Snails Game

Zia-ur-Rehman & Ali Raza Khan

Namal Institute Mianwali

--

[----------]

Abstract

Game Development is a technique to create and describe games and their design. It evolves through a hierarchy of different processes, I.e., designing a game, building a game and testing it. Mostly a crew of more than one person is involved in developing a game. Sometimes a game is played by 2 humans but sometimes humans have to compete a machine, which we call computer bot/agent. Artificial Intelligence is used as the driving force for that bot/agent. In artificial intelligence, machines make intelligent moves and compete a human or sometimes another AI bot/agent. Our game “SNAILS”, is a 2D board game in which a human an AI bot/agent compete. One can win by occupying maximum number of boxes in minimum moves.

Game Overview

Game Development is a technique to create and describe games and their design. It evolves through a hierarchy of different processes, I.e., designing a game, building a game and testing it. Mostly a crew of more than one person is involved in developing a game. Sometimes a game is played by 2 humans but sometimes humans have to compete a machine, which we call computer bot/agent. Artificial Intelligence is used as the driving force for that bot/agent. In artificial intelligence, machines make intelligent moves and compete a human or sometimes another AI bot/agent. Our game "SNAILS", is a 2D board game in which a human an AI bot/agent compete. One can win by occupying maximum number of boxes in minimum moves.

Snails is a 2D board game played between two different players, one of them is human, in our case, and the other one is computer bot, whom we call AI bot or AI agent. Player-1 makes his turn and than AI bot makes an intelligent move with the help of some computation which we call artificial intelligence. A player wins of if he/she has covered more boxes than the other one.

Game Rules:

- • The main objective of the game is to occupy more GridSquares than your opponent.
- • It is a turn-based game i.e. a player takes his turn to move sideways.
- • In each turn, the player can move his Snail horizontally or vertically to an adjacent empty GridSquare. In doing so, his score will be increases by 1.
- • When a player moves his Snail to an empty GridSquare, the Snails leaves behind his trail (Slime) indicating the occupied area.
- • Besides moving into empty squares, a player can move onto his own trail of slime as well. But sometimes, it can be unfavorable as the slime is slippery, and

any move back onto the trail will slide the player to the farthest position on that trail in the direction the move was taken. Note that in this case, no score will be awarded.

- • A player cannot move onto his opponent's Trail of Slime that could be advantageous in most cases.
- • In case, a player plays an illegal move i.e. to move his Snail to an out-of-reach GridSquare or to move onto the opponent's trail of slime, his turn will be lost.
- • Once all the GridSquares are captured, the player with the highest score (most GridSquares captured) is announced as Winner.

Requirements:

- Installation of VS Code and Python 3.0+
- Installation of arcade
- Installation of pyinstaller to make executable file out of python file(s). For pyinstaller you need to work on windows OS as arcade does not support pyinstaller on Linux OS.

Some game terminologies:

- Player-1: Player is human player
- AI Agent: AI Agent is the opponent player which makes intelligent moves
- Snail: Snail represents current position of each player.
- Splash: Splash represents those cells each player have traversed.
- Grid World: Grid world is graphical representation in the form of boxes
- Grid Cell: Grid cell is single box which can contain splash or snail of any player or can be empty.

- Board: Board is a 2D array used in the backend and is synced with grid world.

Constraints:

- Size of backend 2D board, which is a 2D list in python, should be 10x10.
- Size of snail and splash should be less than size of a grid cell.
- Size of window should be 800x800 pixels.
- Window screen is not resizable.

