VIT UNIVERSITY (Estd. u/s 3 of UGC Act 1956) www.vit.ac.in Vellore - Chennai
CHENNAI CAMPUS
Vandalur Kelambakkam Road, Chennai 600127

Foundation OF AI

J Component Project Report for the Course
CSE1014
Winter Semester 2020-2021

Submitted To:

Dr. A. Vijaylakshmi
Assistant Professor Senior
SCOPE - VIT Chennai

Submitted By:

Ujjwal Pandey 19BAI1019
Pranaydeep Mayank 19BAI1026
Shivam Sibal 19BAI1006

ABSTRACT

As part of the CSE1014 Project component a 2d grid-based game- ”**COIN COLLECTING BOT**” has been developed which is powered by AI algorithms. This game is developed with an objective to visualize the working of different AI search algorithms through a game. This game is built with python and pygame for the graphics.

The game has a human player named *“Mariko”* whose job is to collect all coins given in the grid, and also 5 enemy bots whose job is to find and kill the human player. If the player collects all the coins given in the grid it wins. The 5 enemy bots are powered by 5 different AI algorithms. If the human player is intercepted by any of the enemy bots, the game gets over.

This game also demonstrates the optimality and time complexity of various path searching algorithms in a creative way, where one can easily visualize how different search algorithm affects the decision each agent make to decide which path to choose out of multiple possible paths.

INTRODUCTION

This game has been designed to include the components and techniques learned from class Labs dealing with the various AI Algorithms as well as with Pygame. Our game “COIN Collecting Bot” is implemented with Pygame; however, the graphics display of the game, as well as most of the game state/controls, is entirely made from scratch by our team. The game engine portion of our project will be coded in python while the graphical portion of it is done in Pygame.

Game Description:

We have developed a 2d grid-based game which is actually a maze through which the human player (named Mariko) has to run through and collect the coins simultaneously.

Our game is having several modifications to any available 2-d grid AI game. Enemy bots are configured to attack and kill human players. Our game has 5 different enemy bots (named “RED”,” Blue”,” Yellow”,” Pink”, and “Black”). Our game is based upon a point system. It is a one-person game, if the player gobbles up a designated number of coins in the game, then the player wins. Our enemy bots will be competing from head to head with their different AI engines to kill the player. When the game is in player mode, a win condition is given to the highest-scoring human player, regardless of the AI

effect. Additionally, our game state will contain “Magic Pellets” which will offer players special powers such as attack capability, invincibility, wall smashing…etc.
Software/Game Engine: The software side of our project is coded in Python. Pygame libraries have been used for developing game graphics.
The game engine will store the information of the game map along with the states of the player and enemy units. The game engine is divided into the following major components.

- **Game State**: The game state is represented by the grid[][] data structure, it is essentially a 1200X800 two-dimensional integer array.
- **Playing State:** It is a working game with moving enemy bots and human players.

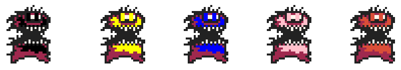
Artificial Intelligence:

It was decided that the AI algorithms would be implemented on our enemy bots, with all 5 enemy bots powered with different algorithms. Thus we wrote our 5 different AI algorithms from scratch.
The basic AI unit of our game conforms to the following algorithm.

1. Do a lookup on the player, and search for the shortest path to catch the human player by Euclidean distance and Manhattan distance.
2. Once found, check human players to go for the attack.
    a. If not found, proceed on the path to the furthest distance, and maintain distance until the player’s attack capability wears off, then go back to step 2.

3. If a player isn’t attacking or invincible, calculate the shortest path to the player and proceed to intercept.
4. If a player is intercepted by the AI enemy bot, it is dead.
5. If all players have attack capability, proceed to stay away from every player.
6. If the player is dead, proceed to deactivate.
7. If the player wins …proceed to deactivate.

Player and enemy bots:



METHODOLOGIES USED

This particular project is built completely on python with the help of pygame as its subsidiary library. The basic mechanism of game development has been utilized in the project along with AI search algorithms which we will discuss in detail here.

The game grid is a 1200 x 800 pixels grid with each cell measuring 24 x 24 cell pixels. Now, this particular grid is made using a map editor software called tiled. (https://www.mapeditor.org/download.html). Now this particular map file is mapped/embedded into a .lay file (***grid. lay***) in the form of various symbols

- **“*”**:- the outer environment of the screen
- **“#”**:- the border, walls, and obstacles or basically all the unmovable grid are represented by using this symbol.
- **“1”**:- represents the movable grid without coin
- “**2”**:- represents the movable grid with a coin.

