PYTHON PROJECT

Title: Development of a Maze Game Using Python

Mohamad Youssof BARAZY, Nicolas REBOULLET, Chia-Ten TANG, Ariel TORJMANE

Abstract: The project is a game developed using Python and the turtle library. The game is a maze game, where the player navigates through a maze to reach the end while avoiding obstacles and enemies. The game features a randomly generated maze, a player character that can move in four directions, and enemies that move towards the player. The project is designed to be challenging, but also fun and engaging. The use of classes and object-oriented programming concepts is demonstrated throughout the project, and the game is designed to be easily extensible and customizable. Overall, this project serves as a solid introduction to game development using Python and the turtle library for beginners

January 2023

A- Introduction:

We decided on a maze game, because of how much it will boost our knowledge in python.
A maze game written in Python can help beginners learn basic programming concepts such as loops, conditionals, and user input. The game can also introduce beginners to 2D arrays and basic graphics concepts. Additionally, creating a game can be a fun and engaging way for beginners to practice and apply their coding skills.

There are many ways a maze game could be implemented in Python, but here is a general example of how the code might be structured:

1- Define the maze: This could be done using a 2D array, where each element represents a cell in the maze. For example, a 0 could represent a path and a 1 could represent a wall.

2- Create a player object: This object would have properties such as the player's position (x, y coordinates) and the player's current direction.

3- Create a function to check for valid moves: This function would take in the player's current position and direction and would return a Boolean indicating whether the player can move in that direction (i.e., whether there is a wall blocking the way).

4- Create a game loop: This loop would handle all the game logic, such as checking for player input, updating the player's position, and checking for a win condition (i.e., the player reaching the end of the maze).

5- Create a function to render the maze: This function would take the 2D array representing the maze and the player's current position and would use a library like turtle or pygame to display the maze and the player's progress on the screen.

6- Implement the main function that ties everything together.

This is just one example of how a maze game could be implemented in Python, and there are many ways to approach the problem. The key is for beginners to practice and apply the basic programming concepts they have learned, such as loops, conditionals, and user input, in a fun and engaging way.

B- Description of the 3 stages

V0: This code is using the turtle library in python to create a game that displays a maze. The game is set up with a black background and the title "Bit Maze Game" and the window size is 700x700.

The code defines two classes, Pen and Player, which will be used to represent the walls of the maze and the player that will navigate through the maze. The Pen class inherits from the turtle. Turtle class, and the Player class inherits from the Pen class. The Pen class is used to create the walls of the maze and the Player class is used to create the player that will navigate through the maze.

The levels list is created and initially set to an empty string. level_1 is created as a string with x's representing walls and p representing the starting point of the player.
The program has an infinite loop that updates the screen, allowing the player to move around the maze.

The program allows the user to move the player using the arrow keys, and the player can move around the walls of the maze. The program also makes use of turtle's built-in "stamp" function to display the walls of the maze.

V1: This code defines five classes, Pen, Player, key, Treasure, heart, and Enemy, which will be used to represent the walls of the maze, the player that will navigate through the maze, the key to open the treasure, the treasure, the heart for health and the enemy to fight.

The Pen class inherits from the turtle. Turtle class, and the other classes inherit from the Pen class. The Pen class is used to create the walls of the maze, the key class creates a key on the map, the Treasure class creates a treasure on the map, the heart class creates a heart on the map, and the enemy class creates an enemy on the map.

The levels list is created and initially set to an empty string. level_1 is created as a string with x's representing walls, p representing the starting point of the player, 'E' representing the enemy, '.' representing the heart, '$' representing the treasure and 'k' representing the key.

The function setup_maze(level) sets up the maze by iterating through the level string and placing walls, player, key, treasure, heart, and enemy based on the characters in the string. It also set the turtle to listen for key inputs for player movement.

The program allows the user to move the player using the arrow keys, and the player can move around the walls of the maze. The program also makes use of turtle's built-in "stamp" function to display the walls of the maze. The player must collect the key, treasure and heart while avoiding the enemies to win the game.

V2: The main difference between V1 and V2 is the introduction of collision detection between the player and the enemy, and the ability for the player to collect the key, treasure, and heart. When the player comes into contact with the key, treasure, or heart, the item is "destroyed" by being moved off the screen and hidden, and the player's key, gold and health attributes are updated accordingly. In addition, the shape of the player is changed to reflect the direction it is moving in.

The Player class is updated to include a function called is_collision() which takes in another turtle object as a parameter, which is expected to be an instance of the Enemy class. It uses the Pythagorean theorem to calculate the distance between the player's current position and the position of the enemy. If the distance is less than 5, the function returns True, indicating a collision has occurred.

Additionally, key, Treasure, and heart classes are updated to include a destroy() function. This function moves the turtle object off the screen by setting its x and y coordinates to large numbers like 2000 or 2010, and then hides the turtle object using the hideturtle() function. This effectively removes the key, treasure, or heart from the game, and make it not visible to the player.

In the Player class, the shape of the turtle is changed to reflect the direction it is moving in by using the seth() function. Which is a turtle function to set the heading of the turtle object to a given angle, where 0 is east, 90 is north, 180 is west, and 270 is south.