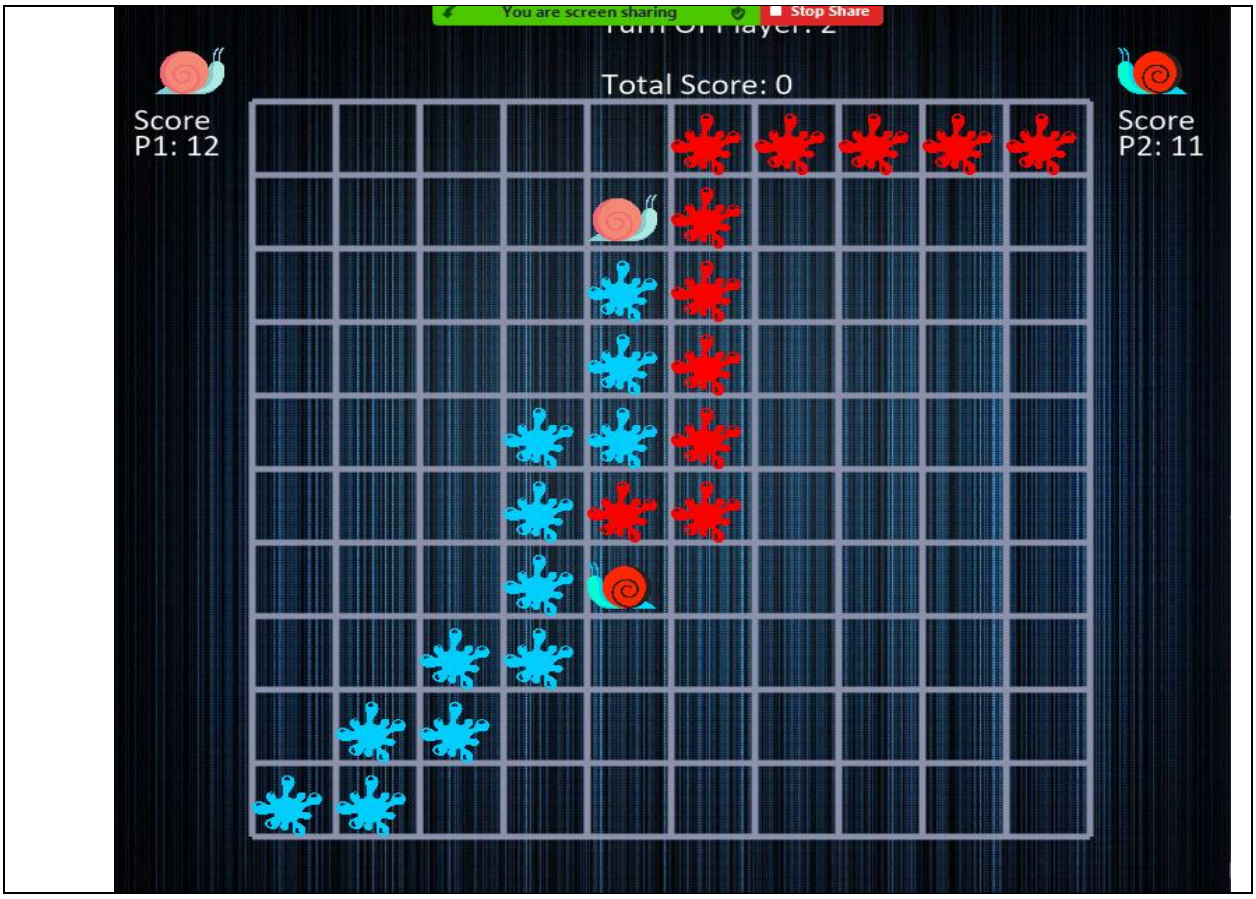
Game controls:

- Reset/Restart: Human player can reset or restart the game
- Pause: Human player can pause the game.
- Snail Movement: Human player can control the movement of his snail using up/down/left/right arrow keys.
- Exit: Human player can exit the game at any moment.

User Interface:

When you start the game, you will see the following screen, main game screen.

Below is a snapshot after few turns by both AI bot and human player:



Illegal Moves:

- A player cannot move diagonally.
- A player cannot make a move by missing one cell
- A player cannot make two moves at once
- A player cannot move onto opponent's trail/splash.
- A player cannot move onto opponent player's snail.