Now the coin here represents the reward that is beneficial to the player.

Now, for the sprites of various characters, it has been taken online mostly from [https://itch.io/game-assets/free/tag-tilemap] and [https://phaser.io/examples/v2/category/tile-sprites].

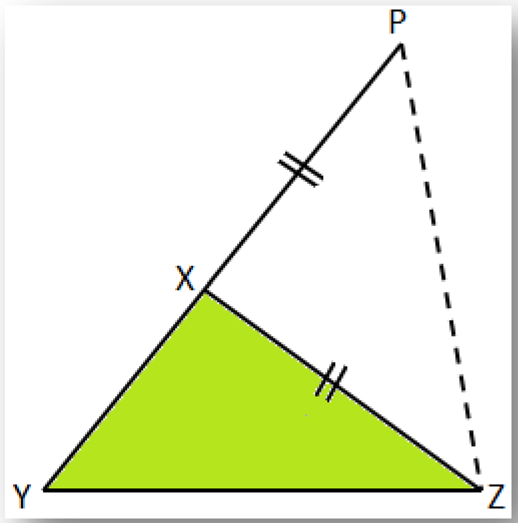
Now the game runs at 60 fps meaning that the screen refresh rate with each clock tick is 60. We have 5 agents as mentioned in the introduction part which is represented by each colored sprite. Now each agent utilizes one of the various search algorithms implemented to devise a strategy and find the optimal path to ambush the player. The agents are as follows.

- **BLACK**:- This character utilizes an Astar search algorithm with Euclidean distance as heuristics to ambush the player. As the Astar algorithm is mostly optimal its speed is 1/8th that of other agents to allow the player fair gameplay.

- **RED**:- this character utilized BFS as its search algorithm. BFS does not require any heuristics as it is a complete search but is mostly slow. Hence there is a clock that is set to some initial value 5, in this case, meaning that whenever the clock turns 0 then only a search will be made. This is done to optimize the time which BFS mostly takes. Also, the speed is more average than other characters.

- **PINKY**:- This character utilized greedy BFS as its search algorithm.

This is the fastest character out of all. As greedy BFS requires heuristics, hence we have used Manhattan distance as its heuristics. Now manhattan distance as you will find is not much optimal than the heuristics due to the following reason



- The hypotenuse being the largest side is what is calculated by the euclidean algorithm. Now Manhattan calculates the sum of the other two sides. Now by the property that the sum of two sides is always greater than the third. Hence Manhattan is somewhat bad heuristics.

- **BLUE**:- This particular character also utilizes A Star but with Manhattan as its heuristics. Its speed is also that of average and it utilized pinky position to create an optimal blockage of the certain grid so that players may find it hard to cross this.

- **YELLOW**:- This character utilizes UFS (Uniform Cost Search) as its search as its algorithm along with a clock tick of 3 to prevent it from oscillating. This is a common problem in Dijkstra Based algorithms where when more than one optimal path is available then we may be stuck between oscillations.

Now the movement of each character be it the player or enemies is based on Vector pair of (width, height) as this is the coordinate system of pygame library, which is non-trivial to what the search algorithm does hence helper library has been provided to assist in the conversion of co-ordinates in and out. Each search algorithm returns a Path class which stores the current path and the current best cost. Whenever a better path with the better cost is found in the subsequent iteration/tick of the game the path vector is updated. It also allows easy pop and pushes operation to get the latest direction where the agents need to move.

Details about each file

- **Main.py**:- The runner file of the whole application
- **App.py**:- The main controller class for the whole components. This is where the game iteration occurs. The grid is also drawn in this class. Also, all the updates and character control are done through this class

only.

- **Mariko.py**:- The player class.
- **Helper.py**:- This where all the helper modules are defined.
- **Setting.py**:- Various settings for the game.

