Snail Game

By Saira (1802032), Qazi Arsalan Shah (1802030), Abdullah (1802016)

November 2020

Abstract

The importance of computer games is growing because of the fact that games that are open today are now open on computers.With the advent of so many new technologies, the world has become very established.Computers can now do everything that humans can do but at very higher speed.Game development has its own world with some specific languages and grammar.For a most recent couple of years, a pattern has been arises to create AI based games.Because in human to computer game both player should be able to think like humans to play more efficient.Computer Game development is a major entry point in computer Science. In this game, we are focusing on grid based games like Chess, Ludo or Go. This game is about Snail space occupation. One important aspect is to make character intelligent enough to draw or beat human. Win situation occurs when human snail occupies less space in grid then bot snail. In Short, This is an intelligent game playable both human to human and human to intelligent bot. This game comes in the category of adversarial search problem.

Contents

0.1 Introduction

Computer Game development is a major entry point in computer Science. In this game, we are focusing on grid based games like Chess, Ludo or Go. This game is about Snail space occupation. One important aspect is to make character intelligent enough to draw or beat human. Win situation occurs when human snail occupies less space in grid then bot snail. In Short, This is an intelligent game playable both human to human and human to intelligent bot. Intelligence in bot is added by using different algorithms for adversarial search and optimal path finding algorithms. Heuristic search is also used in making intelligent decisions.

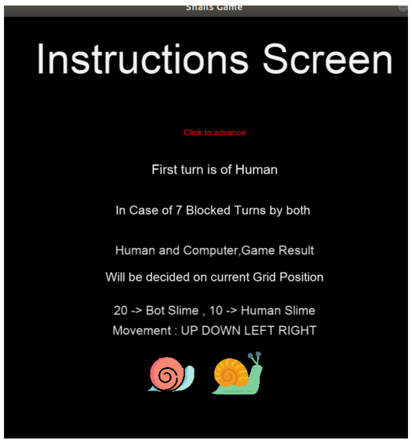
0.2 2 Player Snail

0.2.1 Frame Work

Arcade is an easy-to-learn Python library for creating 2D video games. It is ideal for people learning to program, or developers that want to code a 2D game without learning a complex framework.Arcade games kept on improving with innovation and ongoing interaction advancements.Arcade games frequently have short levels, straightforward and control plans, and quickly increasing difficulty.