The game also uses the turtle listen() function which listens for keyboard events, and turtle onkey() function which binds an event to a key press. These are used to move the player around the maze based on the arrow keys pressed.

C- Debrief:

We started as building a regular maze, but we figured out that it wasn't as challenging as we wanted our project to be. So, we decided to improve our project but keep the maze as the skeleton of it. Thus, we wanted our project to be a small game using the maze as the interface. During this game, the player will be in the maze with "enemies" and he shall get out of the maze by avoiding those enemies. He also can collect circles, to get more health to get the treasure.

The first thing that didn't work well was to randomly creates walls to make it harder for the players with different levels. But then we decided to pre-set it and change the idea of our game, and we realize it was better with enemies. Therefore, it took us some time to generate the enemies, to decide how they should look like, how they could move because we could only make the player move when we play. It turned out well, and this version of "Pac-man" maze game was a better idea.

The hardest part to develop in our project was without a doubt the V1 version. We had to change the V1 version several times to make it more efficient. For example, we first tried with one enemy and one treasure. Then we add the keys, and in the end, we thought we could add circles to give the player more health.

The design was a big part of our project, and we had to understand how to use it efficiently. Indeed, we use graphic design to get a better look for our player, enemies, and the treasure.

Because we had to drop the randomness of the game, now that we have a game which works correctly, it will be interesting to continue from here and add the randomness in the game. The difficulty will be to make it random, but with always a solution for the maze.

There are many ways to upgrade the game, here are a few possibilities:

1- Add more levels: Adding more levels to the game by creating new level strings and adding them to the levels list. This will give the player more variety and make the game more challenging.

2- Add more items: Adding more items to the game, such as power-ups, weapons, or special abilities. This will give the player more options and make the game more interesting.

3- Add a scoring system: Adding a scoring system to the game that keeps track of the player's progress, such as the number of keys collected, treasure found, and enemies defeated. This will give the player a sense of accomplishment and motivation to keep playing.

4- Add a high scores list: Adding a high scores list that keeps track of the top scores for each level. This will give the player a sense of competition and motivation to beat their previous scores.

5- Add sound effects and music: Adding sound effects and music to the game to create a more immersive experience. This can help to set the mood and create a more engaging atmosphere.

6- Add an AI for the enemies: Adding an AI for the enemies that makes them move and act more realistically, such as chasing the player or avoiding obstacles. This can make the game more challenging and engaging for the player, as they will have to use strategy and skill to avoid or defeat the enemies. The added complexity of the enemy behavior can make the game more interesting and dynamic, giving the player a sense of unpredictability and excitement.

D- Conclusion:

For beginners writing this code, some interesting things we found out and we focused on were:

a. Understanding the basics of turtle graphics: The turtle library is a great way to introduce beginners to the concept of creating graphics in Python. By working through the code and experimenting with different shapes and colors, we gained a solid understanding of the basics of turtle graphics.

b. Learning how to control the turtle using the arrow keys: The code uses turtle's listen() function to capture key presses and move the player turtle in different directions. we learned how to control a turtle using keyboard inputs.

c. Understanding the concept of nested loops: The code uses nested loops to iterate through the level string and create the maze. We understood how to use nested loops and how to use them in the context of a game.

d. Understanding the basics of object-oriented programming: The code uses classes to create different objects such as the player, walls, key, treasure, heart and enemy. We understood the basics of object-oriented programming and how classes and objects work in Python.

e. Understanding basic game mechanics: The code implements basic game mechanics such as collision detection, player movement, and item collection. We learned more about basic concepts and mechanics behind creating a game.

The two most important parts we highlighted as a group were:

1. The map of the maze:

For the map of the maze, during the period of the development, we found it difficult and complicated to let the programming produce a pure random map. The algorithm will become more complicated with the growth of maze size, so at the end, we decided to sketch out some mazes as the default database of maze and let the programming randomly pick one in our project. Furthermore, we wanted to increase more entertainment into the maze. That's why we turned our project into not only a pure maze game but a special version of the "Pac-Man" maze game.

2. Adding Opponents' cool feature

As a “Pac-man” maze, some opponents should not be missed in the game. Based on this concept, we made them move by themselves. Although in the beginning, we tried to present the path of opponents to attack the player, there still existed some problems such as too much dead-end path for opponent or no possible way for player to dodge them. However, after the class of linear algebra, we noticed that the “Euclidean distance” may be useful for opponents to calculate their own path.

Basically, the first point highlights the challenge of creating a random maze and the decision to use pre-designed mazes instead. The second point highlights the importance of adding opponents to the game and the use of the Euclidean distance to calculate their movement.

Besides these 2 parts above, during this project, we also realized the power and convenience of Class in python. If we had to do repeated code for the same type of object, the length of our final code would be way longer and more redundant. And we also noticed that when trying to code some topic out, we should always analyze the topic in both conceptual and mathematical ways to have the correct logic for the project. Additionally, the importance of using classes in Python to avoid redundant code and the importance of naming variables clearly for easier collaboration is also noted. These are important considerations to consider when developing a game like this.

To upgrade this game, one could consider adding more levels and mazes, adding different types of opponents with unique abilities, and incorporating power-ups or bonuses to make the game more interesting. Additionally, one could consider adding a scoring system and a way to save progress.