2020

SNAILS Game

AI for Games

SUBMITTED TO: MR. FAISAL ZEESHAN

SUBMITTED BY:

M. REHAN ALI	**1802008**
SHAHZEB KHAN	**1802021**
SANA TARIQ	**1802014**

ABSTRACT

As we know today world is world of technology in this one field is of games. Today people usually play with the help of technology means video games. For this reason, we tried to make turn based 2D game. To follow trend that now this is era of AI based games. In this AI agent would compete with human with same intelligence and try to win. In this project, we developed and AI based game which we named as “Snails”. It is turn based two player game in which one is human player and other is computer (AI agent). This have been implemented in Python with arcade library for user interface.

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

Overview:

In this era there is strong competition between video games because it seems that now people spend most of their time on video games rather than domestic ones. Now its trend of machine-human games means machine will play with human with same intelligence and compete very wisely. But this is challenge for developers to make machine enough much intelligent so that it can compete human. Here AI comes to make machine intelligent as we know today AI is buzz word every man talking about AI in this project you will taste its one flavor that is AI for Games. We have developed a turn-based AI game which we give named as "Snails". Snails is two players turn based games. In which one player will be human and other will be Bot (AI agent). It is Grid Based game. Its winning criteria to occupy as many boxes as possible more than opponent. We will discuss its complete working later.

1.1 Game Rules

The rules to play this game and achieve goal are as follows:

1. In this player move boxes by boxes which are grid cells and try to occupy as more as possible.
2. On turn, player can move only one box adjacent to it. On its movement towards empty cell increase its score by 1.
3. Player can move only in four directions (left, right, up, bottom).
4. When player traversed on next adjacent empty cell, he will leave slimes behind it so to indicate occupied area.
5. Player can also move on his own slime. But in this case slimes are slippery it slides to farthest position until empty cell. In this case no score will be awarded.
6. Player cannot move onto opponent's slimes or opponent himself.
7. Player cannot play illegal moves. i.e. to move outside grid, move on grid lines, move on opponent slimes.
8. Player with highest occupied area will be winner.

1.2 Game Goals

Main objectives of game are followings:

1. Occupy more area to win the game.
2. Bound opponent in small region as quickly as possible.
3. Try to restrict opponent so that you can win easily.
4. One with Highest score means highest slimes will be winner.

1.3 Software Requirements

Programming Language: This game is design and implemented in Python3 a general-purpose programming language. You have to install python environment first.

Libraries: Arcade is used for make an interactive user interface for game. Arcade has many powerful functions to develop 2D games. We have used this one because it is easy to integrate with Python3.

1.4 Assumptions

These are some assumptions to achieve game goals:

- Board size should be 10x10.
- In backend, several assumptions have been made. e.g. players, slimes.
- Screen width is fixed and it is 750x620.
- User interface is integrated with Python by Arcade.

1.5 Game Terminologies

- Bot: AI agent (opponent)
- Player: Human
- Grid Cell: single box in grid
- Splash: occupied box

1.6 How to Start

Once environment setting is done. You have to run game file and then game will start.
When game will start an instruction screen will be displayed in **(figure 1)**:

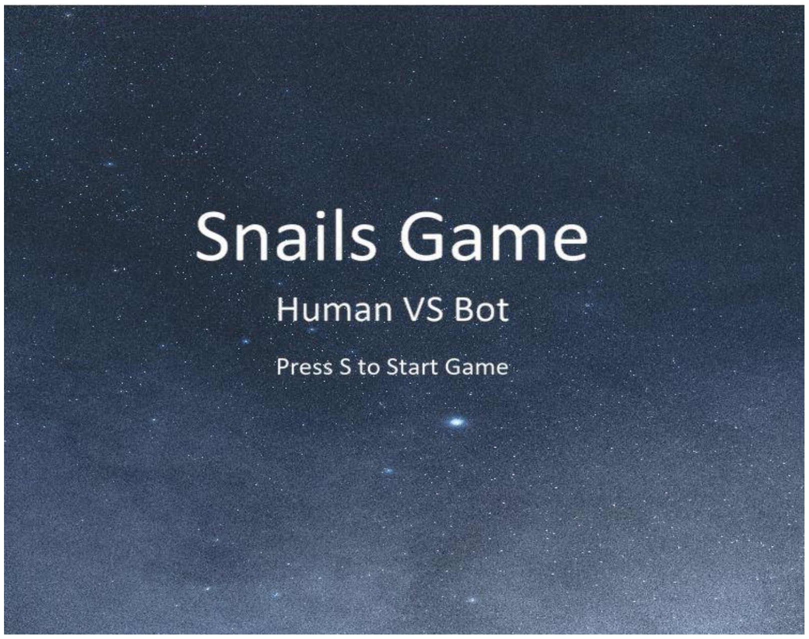


Figure 1: Instruction Screen

When you follow instruction and press accordingly then start state states of game will display in which scores of both player will zero and both players are its start states. At that time board will look like in *(figure 2).*