0.2.2 Instruction Screen

As soon as game is turned on, Instruction screen appears as below



Snails Game

Instructions Screen

Click to advance

First turn is of Human

In Case of 7 Blocked Turns by both

Human and Computer,Game Result

Will be decided on current Grid Position

20 -> Bot Slime , 10 -> Human Slime

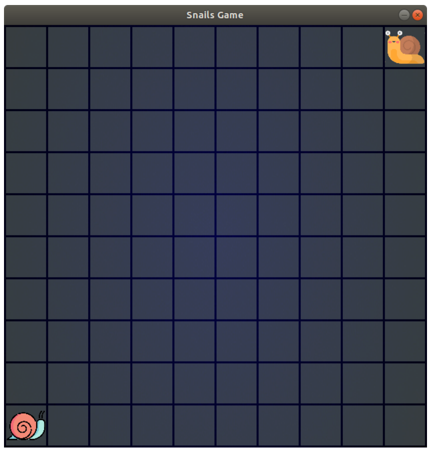
Movement : UP DOWN LEFT RIGHT

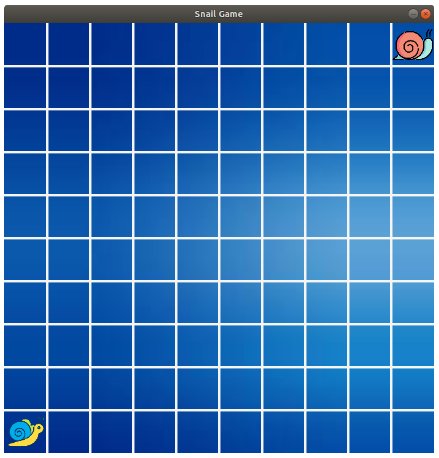
0.2.3 Grid Initialization

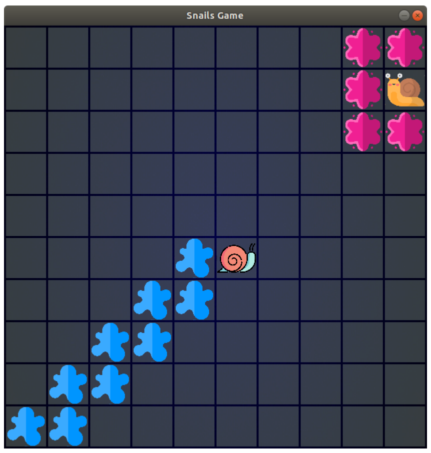
Grid was initialized of 10 rows and 10 columns, total of 100 blocks after that snails were placed at there starting points.One snail placed at (0,9) position and second snail placed at (9,0) position.when game played by human or a computer According to valid move snails change there positions as well as at the back end. We stick to 10X10 grid because of window screen size constraint.

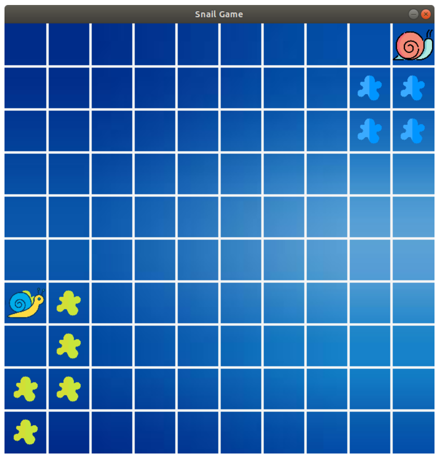
0.2.4 Snails Movement

Snails can move adjacent to there previous place and throw a mark in the form of splash at there visited place like below.we can check the movement of snails are valid or not.As we know in this game we are using adjacent blocks.If the snail's move is consecutive then move is valid, otherwise that move is invalid. Unlike tic tac toe human or bot can choose any box but in snails game there is a constraint on movement. Snail can only move up, down, left and right yet these movements are subjected to validity as well.









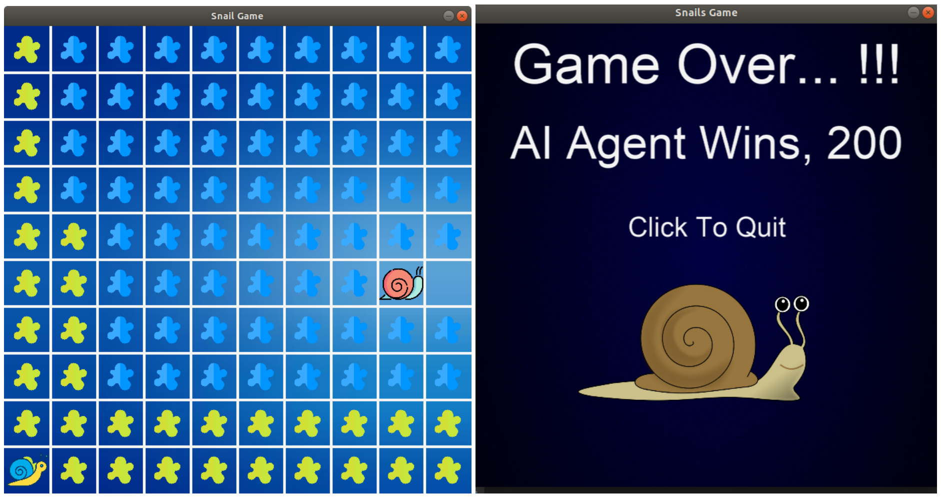
0.2.5 Game Rules

- The main objective of the game is to occupy more grid squares 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 grid square.In doing so, his score will be increases by 1.
- When a player moves his Snail to an empty grid square, 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.
- 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 grid square or to move onto the opponent's trail of slime, his turn will be lost.
- Once all the Grid Squares are captured, the player with the highest score (most Grid Squares captured) is announced as Winner.

