**Maze Solving Application**

**A maze solving application that is able to visualise a run-time process, given user parameters.**

##### Contents

###### ANALYSIS

Defining Project Research

###### DESIGN

Prototyping (DPS Maze Generation)

###### TESTING

###### MAINTENANCE

###### USER GUIDE

###### EVALUATION

##### **ANALYSIS**

Background:

“I decided to choose this “

In this part you need to talk about why you chose this project, who is it for or if it is for anyone. Then start by finding examples online and try to evaluate these examples against a set of expectations and objectives (these objectives will be used in the end as well (call back)

##### **Design**

As per requested by my teacher, a prototype design of our program was required for the teacher to assess how we can produce a program that satisfies the application we decide to develop for the NEA. This maze was developed in order for me to understand the foundations required for a maze generation, and how I can tinker with certain parameters in order to fit the maze for my needs.



A screenshot of a computer program

Description automatically generated



In order to output the maze, I used the **pygame** module as it eases a lot of implementation restraints that I might face during the development of the program.

A grid of squares with a white border

Description automatically generated



Figure 1

Explaining DPS mazegenerator:



* Imported pygame because I will use its library in order to extrapolate tools that I will need to develop this program, it makes it much easy for me to do so.
* Using random, as when I “remove walls” in the maze, I will need it to be chosen without any bias.

A black background with white text

Description automatically generated

* A mixture of variables and constants
* The constant “RESOLUTION” will hold the WIDTH and HEIGHT variables using something called “tuple packing”, this will allow us to run the pygame framework (sc) using those variables, which the user can provide.
* The TILE constant is the given length of one side of a cell within the program, shown by figure 1
* Columns and rows are calculated by dividing the entire WIDTH and HEIGHT by the number of TILEs

A screen shot of a computer screen

Description automatically generated

* Pygame initialised.
* sc = the screen which we will draw upon (later in the program), and it is set to the RESOLUTION constant
* clock = the time clock which we will use to “tick” the program later

A black screen with colorful text

Description automatically generated

The Cell Class was created in order for us to create a cell for each column and row that intersects, within this space. Inside the constructor, we will create several attributes that allows for the cell to interact with each other.

The x and y attributes are initialised as the positions of the wall per cell within the maze.

When initialised, the walls of each cell will be given the value TRUE, this is so that the walls of surrounding each cell are initially present.

The visited attribute is indicated as FALSE in order for use to track whether or not the cell has been visited during the maze generation / traversal of the maze.

A screen shot of a computer program

Description automatically generated

This method is defined as the draw function and will be called upon every instance of the class.

To visually render the walls of each cell, we will use the pygame library and call upon the draw.line module. There are parameters given, such as SC (the screen where this will be rendered), the color of the line itself and the positional argument.

The positional argument uses the local variables x and y. As we initialised it early as itself \* TILE, this variable is given the value of the space between the starting point and the next line, this allows us to give us a space (the space being the value of TILE which can be changed before run time).