Mystery Maze Game Documentation

1.0 - General Overview

Mystery maze is a console based game coded in the C++ coding language. The game presents the user with a randomly generated maze, created using the recursive backtracking algorithm (see section 2). They are tasked with following the path through the maze to the finish line, avoiding randomly moving enemies that can block their path (and cause penalties to be applied), whilst collecting resources necessary to complete the maze.

The object of the maze is for the player to reach the finish line in the smallest number of moves possible, whilst still having the correct amount of resources (tokens) necessary to complete the maze (5).

On each turn of the game, the user is shown the current state of the maze, followed by the number of remaining resources to collect.

The user is then asked to enter a direction of movement using the W,A,S,D keys:

- W - Move the current player position up
- A - Move the current player position left
- S - Move the current player position down
- D - Move the current player position right.

Each time the player inputs a directional statement, the game checks to see if that move is valid, i.e. that the move doesn't place the user in a wall, or out of the maze border and if the move is validated, the players game character (O) will shift one space in the chosen direction in the 2d array containing the maze state. If this is an open space in the maze, the player's character will move normally as expected, and the next turn will start with the new state of the maze being displayed.

If the chosen space contains a randomly placed puzzle, the relevant action will be displayed to the user, and they will be required to complete this action to proceed in progressing through the maze.

If the space contains an enemy on that particular turn, the game will reset the resources collected, and the next turn will begin.

If the current space contains a resource character (R), then the counter with remaining resources will lower, and the player will be allowed to continue the game as normal.

2.0 - Generation Of The Maze Using Recursive Backtracking

During the initial stages of development of this game, a number of maze generation algorithms were researched that could allow the programmer to be able to generate a unique maze each time the game is played. The following algorithms were suitable due to the constraints of this project:

- Recursive Backtracking
- Prim's Algorithm

	How the algorithm works	Pros of chosen algorithm	Cons of chosen algorithm
Recursive Backtracking	This algorithm works by branching out in a chosen direction until the algorithm finds a node with no other available nodes. It then backtracks back from this node until it reaches an available node where it can branch off again. This recursively occurs until a node is found where no connections can be made - this is the completed maze.	Long, tough paths to the solution Relatively simple recursive solution to implement	Slightly slower than Prim's algorithm
Prims Algorithm	This algorithm works similarly to the recursive backtracking algorithm, except instead of retracing to find the next available node, the algorithm selects the next branch from a list of randomly generated nodes.	Shorter paths through the maze, but ones that branch and require more thinking to solve. Faster algorithm to create the maze	Much harder implementation in C++, requiring more functions.

Following an in-depth review of the available algorithms to generate the maze, the recursive backtracking algorithm was employed to facilitate the generation of the unique mazes used in the game. This also allowed the project to be completed within the allocated time frame and to be completed using the minimal time complexity algorithms, reducing unnecessary processor usage.

3.0 - Enemies In The Maze Game

Enemies are added to the original state of the maze when the generate maze function is run at the start of each round of the maze.

2 enemies, denoted by the characters ‘B’ and ‘X’ are added to the game. If the user attempts to move onto one of these characters, the resources that the player has collected throughout the game are lost, and the maze resets the total available on the board back to 5 - these all need to be collected for the player to finish the game.

The enemies in this version of the maze game move randomly with each input from the player, adding an element of chance to the game, improving its playability and unpredictability.

The code checks, when the new positions are generated, if the position selected is the current position of a resource, puzzle or the player, and if this is the case, the program re-generates a new location for the movement to occur: this continues until the program selects a valid grid space, then the enemies are moved to this location.

4.0 - Resources And Puzzles

The mystery maze game contains resources that the player must collect to complete the maze.

There are 5 resources located throughout the maze, with some being in the path for the player to take to the finish, whilst others will require deviating off the path to collect, requiring a plan to use the least amount of moves to collect.

The maze finish line can only be unlocked once all 5 of the resources are collected from around the maze. If the player tries to finish the game without the necessary number of resources collected, the game will remind them that they need to collect all the resources, then return to the finish position to complete the game.

The puzzles in this game take the form of simple, randomly generated maths questions. These always follow the simple form below, allowing all users of the game to understand and be able to progress with the game.

