Decision-Making Maze

# Project Goals

The primary objectives of Decision-Making Maze is to create a simple text-based adventure game that can be used as a base for user to create their own Choose Your Own Adventure types of game.

# Significance of the Project

There isn’t really a huge ambition in this small and simple application that is developed in a relatively short period of time. It is not expect to be something that can lead to a decrease in crime rate, nor is it expect to be something that people will enjoy playing hours on end. It is only hoped that who ever came across this application will be encouraged to embark on a short journey to create their own simple applications or games, that might contributes to the well-being and happiness of others. Plus, my teammate and I, had a fantastic period of time bringing a rough and vague idea to life. It really sparked an immense amount of joy and happiness in both of us, when a function finally worked after when spending hours trying to figure out what went wrong.

# Installation and Usage Instructions

## Recommended software

* GitHub Desktop
* Pycharm
* Python 3.10

## Installation instructions

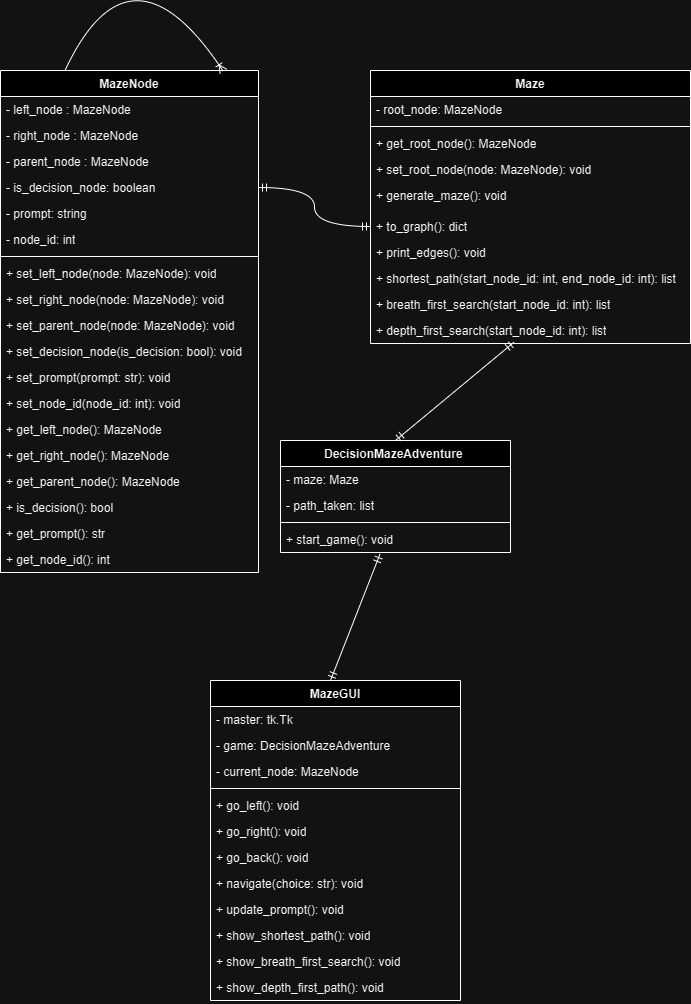
* Ensure Python is installed on the system.
* Clone the repository
* In the Python IDE of your desire, install necessary libraries (tkinter if not already available).
* Run the Python script in MazeGUI.py
* Follow GUI instructions to interact the prebuild maze sample.

## Usage instructions

* Upon running the application, a graphical interface should open.
  + If not check if tkinter is installed
  + Check if DecisionMazeAdventure.py and MazeGUI.py are in the same directory.
* Follow the on-screen instructions to navigate through the decision-based maze.
* Use the provided buttons ("<-- Left", " - Back -", " Right -->") to make decisions and move within the maze.
* Additional functionalities like finding the shortest path or performing BFS and DFS can be accessed via the corresponding buttons.

# Code Structure

## UML diagram



(Provided in file MazeUML.png)

## Explanations

* The code defines classes `MazeNode`, `Maze`, `DecisionMazeAdventure`, and `MazeGUI`.
* MazeNode represents nodes in the maze.
* Maze generates and operates on the maze structure.
* DecisionMazeAdventure manages the game aspects.
* MazeGUI creates the graphical interface for maze navigation.

# List of functionalities and test results

## Functionalities

MazeNode Class:

* Methods for setting and getting attributes of a node.
  + Left node
  + Right node
  + Parent node
  + If the node is a decision node
  + Node prompt
  + Node ID

Maze Class:

* Printing edges of the maze.
* Methods for maze generation.
* Methods for converting the maze from tree to a graph data structure.
* Methods for traversing the maze (shortest path, breadth-first search, depth-first search).
* Printing edges of the maze.

DecisionMazeAdventure Class:

* Initialization of a game instance using the Maze class.
* Recording the path taken in the game.

MazeGUI

* Methods for setting up the tkinter GUI elements.
* Methods for GUI interactions (navigating the maze, displaying paths).

# Discussion and Conclusions

Our application faced two key issues that we aimed to address given more time. Firstly, the currently maze creation process is very tedious, requiring manual creation of each node, resulting in significant time and mental strain. We're exploring a solution by developing a function to parse maze data from a CSV file, streamlining the node creation process. So that the player can have multiple maze ready to be load and interact with. Secondly, the maze feels somewhat empty for the player due to limited engaging functionalities. We aspire to introduce systems that enhance interactivity, such as a point-based system that replaces basic directional choices with engaging questions, requiring users to pick the correct answer from options. Additionally, our goal is to craft more complex mazes. Instead of repetitive navigation to locate the node with a positive outcome, we aim to establish a specific path that users must follow to traverse the entire maze and reach its conclusion.

During the development of this application we mainly applied the knowledge we gathered during the development of other assignment. Some of the more obvious application are the graph traversal algorithms breath-first search and depth-first search, which are also applied in finding the shortest path. We tried to implement Prim's algorithm and Kruskal algorithm, however we didn’t find a need for finding a minimal spanning tree in a maze application so we didn’t include them. Most importantly, we chose to use a tree data structure for the maze mainly because we realized that, similar to how a graph can be represented as an array of linked lists, something with similarities to a linked list could also be utilized