Literature Review

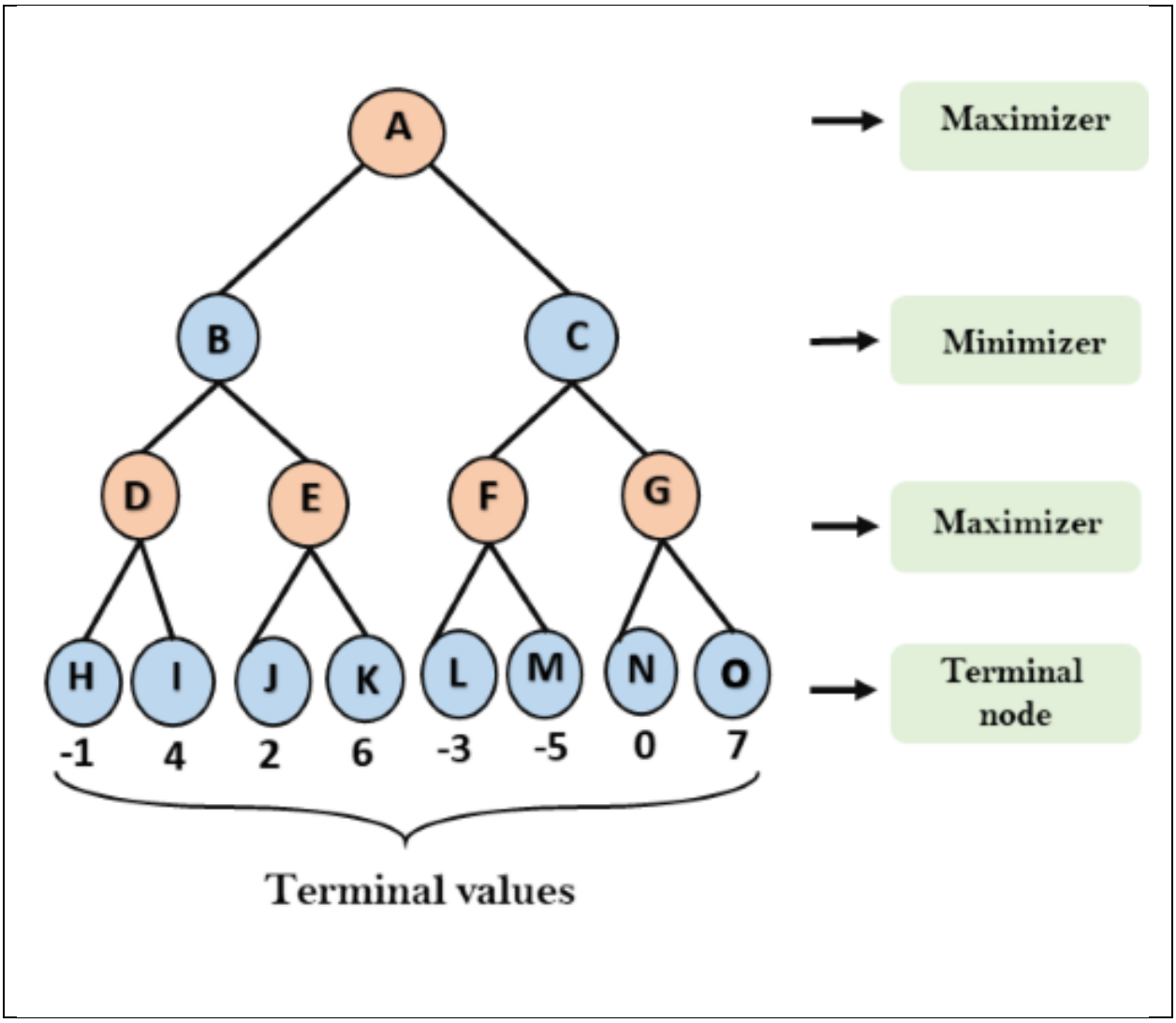
Movement, Finding Path and Decision Making:

Firstly, AI bot looks for empty boxes than it looks for the cells covered by itself and then it looks for edges, this process makes a hierarchy of finding a path. Every time AI bot has to go through this hierarchy to make a good choice.

Secondly, AI bot search for an empty space. When the AI bot finds an empty box than it is turn for AI bot to make a decision based on available empty boxes. And as we said earlier, the AI bot has to choose a box wisely. Here, the AI bot choose best way to reach the center. Because when any of the player the reaches the center, it maximizes its probability of winning the game. Also, with each decision-making AI bot try to minimize the distance between opponent's snail and itself. If the AI bot does not find an empty box around its current position, then it chooses a path where it can find an empty box by moving onto its own trails. After all these conditions, even then it isn't able to find an empty box, then AI bot moves to any edge near its covered cells.

