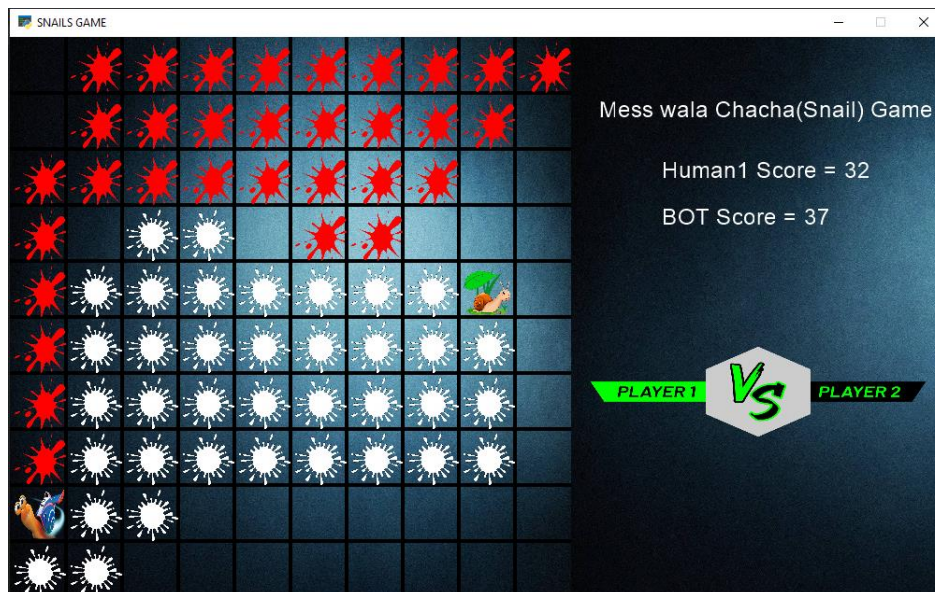
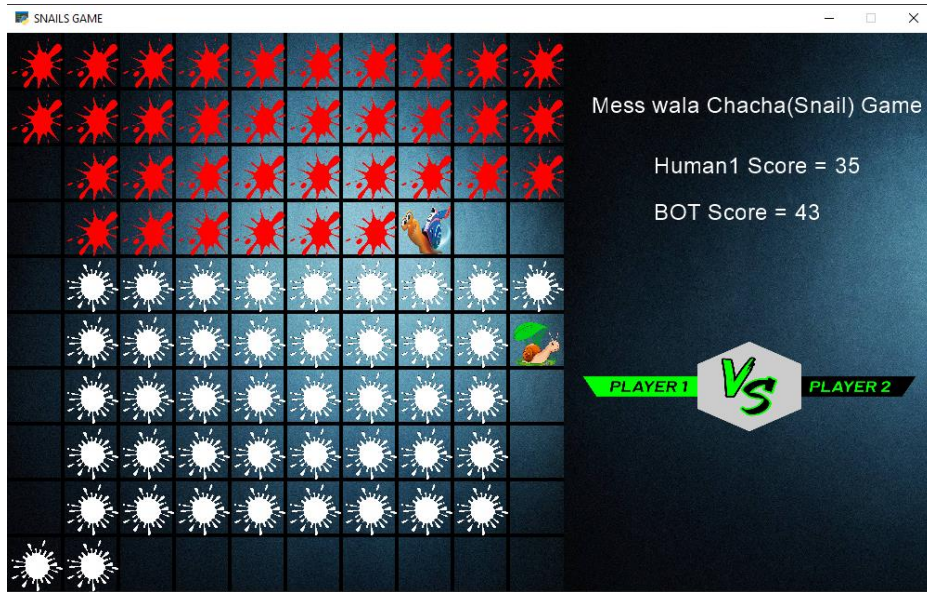
The Minimax function will invoke a heuristic function on each left, right, up, and down movement to determine the optimum winning score. A.I Agent will strive to act wisely based on this greatest winning score. For the best move, Minimax employs the heuristic function. Otherwise, Minimax will always select the ideal option, which will take too much time for a single move.

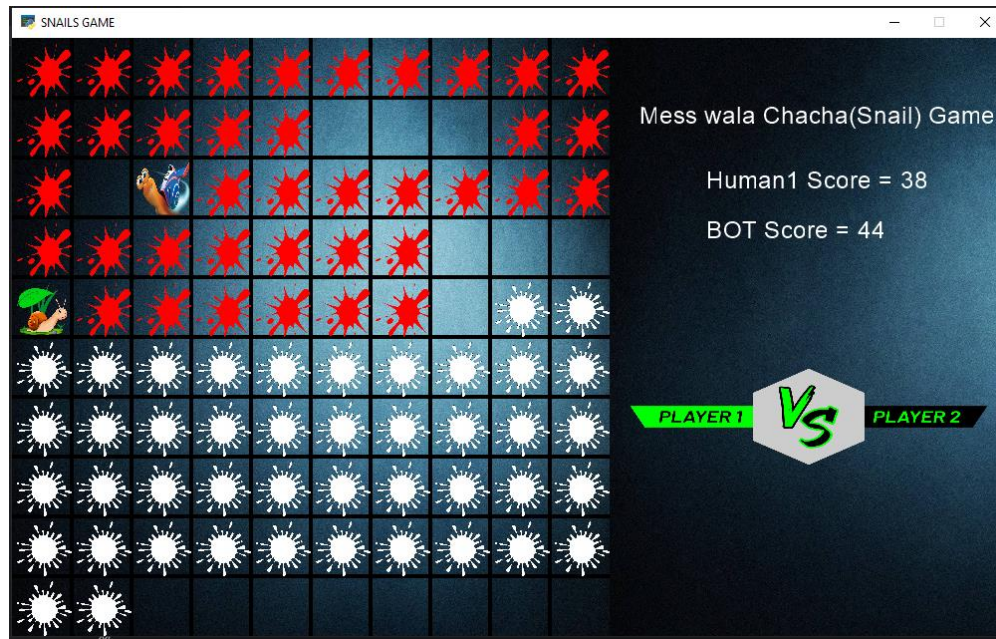
Heuristic ()

The heuristic function will examine the initial splashes of A.I Agent in the board before adding to the winning score. The heuristic function will be directed as to where to look for the greatest winning score for the A.I Agent. And if the A.I Agent is not on the board's walls, the Heuristic function will add some extra points to the winning score. Finally, the heuristic function will seek for the given number of empty boxes in the defined direction as supplied in the heuristic function's parameter.

5. Testing:

On testing it was found heuristic function tried best to compete with human player. Below some test cases just before winning state for A.I Agent.





6. Result & Analysis:

AI Agent did its best to compete with humans, but sometimes AI Agents do not survive competition with humans. However, in most cases, AI agents can easily beat humans. The most important role of the AI agent is the heuristic function, which always finds the best move to defeat the human player, and the heuristic function helps the minimax function to decide what action is best for the AI agent to defeat human players at any time in the game.

7. Conclusion:

We have discussed that how artificial intelligence is being used in game development by implementing two dimensional snail game. In addition, we talked about how we can start game? and what are the literature reviews?. In game, AI agent was used as second player. We have also seen that how AI agent always tried its best to compete human player and heuristic function which is the reason behind success as it looks for best move and helps minimax function to choose best paths.