Python Short Project

Maze - merizing

-

Adam EMMANUEL

Academic Manager

EPITECH

Preface

This short project aims at teaching the students with the basics of several key aspects of programming.

EPITECH's methodology will be used to achieve the short project.

Depending on the students' level, the project length is divided in several bullet points.

Thus, the students can always get new knowledge although the short project is not finished.

Max. Number of students: 15-20 students

Equipment needed: student's own laptop

Software to be installed the day of the workshop: Python 3 compatible with Mac, Windows or Linux.

Content

The following topics will be covered. The purpose of the workshop is to introduce complex topics using a **project-oriented approach** so students may enjoy and discover the EPITECH learning process.

Technology:

- Python 3.7
- UNIX/Windows Terminal

Topics:

- Basics of Programming
- Basics of Python Language
- Basics of Algorithm
- Basics of Data structure
- Basics of Object-Oriented Programming[1]
- Basics of Artificial Intelligence
- Basics of Design Pattern[2]

[1] Basics of Object-Oriented programming is a notion covered in the bonus parts of the workshop
[2] Basics of Design Pattern is a notion covered in the bonus parts of the workshop

Subject

Part II: Living Maze

1)Define a maze with a data structure

To make a maze, we first need to define how we will represent it in our code.

Let’s keep it simple and represent the maze as a grid. A cell of the grid can be free (path) or full (a wall).

Hint

We will use number to represent these states.

Ex: 0 for path and 1 for wall

Hint:

A grid is a succession of cell group as row first, then a grouping of these rows makes a grid.
How to represent this with the right data structure is the key of this exercise.

2)Display our maze

Now to display it, we will use ascii art, or graphic representation with letters. To put it simple, a path cell is represent with a space character “ “, and a wall cell is represented with a x, “x”. A printed maze should look like this on your terminal:

```
x   x x x x x x x x
x       x x x x x x
x   x   x       x x
x   x   x   x   x x
x   x   x   x     x
x   x       x x   x
x   x x x         x
x   x x x   x   x x
x
x x x x x x x x x x
```

Hint

If you don’t know how to proceed at all, you should take a look at the Introduction document, “loops” and “lists and dictionaries”

3)Better with Entry and Exit

To be a proper maze, our maze needs an entry and an exit. Like the previous part, the entry and the exit must be represented with different numerical values.

They will be printed as "E" for entry and "C" for exit.

```
x E x x x x x x x x
x       x x x x x x
x   x   x       x x
x   x   x   x   x x
x   x   x   x     x
x   x       x x   x
x   x x x         x
x   x x x   x   x x
x                 C
x x x x x x x x x x
```

Bonus

As a bonus, try to export it (print it inside a file) and try to import and display it.

Part III: Let's take a walk

1)Pawnify

To make a game out of our maze, we first need a pawn to be move around. This pawn must have several characteristics and capabilities:

- A position (coordinate, x and y)
- A representation inside the maze

To handle the later development of our game, we will start with a clean architecture. Let's put all the pawn related data in a single data structure, named player, that we will update as we improve our game.

Hint:

This kind of data is usually abstract with an object/class, but a dictionary might also do the trick…

A real game should also keep running until we quit the program. This is what we call an event loop, which mean that the program keeps running until we told it we quit.

An event loop is just a loop calling the game different functions in a predefined order until some specific action tells the loops to end and terminate the program

Hint:

Usually, we use a "state variable" that can ne modify by several function to tell the loop to stop if necessary

2)User Input

Now that we have a pawn and we can display it, we need to move it around the maze, respecting some specific rules.

- The pawn cannot go through walls
- The pawn cannot go beyond the entrance or exit
- The pawn cannot go beyond the limit of the maze (outside walls)

Hint:

The principal element of the event loop is updating the state of the game after getting the user input. Among these updates, we have the player position, and displaying the new game state...

3)Victory

Now that our game is almost finish, we need a way to actually "win". The winning rule is simple, reach the exit cell.

As the game ends, we should let the user knows that he won with a success message.

Bonus

Encapsulate the data structure "player" inside a class

Make a Maze class and put all the maze logic inside (generating, export, Import, displaying, winning...)

Part IV: Solve the maze

Being able to play the maze was a first step. Now let's see how to solve the maze without any human action, in other words, with an Artificial Intelligence.

Balance between accuracy, complexity and speed.

The easiest way to implement a simple and fast IA is a backtracking algorithm. By backtracking, the "AI" will go around the maze in every cell it *didn't visit before* until it reaches the exit. If it *gets stuck* in a dead-end, it must be able *to go back on it steps* until it finds a *new cell that it didn't visit before*.

To achieve such a thing, we need *2* data structures. One where we can store the visited cells and one where we can store the path taken, cell by cell, so we *can go backward*.

Hint:

Don't forget that if you go backward, you should "remove" the last cell from our current path.

One thing with this algorithm us that it is not really smart. It will basically test all the possible paths until it finds the exit. It can be very fast or very slow.

One of the preferred algorithms in video game is the A* algorithm. But once again, to keep it simple, we will modify our current solution with one of the principles of A*, the heuristic part.

In simple words, we will modify our AI by telling it where is the exist so it can make better decision on what cell to go when it is choosing.

Part V: Insane maze

Add randomness based on event

Redefine the maze if the player arrives close to the end

Hint:

For this part, please refer to the workshop teacher

Part VI: Enemy Factory

Add enemies using the previous developed AI

Decorator on enemies for different types

Factory for enemies

Hint:

For this part, please refer to the workshop teacher