Minimax Algorithm:

Minimax Algorithm is used is a recursive algorithm or we can a backtracking algorithm which is used in games for decision making. Here we choose an optimal move with respect to opponent player. This is because minimax algorithm also looks for all the possibilities with opponent player can make a move. In minimax we make optimal decision with the help of a game-tree. Minimax algorithm performs depth first search. It goes all the way down to nodes of the tree where possibly game ends, and then backtracks the tree filling all the values in the tree, where Bot/agent try to select the maximum value and when it is human player's turn it selects minimum value. Below is a figure of how it really works in game tree.



The above figure explains what minimax do in steps while searching or looking for an optimal move for the AI or non-human player.

Heuristic(s):

Heuristics are used to make shortcuts in searching algorithms based on some available information. In heuristic functions we often make a guess at some instance that whether we are going in the right direction or not. This is helpful when you have to search a word, for example, in a dictionary and instead of searching the dictionary completely you make a choice or guess after, for example, searching 100 words we will decide whether the word we are looking for is in the dictionary or not. In our game, Snails, we had to use minimax along heuristic function to

make optimal choices but in less time. Because without heuristic function we had to draw or make a complete game tree at each turn. But with heuristic function we decide this early; we can say after 10 branches of the tree we call heuristic function to decide whether we are in the right direction or not.

Game Development Methodology:

Breif Introduction:

In order to understand the structure of 2_player game, basic and simple core python is used for backend with a library called Arcade for front end representation. Each step has separated function and each screen has separate class. This made the code easy to understand and can be modified later on.

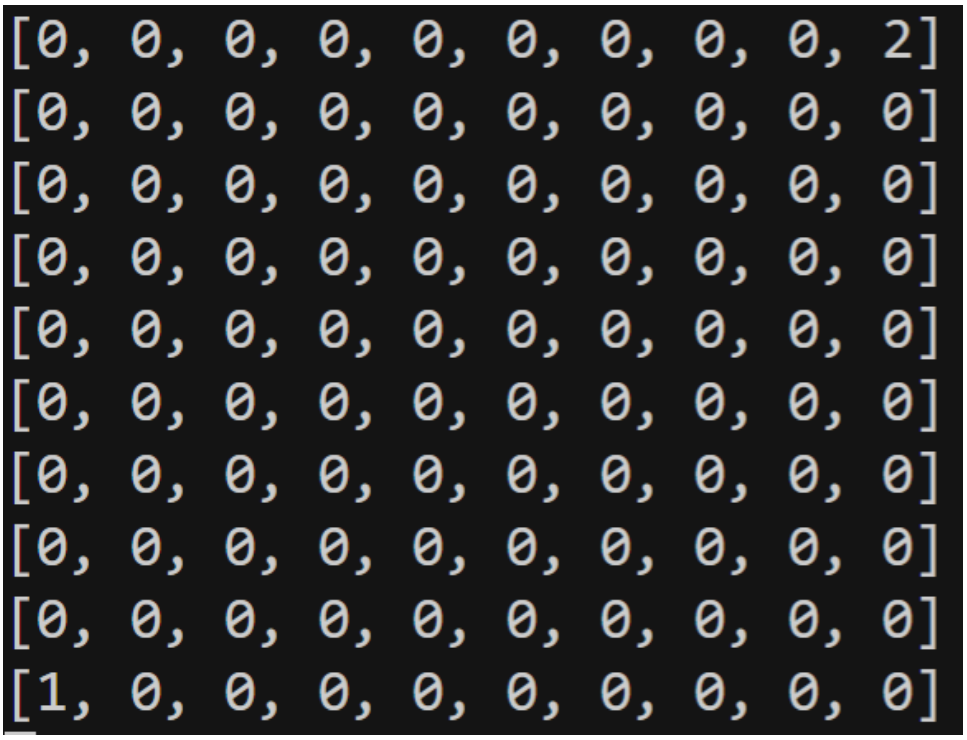
Main Functions:

This section elaborates each function and their functionality. What they and how they do it. Some of the functions are built-in functions of arcade library.

