**# Project Overview**

The goal of my treasure game is for the player to navigate their way towards the treasure while collecting power ups and avoiding any traps. My game consists of a few main features. A randomised 5x5 grid meaning that the traps, treasure and power ups will be in different locations each time you play. A move function which allows the player to traverse around the 5x5 grid. Finally a search function that allows players to search the grid to aid them in avoiding traps and finding the treasure. The type of searches I will be using are: binary search, breadth first search and depth first search. Binary search will show the location of the treasure on the grid. Breadth first search will show the shortest path to the treasure, however it will not avoid traps. The player will be allowed to take one move at a time.

**# Challenges and Solutions**

Going outside of the boundaries of the grid causes the game to crash. To prevent this, I added this snippet of code. It checks that the player is inside the boundaries. If not the if statement will run code that prints a warning message instead.

A black background with text

Description automatically generatedA black background with white and blue text

Description automatically generated

I create my grid like this, however, sometime due to the random nature of spawning certain objects can overwrite each other.

A computer screen shot of text

Description automatically generated

This resulted in the game sometimes spawning an unwinnable grid like this.

A black background with white letters

Description automatically generated

The solution to this was very simple. I simply made the treasure spawn last instead which means it cannot be overwritten. I could also add an if statement at the end to check that both the player and the treasure are spawned and if not, it regenerates the grid.

I wanted the binary search to search the entire grid, this was difficult as it only works in a 1d array/list. To fix this I did some research and discovered you can convert to and from a 1d and 2d array which allowed me to search the entire grid for the treasure using binary search  
A computer screen shot of a code

Description automatically generated

**# Search Algorithms**

The use of a binary search algorithm in this project was redundant as the graph is randomised in nature. For a binary search to be effective it must be an ordered list, this means that using a binary search in this situation would be as effective as a regular linear search.

I implemented a breadth first search algorithm into my game. A breadth first search algorithm works by visiting all surrounding points on a graph and marking down their distance before moving to the next point.

A depth first search works by following a path as far as possible before backtracking and trying the next path.

A large issue I encountered while working on this project was figuring out how to apply breadth and depth first search to a 2d array grid with a randomized start and end point.

**# Flowchart**

The user will be greeted with a welcome message. After that the player will be presented with 4 choices. They can decide to move, Search the grid, check their HP or quit the game. If the player decides to move, they will be presented with 4 options representing the cardinal directions. After the player moves, if they are hit by a trap or collect a power up or find the treasure they will be notified. If the player steps on a trap and reduces their HP to 0 the game will end. If the player finds the treasure, they will be presented with a victory message and the game will end.

If the player decides to search the grid they will be presented with 3 options: Binary search, breadth first search and depth first search. Each one of these options will display different data about the graph such as the location of treasure or the shortest path to the treasure.

If the player decides to check their HP they will be presented with a message with their current HP.

The game runs within a while loop, the only way to break out of the while loop is to quit the game or to win or lose. This ensures that the turn-based mechanic works and the player is only allowed to make one move at a time.

**# Testing**

The first bug I discovered with my code while testing was a list index out of bounds error. I discovered that this was being caused by moving outside of the grid to a space that doesn’t exist.

A screenshot of a computer program

Description automatically generated

To fix this I added this if statement   


A black background with white text

Description automatically generated

The next test I conducted was to see what happened when I entered invalid characters when prompted for an input. I received this error. A screen shot of a computer

Description automatically generated

The error can be caught with a try except statement. When the program detects a certain type of error when running the try catch statement it will activate the code under except and skip the code under try.

A computer screen with colorful text

Description automatically generated with medium confidence

However I think this solution is innefficient as the error could be caused for multiple different reasons that you might not have thought to include in the except statement. After some reasearch I discovered a nice way to catch errors with a dictionary.

A computer screen with text on it

Description automatically generated

The code here will only execute if the user input exists in the dictionary named “direction map”.

**# Conclusion**