**Project Name: ~~The AI’s maze~~ ExploreMaze!**

**Project Description:**

My project will be an interactive game where the player will have to navigate through a generated maze. To enhance the user experience based on this relatively simple game idea, the game will randomly generate a maze every time a new game is initiated, which means the player will have an unexpected playthrough every time they play the game. To further add a layer of problem solving in the user’s navigation, I will be narrowing the view of the maze displayed on the game screen, so that the player only sees a set tile which scrolls as they move around the maze. With a narrowed view of the entire maze, the user cannot determine the exit point immediately, creating a need for exploration.

**Competitive Analysis:**

Maze navigation is a well-known idea that has not only formed many maze-based games across mobile app stores, but has also incorporated in famous, high level games such as Pokémon or Breath of the Wild. However, for the coding level of my project, while I was inspired to pursue a maze-based idea from my experience with high level game maze features, I sought more simple maze project ideas to judge my own game against.

From my investigation into archived 15 112 project libraries, maze-based games are generally based around a Pac-man styled interface and gameplay, where the full maze is displayed on the screen and the user navigates while avoiding enemies. Though a version of these Pac man games has implemented AI algorithms to better navigate enemies, none of these demonstrate randomly generated maze structures, making this aspect of my maze interface unique. One game that leverages a maze-based structure, “Labyrinth”, does demonstrate an aspect of random maze generation based on user input, but this game still displays the full maze for the user, making my game unique with its tiled view.

In terms of similarity, my game idea uses the foundational aspect of any maze’s navigation-oriented puzzles, which all maze-based games possess.

**Structural Plan:**

There are five main components to my game, each of which I will isolate into its own python file, then import into the main game interface file to call the contained functions.

The first component of my game is extracting the tile size from a given black and white maze image. This file will involve 2 functions that examine the maze image and will extract the smallest confined ‘corridors’ vertically and horizontally. The tile will then be expanded onto a blank image to ‘blit’ directly onto the game screen, which will be done by an enlarging function.

The second component of my game will be a file that processes hundreds of black and white maze images, manually made the same size, and stores the pixel colors of each image into a text file. The file will also do the same for images that are not mazes, indicating in each written line whether the pixel colors indicate a maze or not. The resulting file will be used to set up my machine learning model to generate random mazes. Because of the linear-processing nature of this component, this file will consist of very few helper functions, and instead be direct code.

The third component of my maze game is reading my maze data file, extracting out ‘True’ and ‘False’ mazes, and using this data to form 2 machine learning models. One will be able to verify whether a given image is a maze, and the other will be responsible for actually generating the random maze itself, as an image.

The fourth component of my maze is taking my maze tile image and directly editing it so that when the tile is displayed it has a visually appealing color and layout. This will serve to improve game immersion. For TP1, this file will simply consist of functions that find the margins of a maze image and find the entrance and exit.

The final component of my game is the actual game interface. This file will generate the game screen, store the player’s navigation sprite and its movements in various functions that verify the validity of moves, in a Player class. The random generator’s maze image, modified with the correct graphics and tiling, will be loaded, and displayed in a scrolling format through this file.

**Algorithmic Plan:**

The most significant and the most difficult aspect of my game is the random maze generation aspect. I plan to accomplish this by using scikit learn’s DecisionTreeClassifier model.

First I will form a model that takes either 1 or 0 (indicating if the considered image is a maze (1) or not (0)) as the input and is given each pixel’s color values, with r, g, b values all inputted as individual numbers, of the maze images. This data will be obtained from a data file which is created by a separate python program by processing each selected maze image and writing their pixel values, as well as their 1 or 0 validity indication, to the file. This model should then generate a maze when passed 1 as the prediction value to the function .predict().

However, my tech demo and external research has made it unclear whether this model will generate a different result for every .predict(1) call due to small data scopes, so if this fails to generate random maze data by indicating pixel color values for an image, I will use the DecisionTreeRegressionModel, passing it each x,y coordinate (as a single 4 digit integer) and the color that location has stored in each valid maze image only (where 1 means white and 0 means black). I will then use this model, stored in a variable, to make predictions for each coordinate in the image file to host the random maze.

The resulting maze image will then be verified by a separate model that will take the pixel color values of valid and invalid maze images as input and will be given results that indicate whether the pixel values map to a ‘True’ or ‘False’ image, as strings. This model will be formed with scikit learn’s DecisionTreeClassifier.

**Timeline Plan:**

Monday, 12 April: have a basic maze image rendered onto the screen, with sprite positioned at entrance, and with sprite being able to navigate without going through walls.

Monday, 19 April: randomly generated maze and verification complete and functioning

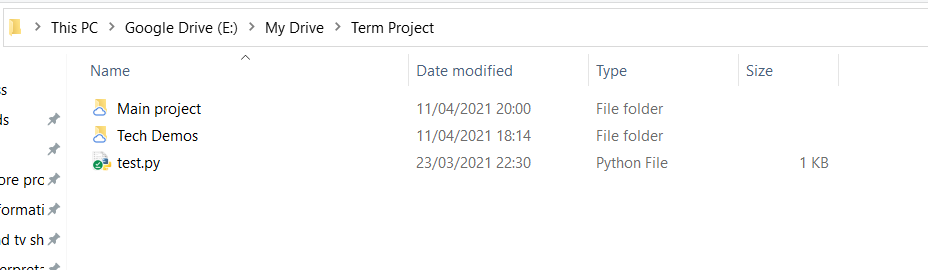
Tuesday, 20 April: scrolling tiled view of maze applied to screen.