Initialize Board ( ):

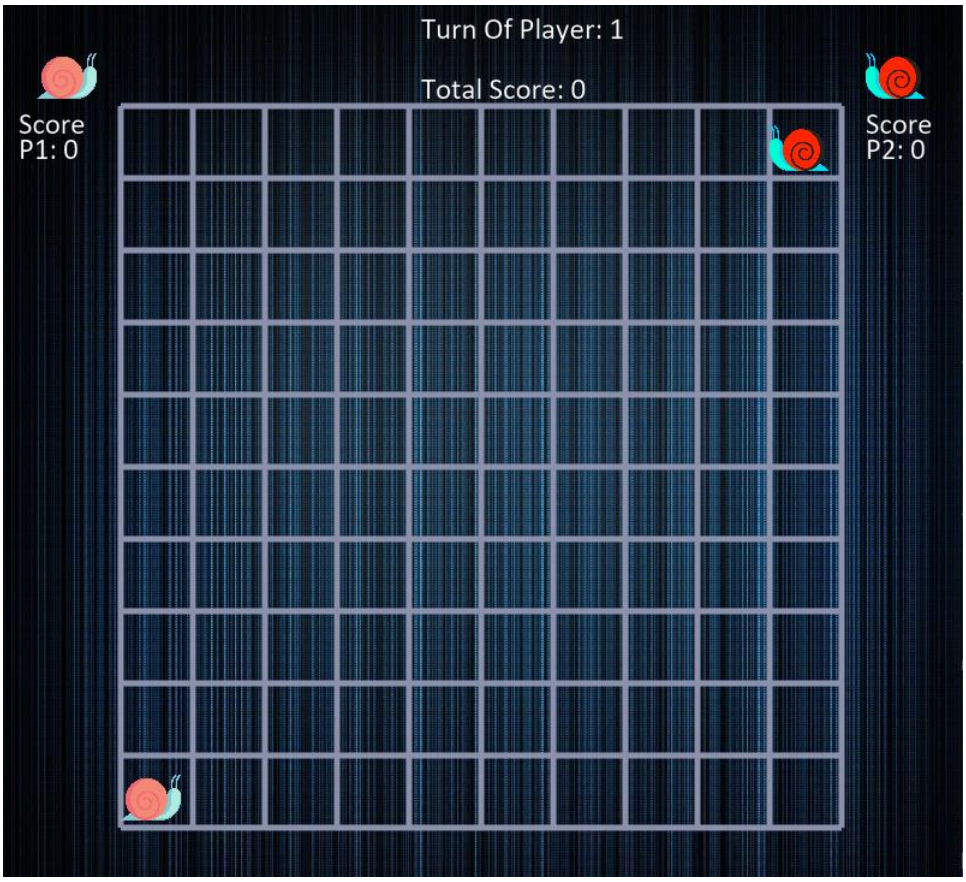
This function creates a 2d board at the backend which is synchronize with grid world at the front end. It is actually a 2d list of python. Other than that, initialize the starting position of Human player and AI bot. All other position is initialized to zero which represent empty boxes.

- Human Player: Represented by 1
- Human Snail: Represented by 11
- AI Bot: Represented by 2
- AI Snail: Represented by 22
- Empty Boxes: Represented by 0

