ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

06/12/2017

Shop Quest

Maze solving game against AI

BAALI Karim / ALGERA Pieter
ADSA CORP – IBO 1

Table of contents

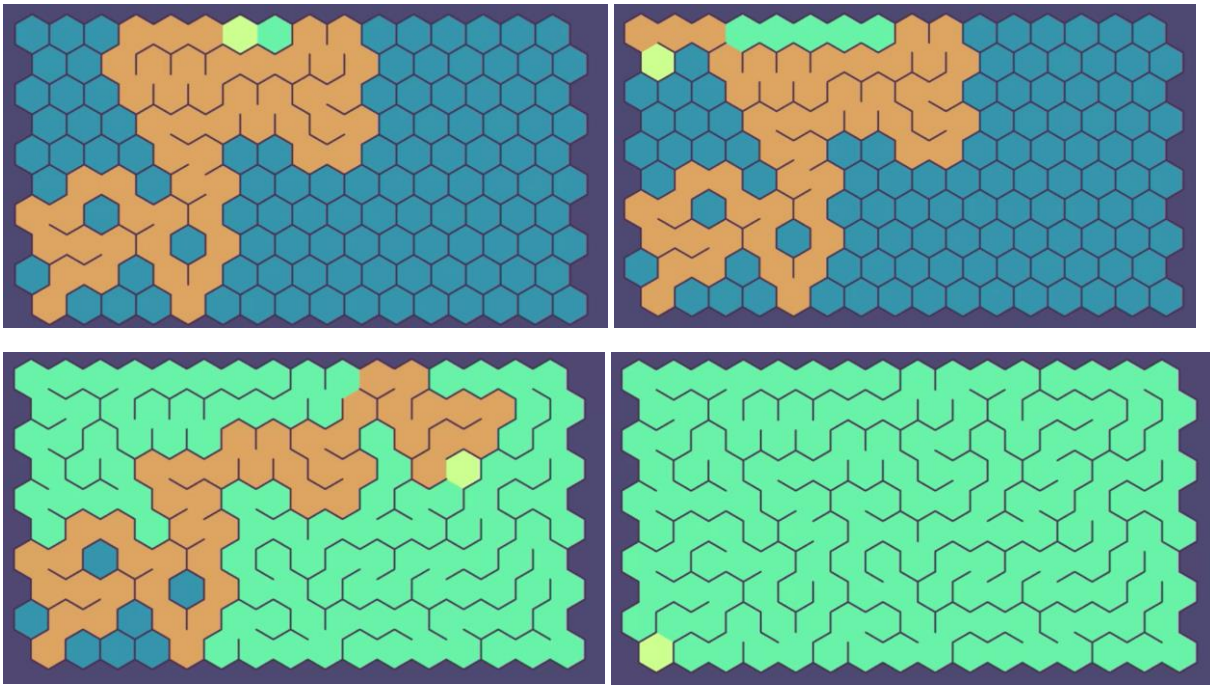
Game introduction

You wake up in a big shop … which looks really strange. After a few seconds, you understand that it's a maze… a real one! Your goal? Find the wonderful banana before the AI. Who's gonna be the faster one?

For each game, a new maze is generated by our program, allowing you to discover a whole new maze whenever you want. Using the directional keys, your job is to go through the maze and find the holy banana. But you're not alone! Another player, controlled by our *very* smart AI, will try to reach it before you, and its maze solving ability is pretty efficient.

Maze generator

First things first, the maze generator. To be able to provide a randomly generated maze for each game, we decided to implement a depth-first recursive backtracker algorithm. Starting at a point, the algorithm chooses a random neighbour of the current cell, if it's unvisited, so it marks it as visited, remove the wall between the two cells, add it to the path, and move on it. If there is no more unvisited cell in the neighbours list, it goes back to the last one on the path, and applies this logic until all the cells have been visited.



(Source : https://en.wikipedia.org/wiki/Maze_generation_algorithm)

In the example above, the visited cells are in orange, when the algorithm can't find any unvisited neighbours (image 1), it comes back until it finds a new possible path. The cells which are on this backtracking path are marked in green. At the end, we can see that all the visited have been visited and there is no more unvisited cells.

To simplify the work on the generation, we decided to dissociate the logical matrix of the maze and the displayed matrix. In the logical one, each cell has data about its wall state (using a boolean array as attribute). It's this one which is used by the algorithm to generate the maze. Then, when the generating part is over, we update the visual one, used for file printing and display. Its size is bigger because each wall has a dedicated place in the matrix. So we can switch from the logical one width and height to the visual with the following formula:

visualWidth = (logicalWidth * 2) + 1

visualHeight = (logicalHeight * 2) + 1

Implementing the matrices this way is a bit tricky, but it felt like the easiest way at the moment. It could be useful to go through the process again to simplify it later.

When everything has been updated, the visual matrix is printed in a file, which will be read by the second part of the program to create the graphical interface.

Artificial Intelligence for maze solving

In our game, we have an AI trying to find the banana before the player. The AI uses the A star graph traversal algorithm to find the shortest path between its position and the position of the banana. To make the game a little easier for the player. When the AI is searching for a path, we randomly choose cells to block so that the AI has to find another path. If the only path is blocked, the AI will move in a random direction to simulate thinking. When the AI has found a path, it will start walking in the direction that it has to go. If the player finds the item before the AI, the AI will find a path to the next item and start moving as soon as it finds the new path.

Game logic using pygame

The game is made around a gameloop with events. The game starts by generating the maze and reading the generated maze from the text file. The maze is then passed through a function that creates the graph. We also figure out which types of walls to place where. At this stage, the player and the AI are placed in 2 positions on the map, the window is initiated and the screen is drawn. The gameloop is launched at this moment. The gameloop allows us to continuesly update the game and try and keep it running at a stable framerate. During the gameloop, we check for user input and update the game accordingly by moving the player in the chosen direction and checking if he has found the item. We then proceed to updating the AI. If the AI already knows where to go, he just moves in the direction he must go otherwise, he looks for a new path to the item. The loop finishes by updating the screen with the new data.

Improvement possibilities

Concerning the maze generator part, we could improve the handling of the matrices, which are separated between a logical and visual one. By merging those matrices and updating the algorithms, we could improve the global time and space complexity of the program.

As we split the project in two parts (generating / solving the maze and creating the graphical interface) to be able to work in parallel on the project, some data structures are a little bit redundant, and we could merge them by making little updates to the code structure.

We could create a better main menu and an in-game menu that would allow the user to change some settings, save and quit the game.

We could also add some more game elements like power ups and extra points.