0.2.6 End Results

There are always three Situations in the end win, lose or draw. That snail which occupies most boxes will win rest will lose or equal will draw. An example is below



Game Over... !!!

AI Agent Wins, 200

0.3 Literature Review:

Literature review of any article of research paper provides us with good amount of information regarding our concerned problem and gives some sort of road map for further proceedings. In making of our AI agent intelligent we felt a need to refer to different searching techniques that are efficient and will make our agent intelligent. Rather then sticking to hard coding engineering techniques. There is a good development in AI based games in which AI agent can compete human mind by thinking rationally as human. So there are few techniques that help AI agents to compete human mind. These techniques will be explained in code explanation section.

0.4 AI Based Agent

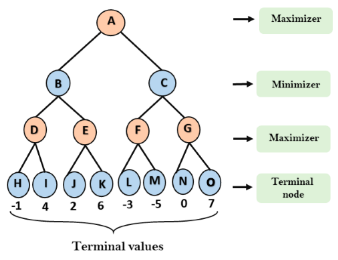
To make one snail Intelligent, we used a different programmatic approach for its implementation. As it is an adversarial search problem in which a character has to compete with an opponent and both try to get victory, there is a very simple yet good algorithm for this type of search problem which is Mini-Max algorithm. This algorithm in simple words is used for optimal decision making by exhausting all the possibilities and then choosing the best optimal path to go with. But for large complex games like chess and our snails game, Mini-max alone cannot optimally and efficiently helps us in decision making as it computationally not possible to look all the possibilities and then try to select best optimal path. So there are different techniques that helps Mini-Max do its works efficiently and optimally while meeting the possible computational power and possibility. These techniques include Alpha-Beta pruning, limited Mini-Max etc.

0.4.1 MINI-MAX Algorithm

The Minimax algorithm relies on brute force search and a simple evaluation function. Let's assume that every time during deciding the next move we search through a whole tree, all the way down to leaves. Effectively we would look into all the possible outcomes and every time we would be able to determine the best possible move.

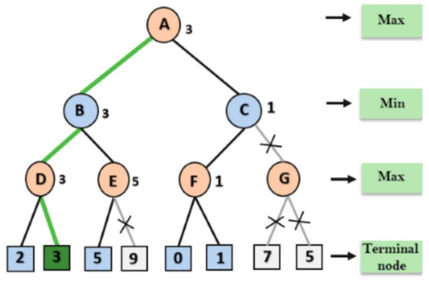
- Working of the minimax algorithm can be easily described using an 2 player game.
- There are 2 player, one is maximizer and other one is minimizer.
- Maximizer tries to get the maximum possible score(+1) while Minimizer tries to get the least(-1).
- Its working is exactly same as DFS, goes till leaf nodes and return back (Backtracking algorithm).
- At the terminal nodes, we have utilities for each leaf node and based on that we can decide best move to choose.

Best thing about Mini-Max is that it considers that opponent will also play optimally so it stimulates in way by considering that opponent will play very smart and optimally, so it selects best move for AI based on that strict assumption. In practice, however, we may encounter an opponent that makes silly moves. When this happens, the algorithm's assumption deviates from the actual opponent's behavior. In this case, it still leads to the desired outcome of never losing.



0.4.2 Alpha-Beta Pruning:

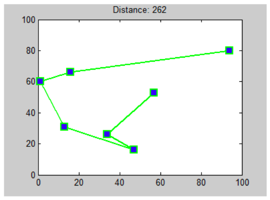
A way to optimize Minimax, Alpha-Beta Pruning skips some of the recursive computations that are unfavorable i.e. throwing off complete sub graphs as well. After establishing the value of one action, if there is initial evidence that the following action can bring the opponent to get to a better score than the already established action, there is no need to further investigate this action. This technique uses two parameters alpha and beta. Alpha is the best choice found so far at any point along the path of Maximizer player. The initial value of alpha is -infinity. Beta is the best choice found so far at any point along the path of Minimizer. The initial value of beta is +infinity. So by pruning the unfavourable nodes our Mini-Max becomes very fast.