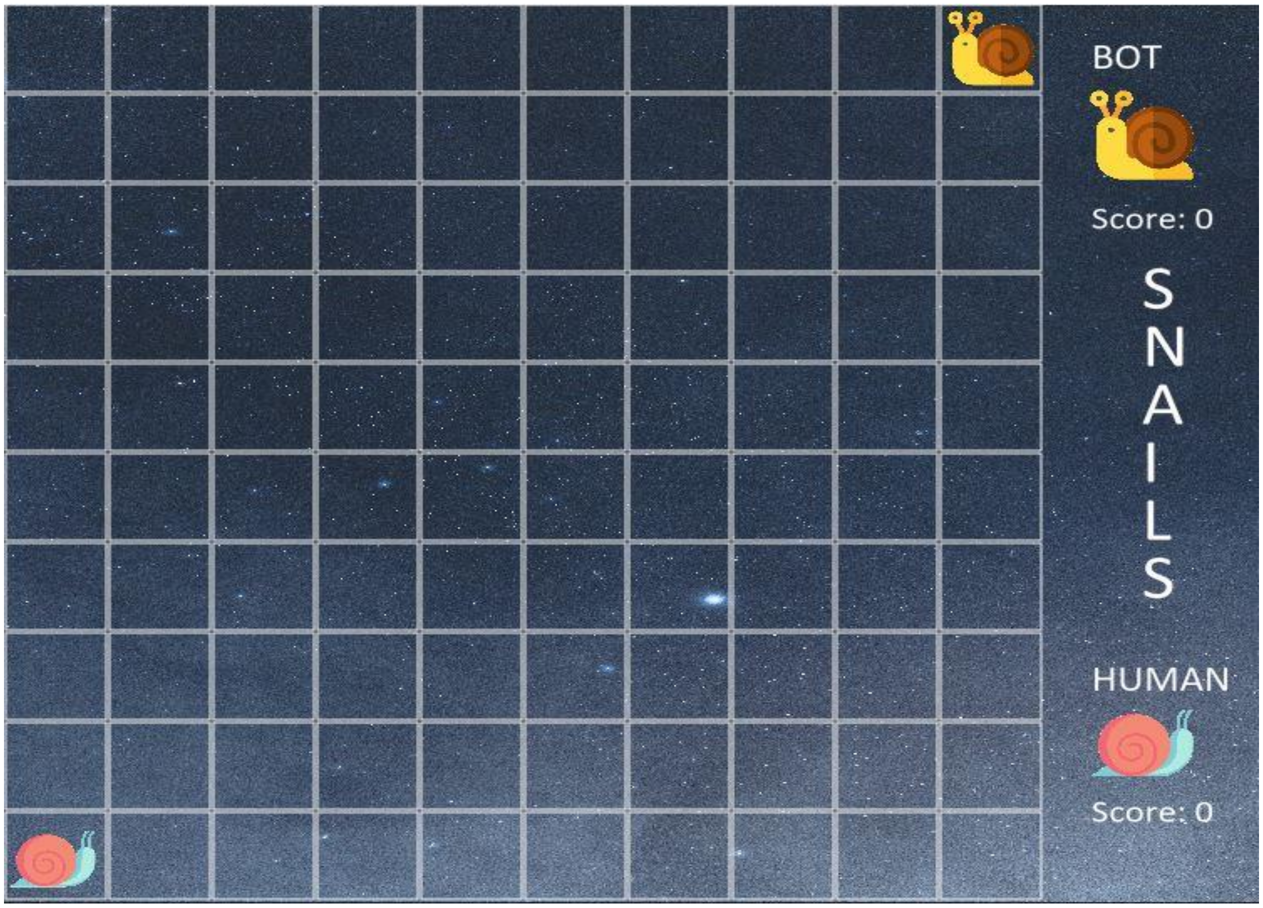


Figure 2: Initial States

Now you can start game by following rules of game and should try to win.

1.7 Movement Scenarios

During game player must be careful about moves either they are legal or not. There are some cases in which some are legal some illegal.

- Move to adjacent empty cell (not diagonal) is legal move.
- Move to his own adjacent slime is legal but no score will be awarded.
- Click on grid lines is illegal.
- Click area outside board is illegal.
- Click on opponent or opponent's slimes.

1.8 How to Win

To win the game you should follow these points:

- Occupy as more area as possible

- Try to restrict opponent in small region
- Player who will occupy more area than other he will be winner.
- Player who bound opponent in small region will be winner.

2. Literature Review

2.1 Decision Making

In games as player know on his turn, he has to do decision making that which move can be more helpful for me. During decision making player have to check your current position and opponent position and then player decide about future. During this process player have to keep in mind rules of game because he is restricted to follow game rules.

2.2 Minimax Algorithm

It is backtracking algorithm which is most widely used in AI for games. It is based on recursion a tree would generate and goes into deeper until terminal node and then backtrack to perform efficient task. It is beauty of that algorithm in this best move strategy used. Mostly used in 2 player games in which one will maximize score other will minimize. Max will maximize its utility and Min will minimize utility.

Its basic functionality is same as well-known phrase "keep yourself in opponent shoes". In this we assume that both players will play optimal move means both are intelligent. In this on every turn on every state it expands tree until end state and then chose best possible move. Minimax function can be best understood by making its tree as such tree you can see in (figure 3).

This algorithm is most widely used in 2 player games like chess, tic tac toe and snails etc.

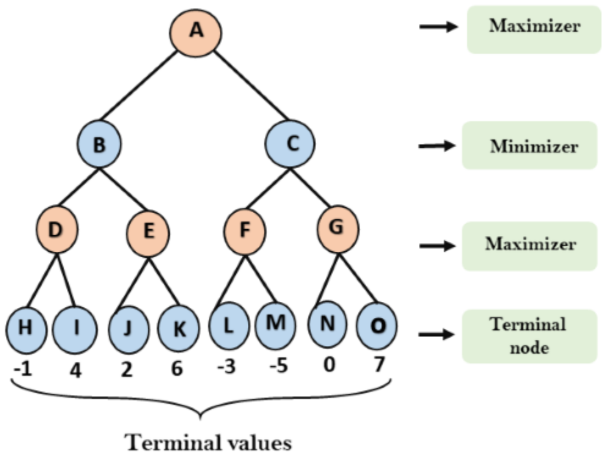


Figure 3: Minimax Tree

(javatpoint, n.d.)

Example:

Here in given example (figure 3) it can be seen that when maximizer's turn it goes into deep until terminal and decide which turn give him more profit. In given example maximizer would have 4 as maximum value although 7 is highest number this is beauty of this algorithm. It will also keep an eye on opponent that if I take this move opponent will take which move.

2.3 Alpha Beta Pruning

Alpha-Beta pruning is not 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. In this case two extra parameters will pass in Minimax that is Alpha and Beta. Alpha represent for maximizer and Beta indicate minimizer (Geeks for Geeks, n.d.).

In this we have to not check all branches we decide it from left to right it is descent algorithm it reduces time as compared to Minimax. It prunes some branches and save time and cut off that branch from tree.