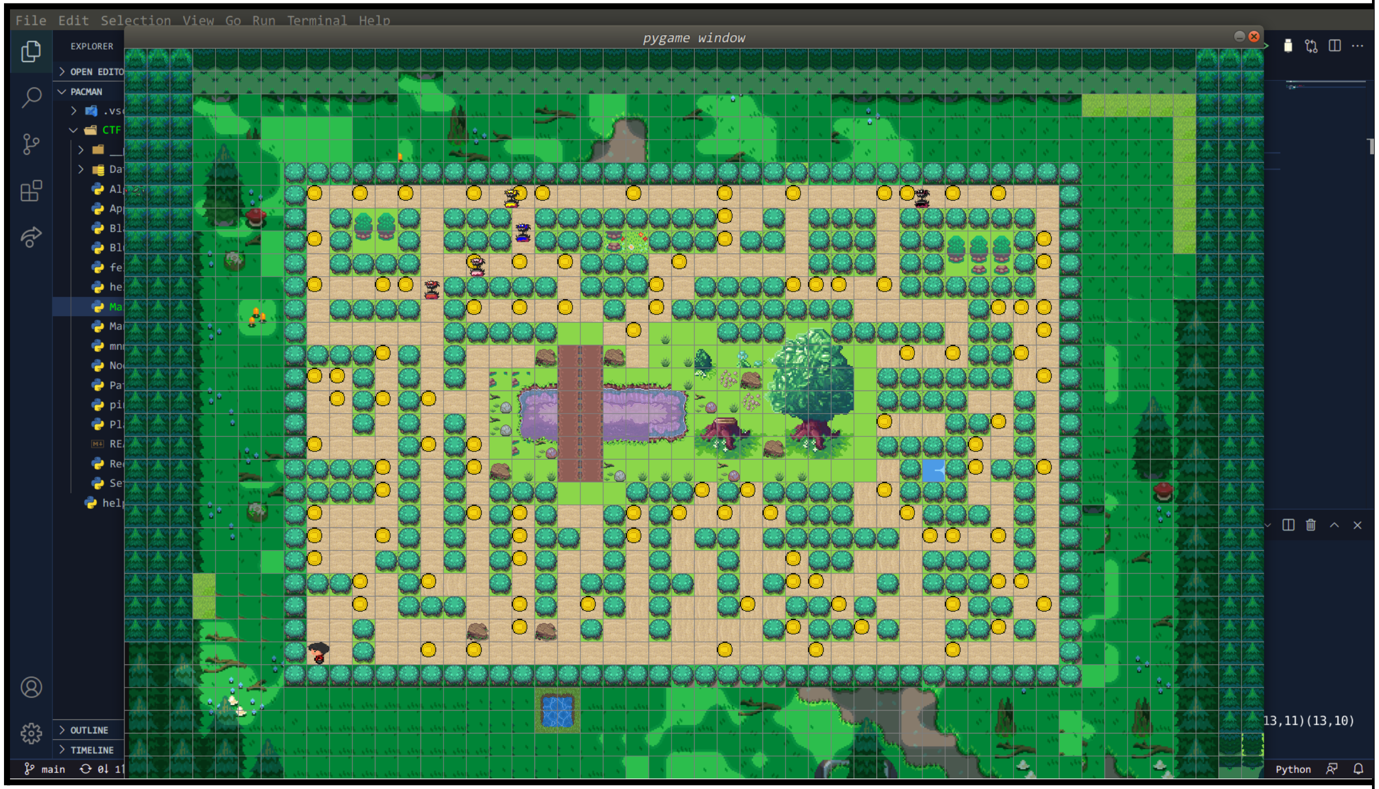
Rest all are agents file where each specific agent’s behavior is defined as mentioned previously.

IMPLEMENTATION

The implementation of our project is very big and contains multiple files and images. As a result, we are providing the respective GitHub repository as an alternative.

[https://github.com/Ryednap/Maze-Game-using-AI-algorithms.git]

OUTPUT/SCREENSHOTS



This is Starting screen of the game.

For the demonstration video please see the **demo.mkv** video file in the **GitHub repo**, as this is a game so output can be only demonstrated with the help of video.

CONCLUSION

In this report, an introduction to game architecture using pygame is given. Then the concept of basic search algorithms in AI is explained. In further progress, artificial intelligence using search algorithms in a maze /grid is illustrated with the help of enemy bots moving on a particular search algorithm programmed on it. Finally, the underlying topic of how to implement AI-based algorithms in game development is briefly illustrated.

For further development, there would be an attempt to automate the player movement with the help of an AI algorithm to ensure high quality and flexibility in-game implementation.

REFERENCES

- https://realpython.com/pygame-a-primer/
- https://towardsdatascience.com/ai-search-algorithms-implementations-2334bfc59bf5
- https://www.mygreatlearning.com/blog/a-search-algorithm-in-artificial-intelligence/
- https://stackabuse.com/basic-ai-concepts-a-search-algorithm