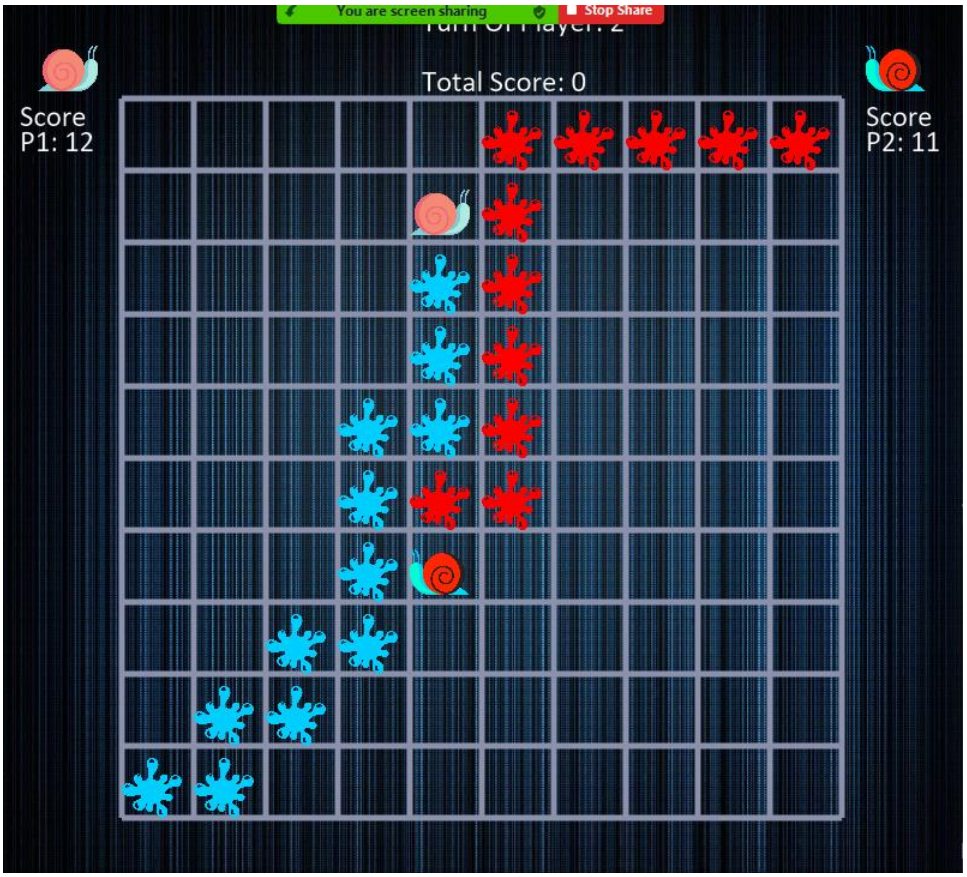


On Draw( ):

This function draws everything on the front end. This is a function of View class of arcade library. This is a built-in function of arcade library and called automatically **arcade.start_render( )** and **arcade.finish_render( )** are the calls of this function everything between them is drawn on screen.



This is a front-end Grid drawn by it. It is synchronized with the backend 2d board.It is drawn after each step



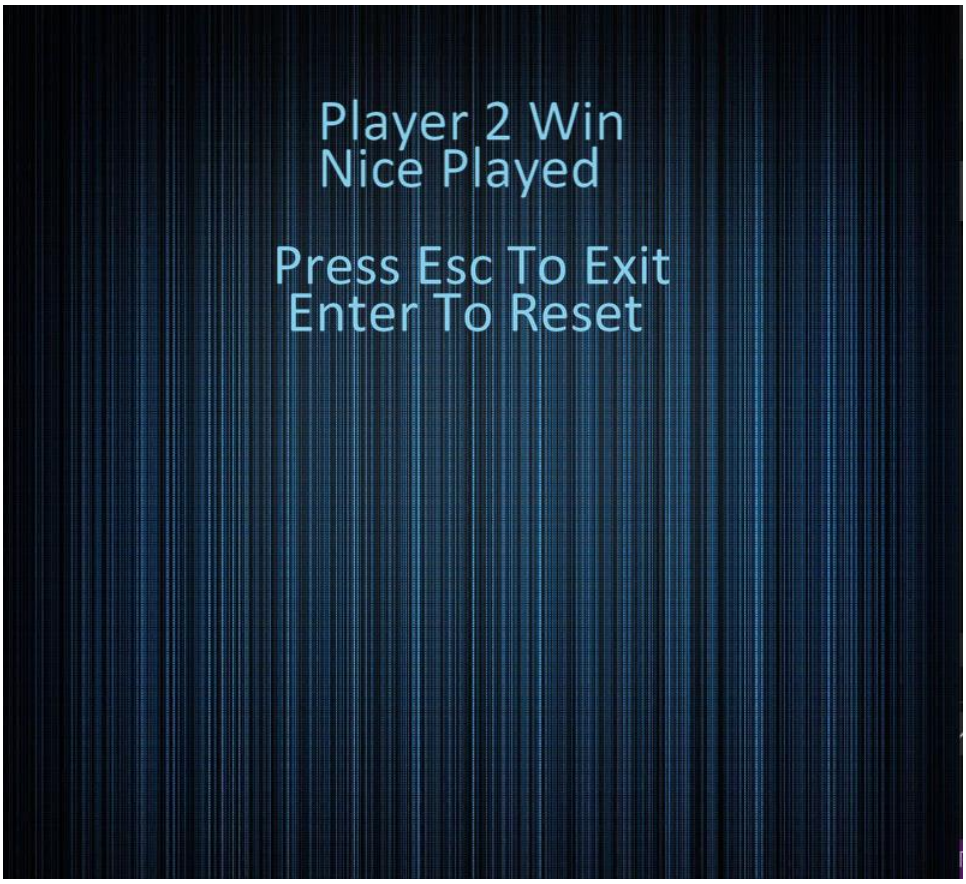
On Show ():