Example:

Here is example of this descent algorithm that how it reduces computation and work with same efficiency. You can see in (figure 4) that how wisely it cuts off some branches and reduce computation.

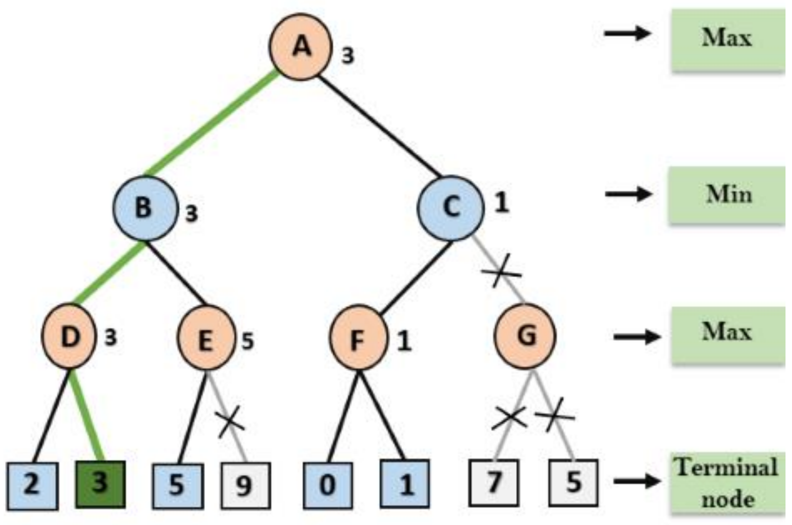


Figure 4: Alpha Beta Pruning

(javatpoint, n.d.)

2.4 Heuristic Search

The heuristic function is a way to inform the search about the direction to a goal. It provides an informed way to guess which neighbor of a node will lead to a goal.

It is technique designed to solve problem quickly and more efficiently. Heuristic search tries to solve problem in minimum number of steps. We have used heuristic to make game more efficient about time as well as performance its basic purpose is decide move quickly and efficiently.

Its challenge for developers to make heuristic function as perfect so that it can compete with human. Many techniques are used in different scenario in different games like heuristic for chess is different than tic tac toe but their functionality is same to get optimal solution. Without going till end state heuristic is performed at somewhere middle with same best solution. Moreover, as heuristic is good AI agent will play very well. Which heuristics techniques used in this game are discussed in detail in implementation chapter.

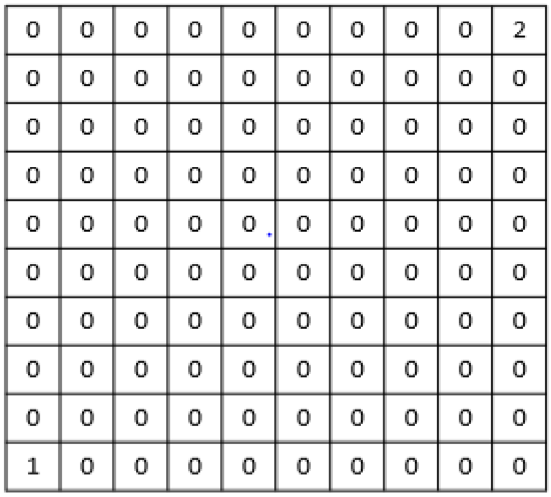
3. Implementation

3.1 Game Class

During development of game developers used OOP method because it is very easy to understand and easily can be updated later. Also keep in mind future OOP strategies are perfect. In this we done all functionality in class and named it as mygame which is inherited from arcade. View class. Later you will see detail of each functions.

3.2 Initialize Board

This function is used to initialize 2D list in backend against given dimensions that are 10 by 10. In backend we place some number to represent empty boxes. Also initialization of player places is done at this stage, some specific number placed at starting positions. Initially how's looks like board in backend can be shown in *(figure 5).*



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

Figure 5: Initial backend board

3.3 Backend Representation

Here developers set some value to indicate players or slimes on backend. There can be any value but for this game setting is that:

- 1: Human player
- 2: AI Agent
- 10: Human splash
- 20: Bot splash
- 0: Empty box

3.4 Initialize Grid ()

Functionality of this function two initialize frontend grid. It generates 2D frontend grid of given dimensions. Also, certain objects would place at starting positions that indicates players. Moreover, score of both players would set to zero. After calling this function with previous functions output can be shown in *(figure 2)*.