- X + Y x Z

There is the option for a maximum of 2 random puzzles to be required to be solved each game. On the maze grid, these are denoted using the character ‘P’. When the player moves their character onto one of these squares, the game will generate a puzzle and ask the player the corresponding puzzle.

a. If they answer correctly, the game will continue as normal, and the puzzle will be cleared.

b. If the player answers incorrectly, the game will ask them again, and add 10 moves to the players total.

5.0 - Scoring System

Whilst the main objective of the mystery maze game is to complete the maze, gaining the necessary resources to reach the finish line, the player is scored on the number of moves required for them to reach the end successfully. This number is stored in a variable throughout the playing of the game, and once completed, is added to a file to a record of scores that the player has achieved. This can be cleared at any time using the relevant menu option. The file is written using the ifstream library, with each score being stored on a new line in the file.

From the menu, there is an option to show the current high score - this will be the lowest number of moves it took the player to reach the finish with the necessary materials. The program searches through the file, identifies the lowest score and outputs this to the user when the player chooses the ‘show high score’ menu option. In the case of my game, approximately 100-200 moves mark is considered to be a good score for the first level of the game.

6.0 - Timing System Used

During the playing of my maze game, the player is also measured on the time they take to complete the maze. This is outputted at the end of the game, and used to create the final score. This is accomplished by, when the play_game subroutine is run, the system takes a snapshot of the current time. This is repeated at the end of the game when the player finishes the maze - the time between the snapshots is taken and outputted to the user - this calculates the time the user has taken to complete the puzzle. When the score is outputted to the user, the time they have taken to complete the maze is used along with the number of moves undertaken to calculate the score (How?).

The game also has a user modifiable time limit of 4 minutes (total game playing time). This time limit is calculated by using 2 timepoints per each user term - one at the start and end. These individual time periods can then be totalled to identify the total time taken and this exceeds the agreed total duration, the game will be terminated. The user can change this total game playing time using the menu option, which allows the user to enter a time for the game in seconds.

7.0 - Generating a hard maze

When the player opens the game, or when they complete the first level, they are offered the option to play the standard level, which involves a 21x21 maze, or the harder level, which involves a 31x31 maze. The player can select this from either the main menu, or when prompted to after finishing a successful maze. If the player selects to do the more , the maze_hard_flag, defined at the start of the program, is set to true, specifying to the program that the user would like to do the harder maze, and telling it which functions to run: the ones specific to the hard maze (some functions are different depending on the level, such as the generate maze function). If the hard maze is selected, the game generates a unique 31x31 maze, and game play continues as normal.

8.0 - Saving and Loading the game from a file

When playing the maze game, the user is given the option to save the game to a file, and resume this game at a later state. This can be carried out either at the end of the game, by using the prompts the game displays, or can be done by, during any turn, entering the 'm' character into the game - this causes the user to be prompted to save the game. When saving the game, the user is asked to enter a filename; this filename is the one used to save the current state of the maze - this action is performed using the fstream library in C++. Alongside this file saving the current state of the maze game, the program also saves an auxiliary file - this file is named stats.txt, and stores the current position of the user on the board, the time remaining for that game, and the current number of resources collected - this allows the user to return to the game and not require to collect all 5 resources again when the game is resumed.

Loading a game from a file can be carried out either at the start of execution of the program, or when deciding to undergo another game following completion of the first game. This prompts the user to once again enter a valid filename - this would be the one that they specified when originally saving the game. Once the user inputs a valid filename (the game re-displays the menu each time they enter an invalid name), the game loads the specified maze from the file, and sets this to the current maze state. The user can then play this game by using the appropriate menu option to 'play loaded game'.

9.0 - Saving and Loading high scores

Once each game is completed by the user, a score is generated based on their movements and accuracy when playing (See section 5 - Scoring System). This score, at the end of each round of the game, is saved into a file called scores.txt. This file contains all the recently saved scores, and can be used to display the high score in the game if the user decides to do so. This is completed in the same way as the saving and loading of gameplay, and when called upon, the game displays the current high score by figuring out the lowest number in the file.

SUPPLEMENTARY VIDEO - https://youtu.be/OCspgv5kkEE