This is also a built-in function of arcade which allows the mouse cursor to be shown or not on the game screen. There are also many other attributes which have we have not used.

On Key Press ():

This function is called as soon as any key is pressed on keyboard. This is also a built-in of arcade. There are some states design according to which different keys are allowed at each state. Like on **Game ON** state only movement keys and game Esc key work.

On **Game Over** Only Esc and Enter key work



Get Human/Bot Position ():

This function traverses the 2d board of blackened and find the location of human snail on it and return the x, y coordinates human/bot snail current position. It is called at key press. To check whether the next move will be possible or not.

Slip Up/Down/Right/Left ():

When no empty box is surrounding the snail then. According to the rules of game a snail can move on its own trails only. The snail moves to the last splash of the given direction. The slip function is design to find the last splash and return the coordinates of it.

Score Count ( ):

This function keeps the record of score by traversing the 2d board and counting the values of splash of each player. This function is called after each movement either by Human or by AI.

Eval ( ):

This function checks the evaluation of game. That whether the game is on continue state or draw or win state. This is done by keeping the track of score. If they cross the given threshold of maximum score the are declared win. There is also some part of it hard coded to check whether a player is stuck or not. If a player is stuck then the player with max score is declared win.

Possible Move ( ):

This function is a helper function for heuristic and minimax algorithm. According to the game rules a snail can only move one box in all the 4 directions. In order to check if the move is valid or not for the AI Bot. This function is given the coordinates of AI Bot with the box coordinates and it return (True or False). It helps to optimize minimax and heuristic and avoid unwanted moves.

Heuristic ( ):

This is the function which design all the strategy for AI Bot movement using heuristic and A* approach. Main points

- Check Opponent Distance
- Check Center Distance
- Check Valid Moves
- Check Empty Boxes

- Check Empty boxes after trails

In this function there are some priority measure which help a little bit to find the best move at each position depending upon the surrounding of AI Bot.

First:

First of all, its check for possible moves. If there is any valid move in its surrounding it keep track of it. It compares it with the distance of opponent and center and if any move is decreasing it takes that move is considered the best one.

We also consider to implement the empty boxes of each valid move to find the best move.

Second:

If there are no empty boxes in the valid moves of AI current position than it has to move on its own trails. In order to choose the best trail. All the possible trails are kept in list and later on the best one which decrease the distance against opponent is chosen but still here to move towards those trails which will give more empty boxes in the next coming move is considered to be implemented.

Third:

If there are no empty boxes surrounding it and none of the trail give empty boxes either then it has to choose any edge point having opponent splash or snail. This logic is still not completed to choose the best one which may help in the upcoming moves.

Assumptions:

This section will give an overview of that part which we thought to implement but yet not implemented.

Minimax:

Before implementing heuristic, we tried to implement minmax. But how backtracking was helping us in it we were not able to understand that, how max value is changing. We tried to implement minimax but we don't get the desired output of best moves and was only working till 3 depth and on empty boxes. Therefore, we neglected the minimax and started working on heuristic.

We understand that heuristic only check the possible best move max to one possible location. But embedding that code in minimax we can increase that to 4 or 5 location or till complete board. We have to check the empty boxes around moves in heuristic. Which will increase the efficiency of best moves. In order to implement those things, we need some more time. The backtracking in minimax work on the values return with the coordinates. After implementing heuristic, we come to know that how this value is changing and how will it help to choose the optimal move in minimax.

After that we get the whole idea how heuristic and minimax contribute with each other and how they will optimize and get the best moves.

Heuristic will check these conditions

- Empty spaces around it
- Distance between opponents
- Distance towards the center

It will return some integer value with each move and that number will help to choose in best move.