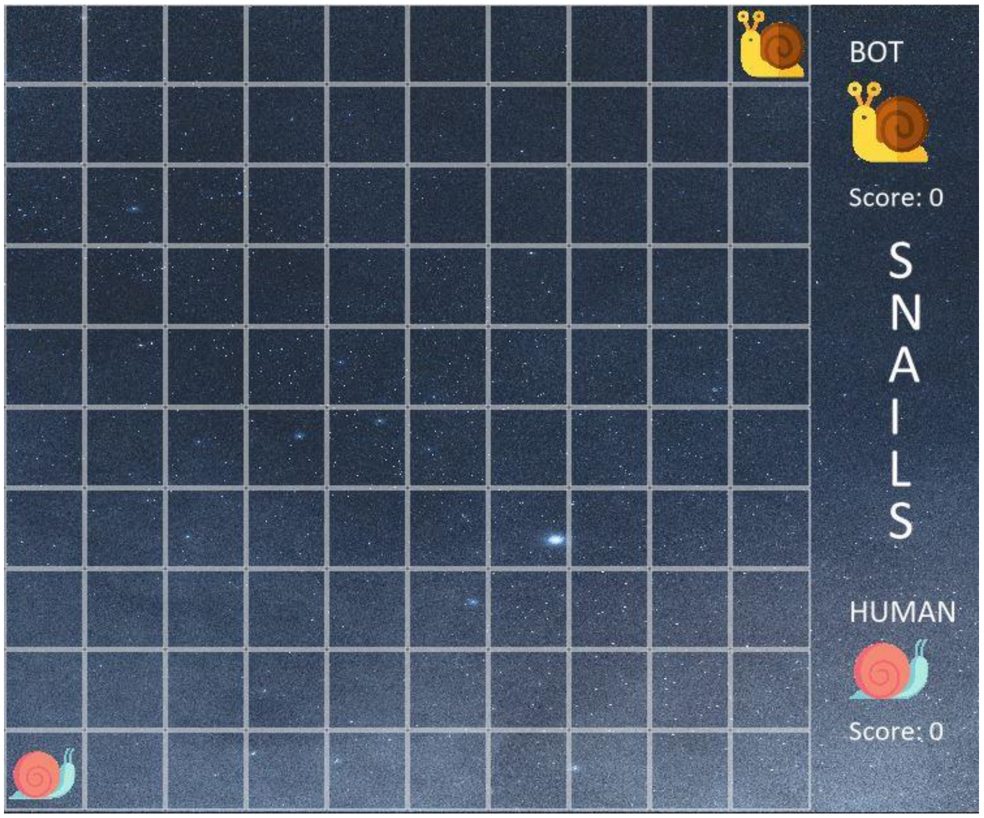


Figure 2: Initial States

3.5 On Draw ()

This function is built in Arcade Library basically this is used to render screen. Simply its work is that what else on screen show it. In this function three states are defined for three different scenarios. These three states are following.

- Game Menu: To display instruction screen as game starts will show.
- Game On: When players are playing to show their movement.
- Game Over: To show who is winner when game over.

3.6 Resyncing Board ()

This function is used to update grid after movement of player. Actually, when player moves change in backbend occur now, we have to show this change on frontend. This function generally works that. It took values in backend and show it on front according to related values. We can also name it as update grid means update frontend.

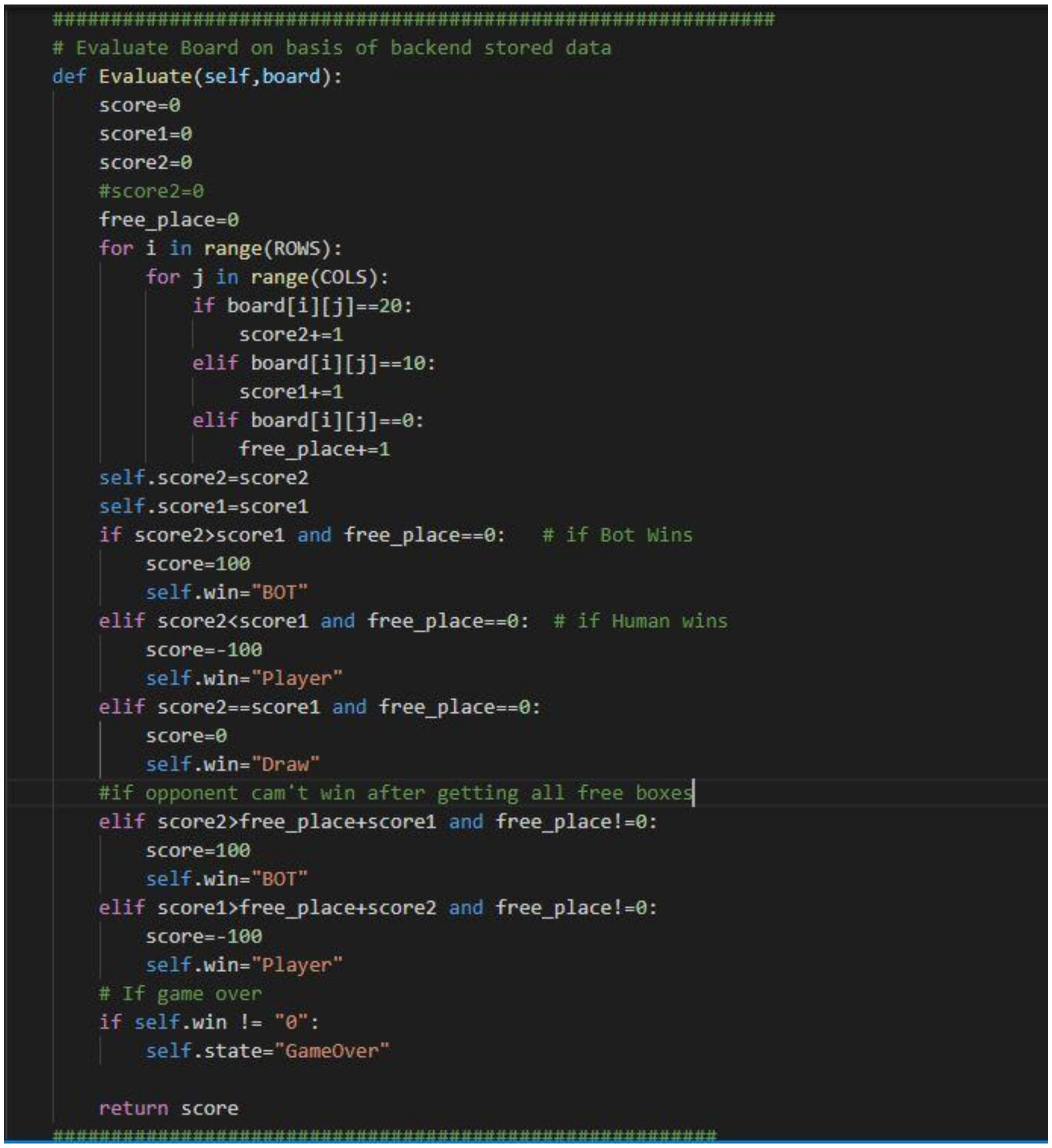
3.7 Evaluate Board ()

This function checks the current state of game and return result in possible four states.

- Human Wins
- Player Wins
- Game Draw
- Game Continue

  It counts scores of both players and then decide who is winner. One who have higher scores is winner.