0.4.3 Heuristic Algorithm

Heuristic Algorithm belongs from the class of Decision problems. Its is used to solve the problem quickly and efficiently. Heuristic algorithms are most often employed when approximate solutions are sufficient and exact solutions are necessarily computationally expensive.

A very well known problem of heuristic is Travelling salesmen problem in which we are given with cities and distance between them. we finds the route that visit each city exactly once also known as NN Algorithm. We have used heuristic function for estimating our AI agent current situation in game. it will be explained in later sections. Heuristic function is used with Depth limited Mini-Max in which after reaching at a certain level we can predict best move. While using limited Depth MiniMax, correctness of algorithm is completely dependant on heuristic function.

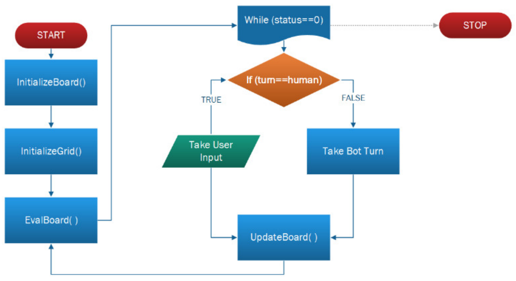


0.5 Code Implementation and Explanation:

There are different views in our game. It includes Menu Screen , Instruction screen , Game View and Game over screen. Game view contains main logic of game. Overall game follows a game loop described in project requirements. Below is the generic explanation of game loop. After game loop section all the helping functions and game logic is explained.

0.5.1 Game Loop

It starts with the initializeBoard() function that initialize the back end board. Then initializeGrid() it makes front end board ready with respect to backend grid. Then evaluateBoard() evaluates the board and returns the game utility. Then bot and human alternate there turns and after each turn board updated and then evaluated using evaluateBoard() and then game state is decided. Here is the pictorial explanation.



0.5.2 Important Helping Functions And Explanation

Below is the brief explanation of important helping function used in game logic.

EvalauteBoard():

As name suggests it simply takes the board evaluates it and returns its state. if there are zeros left the board so it mean game is continued. In case of Human victory it returns 100 and in case of Bot victory it returns 200. Draw state is represented by 50. For more understanding of this function, please refer code file.

IsLegalMove():

As there is a constraint on snails movement and all the moves are not legal, so this function returns the validity of the move i.e. either this move is valid or not. For legal moves it returns validity as true along with legal move coordinates. This uses a sub function checkPossibleActions() which is explained after it.

CheckPossibleActions():

This is a generic function used for both Human and AI Agent and it becomes the sub function for both Human and Bot legal move logic. It takes the current position on the board and returns all the possible moves like up down left right. Then isLegalMove() logic checks whether it is legal or not i.e. there must not be opponent slime or opponent.

SliprySurface():

This function handles the slippery surface of slime. If player clicks on its slime. This function is called in isLegalMove() and slips the snail to its correct position.

0.5.3 Functions Specific To AI Agent:

BotPlaying():

This function is main function for the AI agent which plays the game on AI Agent behalf. if turn is of bot, then this function checks the current position of bot and based on it calls other subsequent functions and moves the bot on different legal positions. It calls FindBestMove() function, that finds best moves according to some intelligence.

AllowedActions():

It takes current postion, board and turn as an input and returns the legal moves. This function uses checkPossibleActions(). This function is used during finding best moves on each bot turn.