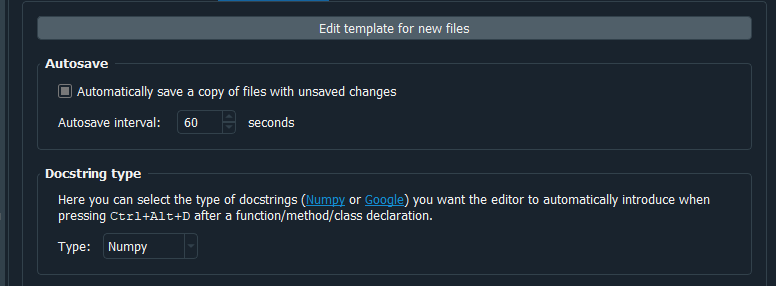
Wednesday, 21 April: game menu implemented in main game file and show game complete banner when maze completed instead of closing game.

Friday, 23 April: chosen graphics look applied to tile view of maze in app.

**Version Control Plan:**

I will be storing all my python and associated image files in google drive. I have downloaded google drive’s desktop client, allowing me to directly save python files onto my google drive. This means that whenever I save my python files, it stores onto the google drive cloud, allowing me to access the files from a different computer in case of an emergency. My python interpreter is also set to save my code every 60 seconds.



**Module List:**

* Pygame
* Scikit Learn

**TP2 update:**

**Structural Plan:**

There are five main components to my game, each of which I will isolate into its own python file, then import into the main game interface file to call the contained functions.

The first component of my game is extracting the tile size from a given black and white maze image. This file will involve 2 functions that examine the maze image and will extract the smallest confined ‘corridors’ vertically and horizontally. The tile will then be expanded on the game screen by expanding pixels to enlarged rectangles with pygame’s draw.rect function.

The second component of my game will be a file that produces a 20 by 20 or 25 by 25 square grid. Each of the two grids will be formed by their own functions, using helper functions to draw the border and the horizontal and vertical lines. Because the grid sizes are predetermined, the grid will be drawn with hardcoded values and will be saved under the filename “grid made.png”.

The third component of my maze game is actually generating the random maze itself. This file will first form a maze path in one function and then adjust the remaining cells to form the rest of the maze. Each of these two functions will form the maze paths by removing lines from the squares in the grid previously made. The removal of lines will be aided by helper functions that each remove the top, bottom, right or left line of a square.

The fourth component of my maze is taking my maze tile image and directly editing it so that when the tile is displayed it has a visually appealing color and layout. This will serve to improve game immersion. For TP1, this file will simply consist of functions that find the margins of a maze image and find the entrance and exit.

The fifth component of my game is the classes that will aid my game interface. I will have a file that will store all my game classes. The first class will store the player’s navigation sprite’s class, which will include functions to allow the sprite to make valid moves. The second class will store the width, height and top right x,y coordinate of the tile being displayed from the maze, as well as functions that allow the tile to scroll down onto the next tile segment while not scrolling past maze bounds. The final class will be a button class that stores the location and size of a button and will have a function that determines if a user’s mouse click was on the button, which will be used to carry out actions in response to button clicks.

The final component of my game is the actual game interface. This file will generate the game screen, store the player’s navigation sprite and its movements in various functions that verify the validity of moves, in a Player class. The random generator’s maze image, modified with the correct graphics and tiling, will be loaded, and displayed in a scrolling format through this file.

**Algorithmic Plan:**

The most significant and the most difficult aspect of my game is the random maze generation aspect. I will accomplish this by removing lines from squares in a square grid image using the ImageWriter library.

Firstly, I will write four helper functions which remove the top line of a square, the bottom line of a square, the right line of a square and the left line of a square. These four functions will take the top right coordinates of the inner square (i.e., excluding the outline size) and then use the outline size to find the x, y coordinate bounds of the line to be removed and then will set all the pixels in the bound to the background color, white.

Next, I will write a function that will form the solution path of the maze to ensure the maze is always solvable. This function will start at the top rightmost square and randomly select a direction from top, right or bottom, then remove the line in the given direction. This will then be repeated with the next square, which is selected depending on the direction chosen, until the right or bottom edges of the maze are reached, forming an exit.

The resulting path in the grid will then be processed in another function that will change the remaining cells to create the maze ‘look’. This function will go over each cell and randomly remove the right or bottom line, creating maze walls and turns. Particular cells that form turns in the solution path will be skipped over to maintain these turns.

Finally, the resulting maze will have isolated sections the user cannot access. Because the maze walls are made by randomly picking between right or bottom only, these sections are only connected to the right and bottom boundary of the maze, which cannot have their right or bottom boundaries removed respectively, in order to maintain the maze bounds. To open these isolated sections, all the remaining walls in the bottom row and rightmost column will be removed, excluding particular turning and exit cells from the solution path to section off the exit as much as possible.

**Timeline Plan Changes:**

*Tuesday, 20 April: chosen graphics look applied to tile view of maze in app.*

*Thursday, 22 April: scrolling tiled view of maze applied to screen.*

**TP3 Update:**

The following features were added to the game:

* A mini-view, where player can see the exit as a yellow dot, and their position relative to the whole maze as a red dot - this can be toggled on and off with the v key.
* A home menu screen with an information button in the bottom left corner and two mode playing buttons.
* Two different modes that can be chosen, a ‘normal’ which uses a 20 x 20 grid for the maze, and a ‘extensive’, which uses a 25 x 25 grid to form the maze, each accessible with their respective buttons.
* An option to return to the home screen during the game using the ‘h’ key - this ends the current game.
* A maze complete screen which lets you start a new game by choosing one of the level buttons or return to the home screen with the home button on the bottom left.
* Sound effects for when a player changes maze tiles, when they finish the game and when they collide with a wall.
* A loading screen that displays while the maze is rendering.

Only pygame was used as an external module.

Project Name changed: ExploreMaze!