  In this game which evaluation function is used to evaluate game can be shown in following code snippet *(figure 6)*.



```
##############################################################
# Evaluate Board on basis of backend stored data
def Evaluate(self,board):
    score=0
    score1=0
    score2=0
    #score2=0
    free_place=0
    for i in range(ROWS):
        for j in range(COLS):
            if board[i][j]==20:
                score2+=1
            elif board[i][j]==10:
                score1+=1
            elif board[i][j]==0:
                free_place+=1
    self.score2=score2
    self.score1=score1
    if score2>score1 and free_place==0:   # if Bot Wins
        score=100
        self.win="BOT"
    elif score2<score1 and free_place==0:  # if Human wins
        score=-100
        self.win="Player"
    elif score2==score1 and free_place==0:
        score=0
        self.win="Draw"
    #if opponent cam't win after getting all free boxes
    elif score2>free_place+score1 and free_place!=0:
        score=100
        self.win="BOT"
    elif score1>free_place+score2 and free_place!=0:
        score=-100
        self.win="Player"
    # If game over
    if self.win != "0":
        self.state="GameOver"

    return score
#########################################################
```

Figure 6: Evaluation Function

3.8 On Mouse Press ()

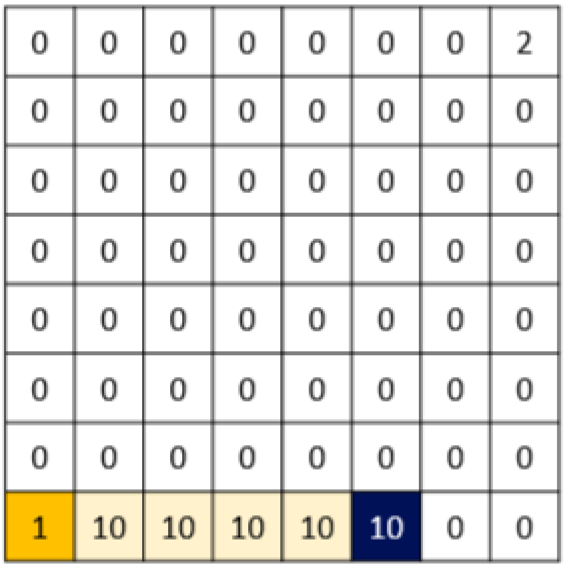
It is built in function of Arcade. It returns coordinates of grid where mouse clicks. Moreover, game developers convert these coordinates into row and column so that in backend we update value on converted row and columns. With the help of coordinates we update backend and then call resync board () to update grid.

3.9 Is legal move ()

This function is used to check is clicking area is legal or not. If user click on grid lines or outside grid it will consider illegal move and consider this move as foul. If move is legal means according to rules then board would update. Which moves are legal and which are illegal we discussed it above in chapter 1 in movement scenarios.

3.10 Player's Position ()

This function return position of player on board. It can be used for both player with player in parameter that of which player we want to get position. Its outcome is row and column where player is placed. A number is given to player search by its number in backend board.



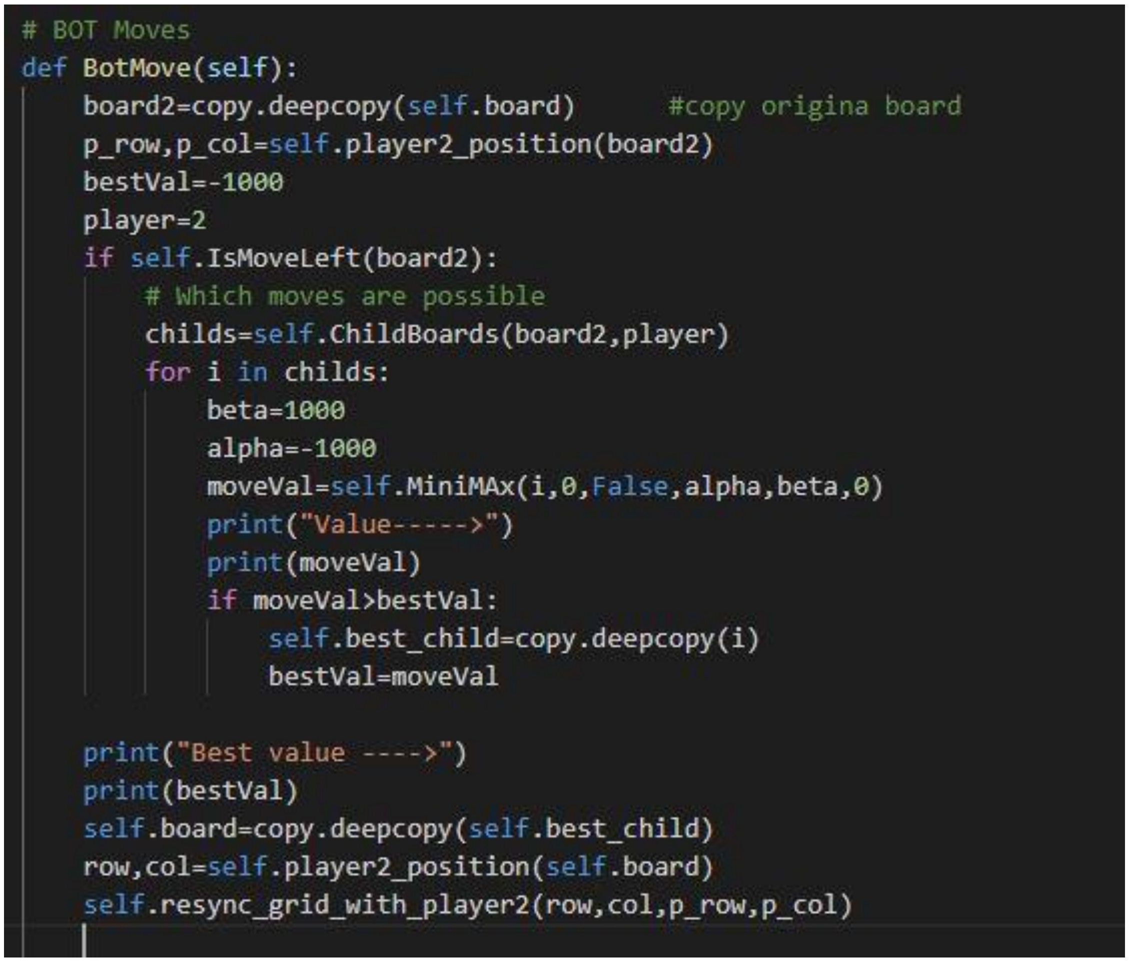
0	0	0	0	0	0	0	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	10	10	10	10	10	0	0

Figure 7: Board after some move

Here in *(figure 7)* yellow box determines position of player 1, 0 determines empty boxes and 10 represent player 1 splashes.