FindBestMove():

Here is the start of intelligence in our game. This function is called every time when bot is playing and returns the best move from list of allowed moves based on output of Mini-max and heuristic function. Now there can be a case when there are no empty boxes around the current position of bot, so for this we wrote an extra function that checks after moving on this slime what will be the game position of bot and it returns a heuristic value based on bot position on board. So in this case we haven't called Mini-max rather we used a helping heuristic that tells how good is that slip. For slippery moves two sub function are used, aiSlipChecking() and checkSituation().

AiSlipChecking():

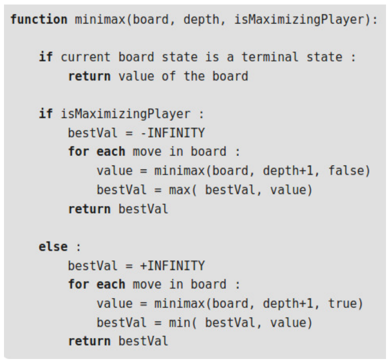
This takes a board , current position , new position and turn as an input and returns the new position after slipping.

CheckSituation():

After Slipping, we are moved to new position, so this function gets the postion and checks current situation of the game around the position and returns a heuristic number for that move. Like number of empty boxes around it by making a 4X4 sub board.

Mini-Max In Our Algorithm:

As in Mini-Max one player maximizes the score and other minimize it. Our AI agent is assigned as a maximizing player. Minimax() takes an input the copy of game board, depth of tree, max depth of the tree and a boolean isMax which is True if it is maximizing player turn. We have implemented depth limited MiniMax which at a specific depth of tree decides which move is better by piggy banking on heuristic function. Below is the generic psuedo code of minimax algorithm to get the rough idea. For better understanding of our code please refer code file.



```
function minimax(board, depth, isMaximizingPlayer):

    if current board state is a terminal state :
        return value of the board

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each move in board :
            value = minimax(board, depth+1, false)
            bestVal = max( bestVal, value)
        return bestVal

    else :
        bestVal = +INFINITY
        for each move in board :
            value = minimax(board, depth+1, true)
            bestVal = min( bestVal, value)
        return bestVal
```

Heuristic Techniques used:

There are many different techniques of heuristics but best suited ones are used in our game. In minimax, at the beginning board is evaluated, if terminal state is reached it returns the control. But as game size is big and it is computationally not possible to exhaust all the possibilities for each single move, so max depth is decided if til max depth, we haven't reached the terminal state, we will stop traversing the tree and will rely on heuristic function. For AI agent to win, it wants to get as many boxes as possible and to make other player stick to low number of boxes. First thing heuristic function does is it calculates visited boxes by the AI Agent and add these to wining chances of agent. Second thing is also checks if we are in center of the board it adds more value to the wining chances of the board. Third thing is to block the opponent player i.e. follow it. So it calculates euclidean distance between the bot and player and if distance is less, this is the favourable move so it adds more value to wining chances of the bot. At last it creates as mini 4x4 or 5x5 board around the current bot position and calculates the heuristic in that board i.e. are there possible empty moves and add them to its wining utility. And then this utility is also added to to wining chances of the bot and then our main heuristic function returns all that wining chances. Now based on that heuristic best move is selected by minimax.

Alpha Beta Pruning:

Our plan was to add alpha beta pruning and then test our agent it will surely add more efficiency and accuracy to our agent but were not able to do that pruning but will add pruning to in it later. It smartly prune the unfavourable sub graphs.

0.6 Recommendations and Discussion:

- For complex games like chess and our snails game where minimax can't be used alone to suggest optimal move efficiently, different techniques must be used like alpha beta pruning , heuristic estimation as described in our paper and there are other techniques as well which helps Minimax to suggest optimal moves efficiently by taking care of computational possibility and efficiency.
- For better understanding of algorithm, refer to code file.

0.7 References:

- cs50.harvard.edu/ai/2020/notes
- https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

- https://medium.com/swlh/optimizing-decision-making-with-the-minimax-ai-algorithm-69cce500c6d6

- https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/ Allis, V. (1994).Searching for Solutions in Games and Artificial Intelligence. Maastricht University,Netherlands.