3.11 Bot Move ()

It is core of that game that how bot moves. This function get current board position and try all possible moves and through most know algorithm for AI games "Minimax" it will go into deep and then perform his best move that would leads bot to winning state. It is main function of game that bot move will be best move for bot and worst move for player. After Bot move again evaluation is required to check is there may be game over. In this game developers have used some hardcoded moves so to make bot more competent. You can easily understand how bot move by seeing *(figure 8)* the code snippet of this function.



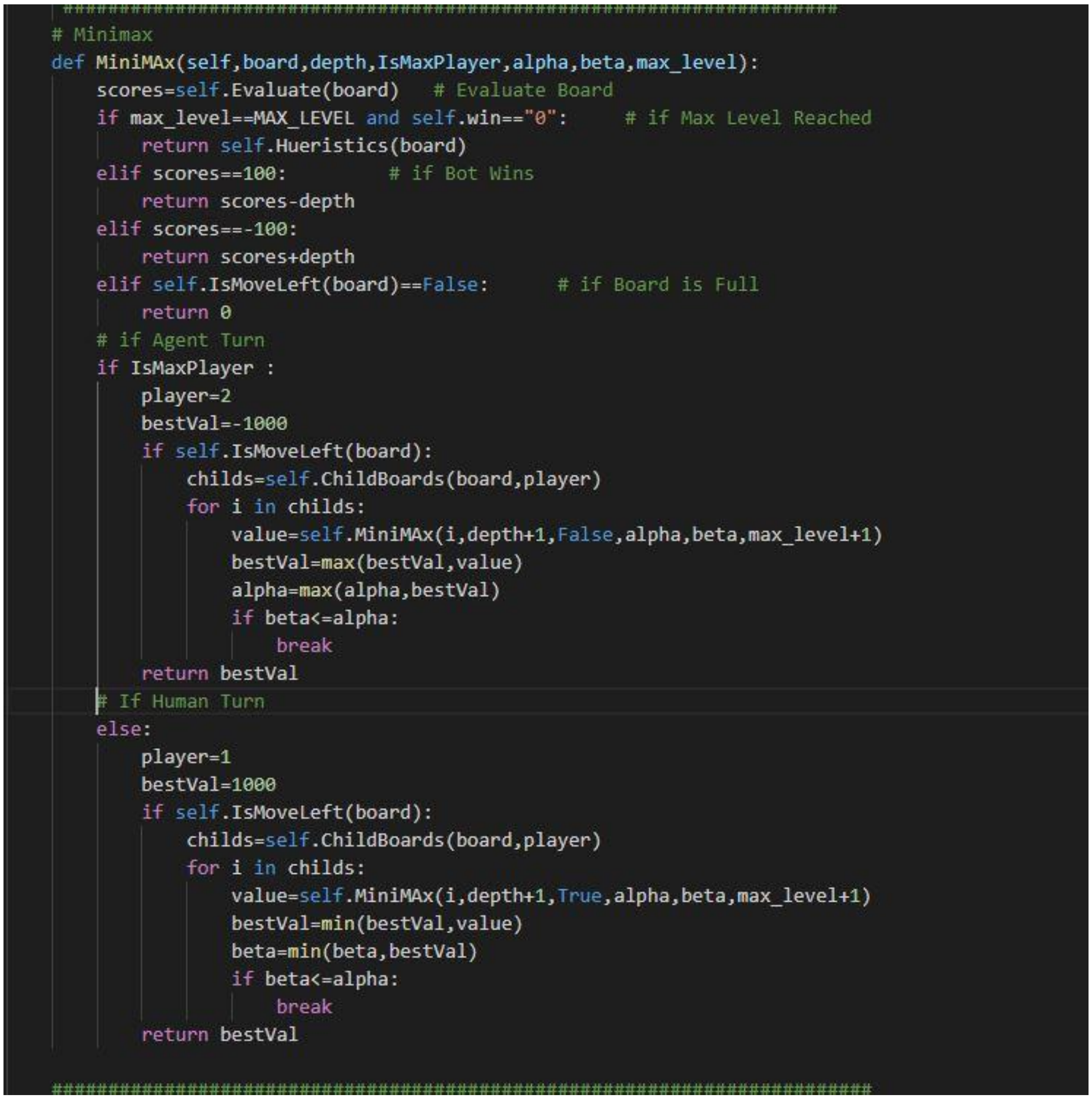
```python
# BOT Moves
def BotMove(self):
    board2=copy.deepcopy(self.board)      #copy origina board
    p_row,p_col=self.player2_position(board2)
    bestVal=-1000
    player=2
    if self.IsMoveLeft(board2):
        # Which moves are possible
        childs=self.ChildBoards(board2,player)
        for i in childs:
            beta=1000
            alpha=-1000
            moveVal=self.MiniMAx(i,0,False,alpha,beta,0)
            print("Value----->")
            print(moveVal)
            if moveVal>bestVal:
                self.best_child=copy.deepcopy(i)
                bestVal=moveVal

    print("Best value ---->")
    print(bestVal)
    self.board=copy.deepcopy(self.best_child)
    row,col=self.player2_position(self.board)
    self.resync_grid_with_player2(row,col,p_row,p_col)
```

Figure 8: Bot Move Function

3.12 Minimax ()

It is used to implement Minimax algorithm. In this function board is expanded until terminal state and then decision is taken that which move should take. But In this 10 by 10 board states much and it would be time consuming to compute each state. So in this function we pass another parameter named “level” we change our base case that without reaching terminal take decision when max level reached on base of these results take best move. In this scenario another other function will call named “heuristic” we discussed it later. *(figure 9)* is implementation function of Minimax in Python.



```
#########################################################################
# Minimax
def MiniMAx(self,board,depth,IsMaxPlayer,alpha,beta,max_level):
    scores=self.Evaluate(board)   # Evaluate Board
    if max_level==MAX_LEVEL and self.win=="0":     # if Max Level Reached
        return self.Hueristics(board)
    elif scores==100:         # if Bot Wins
        return scores-depth
    elif scores==-100:
        return scores+depth
    elif self.IsMoveLeft(board)==False:      # if Board is Full
        return 0
    # if Agent Turn
    if IsMaxPlayer :
        player=2
        bestVal=-1000
        if self.IsMoveLeft(board):
            childs=self.ChildBoards(board,player)
            for i in childs:
                value=self.MiniMAx(i,depth+1,False,alpha,beta,max_level+1)
                bestVal=max(bestVal,value)
                alpha=max(alpha,bestVal)
                if beta<=alpha:
                    break
        return bestVal
    # If Human Turn
    else:
        player=1
        bestVal=1000
        if self.IsMoveLeft(board):
            childs=self.ChildBoards(board,player)
            for i in childs:
                value=self.MiniMAx(i,depth+1,True,alpha,beta,max_level+1)
                bestVal=min(bestVal,value)
                beta=min(beta,bestVal)
                if beta<=alpha:
                    break
        return bestVal

#########################################################################
```

Figure 9: Minimax Function

3.13 Child Boards ()

This function takes board and player as input and return all possible moves that player can traverse. It returns list of child board which are used in Bot Move function. Childs are created keeping in mind of legal moves that which are possible and which are not.

3.14 Is Move Left ()

This function is used to check whether move is left or not means is there empty box in grid. It returns Boolean value True if empty box found otherwise return False.

3.15 Heuristic ()

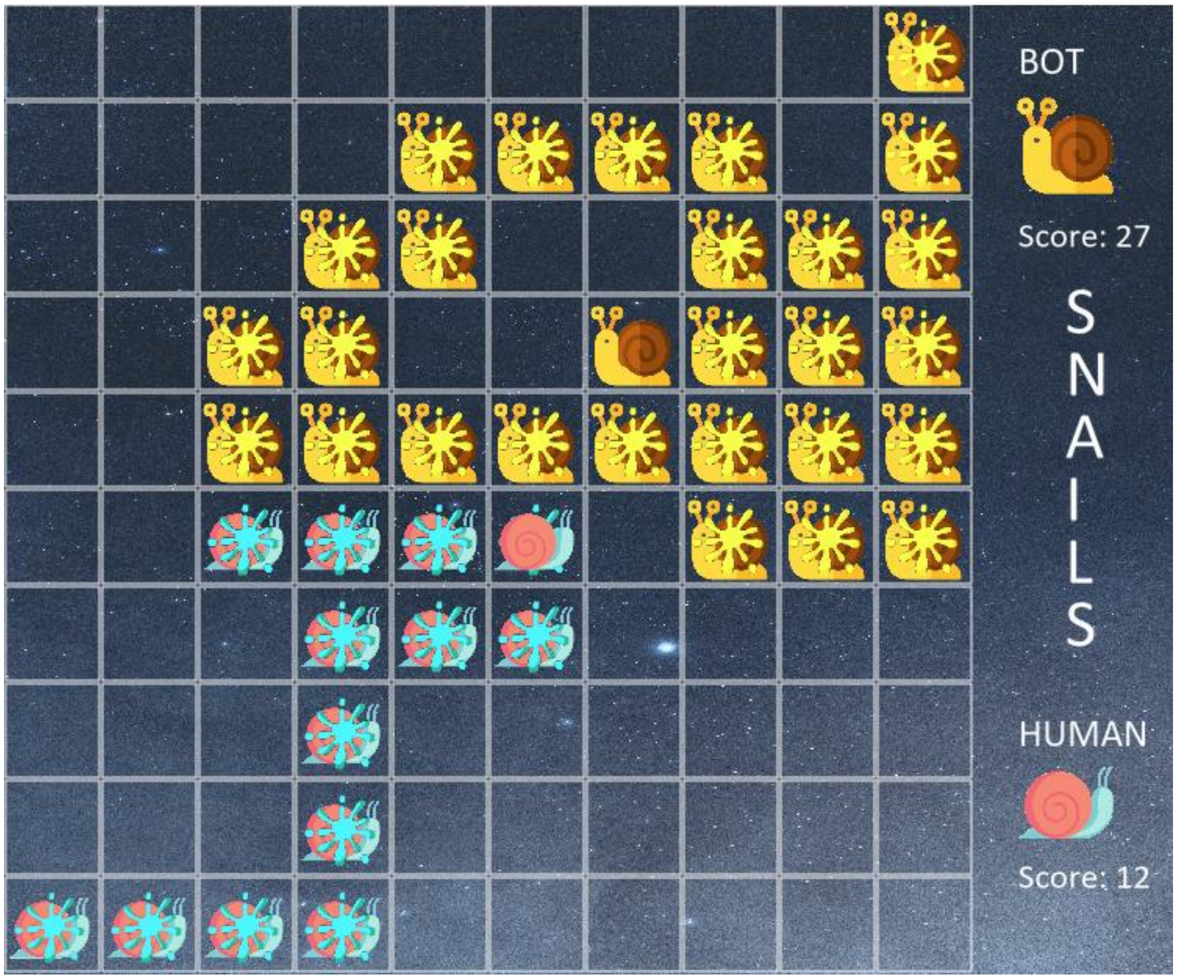
The second core function of game. This function basically performs some techniques and on basis of these techniques return value that is useful for bot and worst for player. Techniques that developers used in this game are following.

- Try to reach center as fast as possible means if bot is at center when max level is reached winning chances of bot is high because now it can traverse many boxes.
- First some moves are diagonal to reach center quickly.
- Check empty boxes around bot where there are many empty boxes around bot more probability of winning bot.
- How many boxes it traversed if this move take more boxes means more winning chances.
- Restrict opponent if bot somehow restrict opponent its winning chances are very high than this can be best move.
- In central region probability winning is very high
- Minimum distance between player and bot means bot restricting player and try to bound it than winning chances of Bot is higher.
- We have also tried to work on miniboard instead of whole 10 by 10. Developers have created 4 by 4 board around Bot position and then by passing it to minimax to find exact best move but it didn't work correctly that's why we quit that techniques but if you want that something try that technique for your tasks.

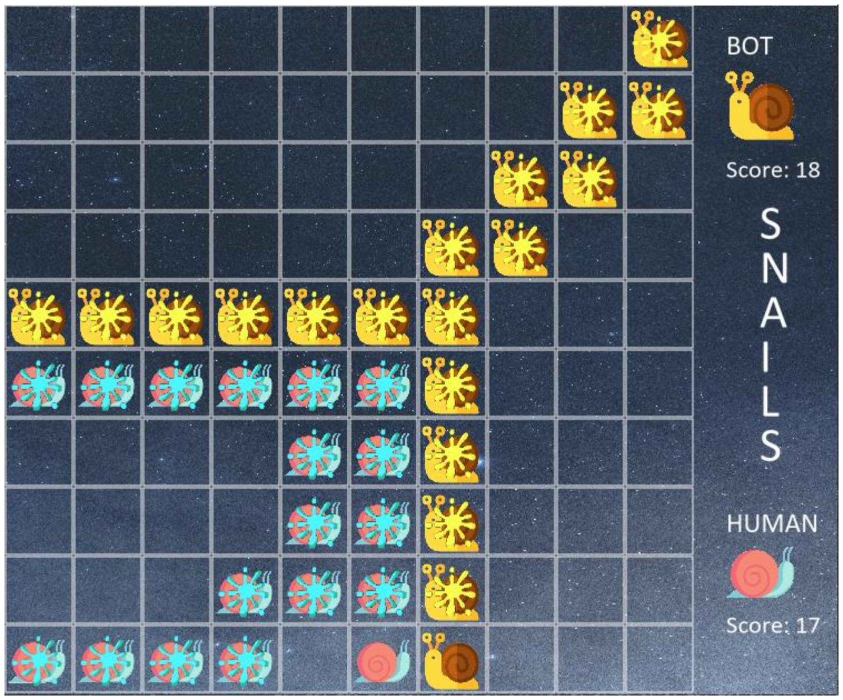
In following game developers implement above mentioned techniques and question raised here is where heuristic would call?. Basically, we can say heuristic as guess without knowing exact solution. In this we specify a max level when this level reached then without going deeper call heuristic at that level. Heuristic function returns winning chances of bot according to give techniques. If maximum value returned by heuristic it means by performing that move more probability of winning.

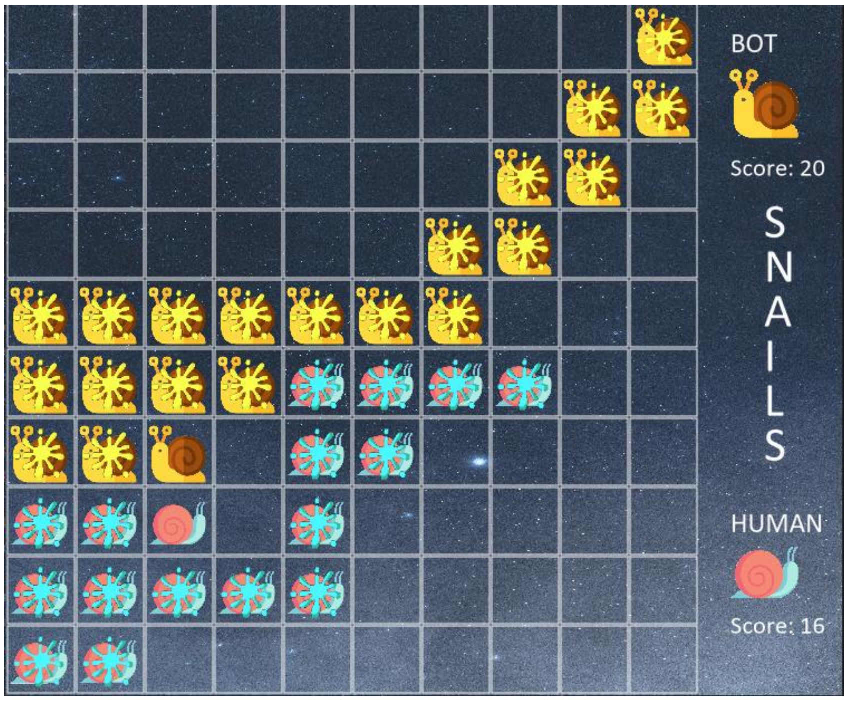
4. Results and Observations

Here are some states of game that how bot restrict player and get more profit by efficient move.



- Bot restrict opponent in small region and this is winning strategy.





References:

artint. (n.d.). *artint.info*. Retrieved from artint.info: https://artint.info/html/ArtInt_56.html

Geeks for Geeks. (n.d.). *Game Theory*. Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/?ref=rp

Geeks for Geeks. (n.d.). *Minimax Therory*. Retrieved 11 30, 2020, from GeeksforGeeks: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

javatpoint. (n.d.). *javatpoint*. Retrieved 11 30, 2020, from https://static.javatpoint.com/tutorial/ai/images/mini-max-algorithm-in-ai-step1.png

javatpoint. (n.d.). *Javatpoint*. Retrieved from Javatpoint: https://static.javatpoint.com/tutorial/ai/images/alpha-beta-pruning-step8.png

springer. (n.d.). *springer*. Retrieved from https://link.springer.com