Table of Contents

[1.0 Introduction 1](#_Toc513069982)

[2.0 Game Design 2](#_Toc513069983)

[2.1 Design of Maze (Layout A) 2](#_Toc513069984)

[2.2 Constraints of Randomly Generated Maze (Layout R) 4](#_Toc513069985)

[2.3 Random Maze Illustration 6](#_Toc513069986)

[2.4 Gameplay 8](#_Toc513069987)

[3.0 Software Design 9](#_Toc513069988)

[3.1 Class Diagram 9](#_Toc513069989)

[3.2 Text File Input 11](#_Toc513069990)

[3.3 Source Code 15](#_Toc513069991)

[4.0 Software Testing 25](#_Toc513069992)

[4.1 Program Instructions 25](#_Toc513069993)

[4.2 Test Cases 27](#_Toc513069994)

[5.0 References 32](#_Toc513069995)

[6.0 Appendixes 33](#_Toc513069996)

[Appendix I - Print Error Message Function 33](#_Toc513069997)

[Appendix II - Print Fancy Message Function 33](#_Toc513069998)

[Appendix III - Quit Program Function 33](#_Toc513069999)

[Appendix IV - Is Valid Text File Input Function 33](#_Toc513070000)

[Appendix V - Get File Name Function 34](#_Toc513070001)

[Appendix VI - Get Current Room Label Function of Maze Object 34](#_Toc513070002)

[Appendix VII - Get Escape Room Label Function of Maze Object 34](#_Toc513070003)

[Appendix VIII - Get Room Label Object of Room Function 34](#_Toc513070004)

[Appendix IX - Print Direction Function of Exit Object 35](#_Toc513070005)

[Appendix X - Get Direction Function of Exit Object 35](#_Toc513070006)

[Appendix XI - Get Next Room Function of Exit Object 35](#_Toc513070007)

# 1.0 Introduction

This report will document the aspects of The Maze Game – a consoled-based C++ program. The aim of The Maze Game is to provide an enjoyable experience through a program that, by today’s standards, require little processing power to run. The objective of The Maze Game, besides having fun, is to improve player mentality by stimulating motivational drivers of human behaviour (Robson et al., 2015)

McGonigal (2011) states the four defining traits of a game: a goal, rules, a feedback system, and voluntary participation, and “everything else is an effort to reinforce and enhance these four core elements”. In other words, playing a game is a voluntary attempt to overcome necessary obstacles.

The Maze Game is played by placing the player in a starting room with the goal of escaping the maze by reaching the exit room. The program will print messages in the console and receive user input to interact with the player, forming a feedback system. Players are given a choice of two maze layouts – a fixed layout (layout A) and a randomly generated maze (layout R). The rules of the game restrict the player by only accepting a list of responses given by the program, including the option to quit at any time.

Upon completing the game, the program outputs a statistic of moves taken for the player to complete the game. The idea of displaying a score enables comparison between players, enabling friends to challenge each other. Furthermore, it provokes positive emotion that drives a player to try harder to achieve a better score in The Maze Game.

# 2.0 Game Design

## 2.1 Design of Maze (Layout A)

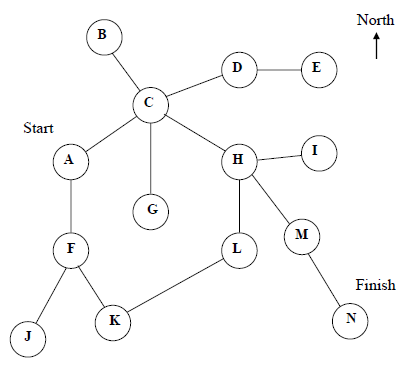


Figure 1: Blueprint of maze with layout A

1. Maze contains 14 rooms

Layout A contains 14 rooms labelled with letters A-N.

1. Start in room A; end in room N

When starting the game in layout A, the player will be placed in room A. The player has to complete the game by arriving in room N.

1. Exits of rooms are limited to ‘N’, ‘E’, ‘S’, ‘W’ and ‘T’

‘N’ represents north, ‘E’ represents east, ‘S’ represents south, ‘W’ represents west and ‘T’ represents trapdoor. Exiting through trapdoor is unique to room C because it has 5 exits.

1. Two-way connection between rooms

A path that connects two rooms is accessible from either room, enabling to and fro movements.

1. Exit direction in a pathway that connects two rooms are mirrored

Moving in an opposite direction immediately will return to the previous room. For example, in room C, travelling south will move to room G, while travelling north from room G will move to room C.

## 2.2 Constraints of Randomly Generated Maze (Layout R)

1. Maze contains 12 rooms

Number of rooms in a randomly generated maze is fixed at 12, using letters between O-Z to label the 12 rooms.

1. The first room of a random maze is room R

When playing in a randomly generated maze, the player will always begin in room R.

1. The exit room of a random maze is room O

The win condition of a randomly generated maze is fixed whereby players have to travel to room O.

1. ‘N’, ‘E’, ‘S’, ‘W’ is used as exit directions in room

North, south, east, west, in no fixed order, will represent possible exit directions. Directions are unique in each room, therefore there will not be instances of two similar exit directions in a room.

1. A connection must exist between first room R and exit room O

The random layout is initialized by connecting room O to room P, then connecting room P to room Q, and finally connecting room Q to room R, forming an indirect route from room R to room O. Rooms P, Q, R are characterized as ‘vital rooms’ because a route to exit room O exists in all of them.

1. A ‘vital exit’ is placed within each room

‘Vital exit’ of rooms P, Q, and R, are the exit that forms a connection to exit room O. For the other rooms, a ‘vital exit’ is generated but it must connect to a ‘vital room’. Other generated exits will lead to rooms in range of S – Z. ‘Vital exits’ are required to guarantee a route to exit room O, eliminating the prospect of an endless maze.

1. Each room will have between 2-4 exits

Each room will have a minimum of 2 exits and a maximum of 4 exits, including a ‘vital exit’. It is not possible to exit from one room but end up in the same room, for example, an exit in room T must not lead back to room T. Furthermore, the exits are unique, meaning no two exits will lead to the same room.

1. One-way connection between rooms

Unless it is coincidentally generated, there will be no immediate return path after exiting a room. For example, in room X, there is an exit that leads to room Y, but there might not be an exit in room Y that returns to room X.

1. A new random layout is processed upon each selection of random layout

Each time the random layout is selected, the program will generate a new random layout. It is not possible to return to a previous layout that has been generated.

## 2.3 Random Maze Illustration

The following images illustrate the formation of a random maze:

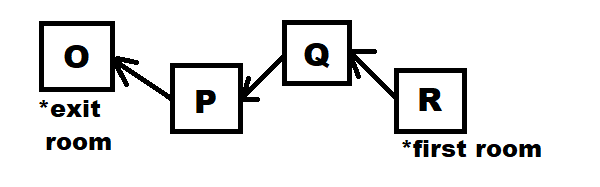


Figure 2: Formation of OPQR link.

One-way connections are formed to join room R to room Q, room Q to room P, and room P to room O.

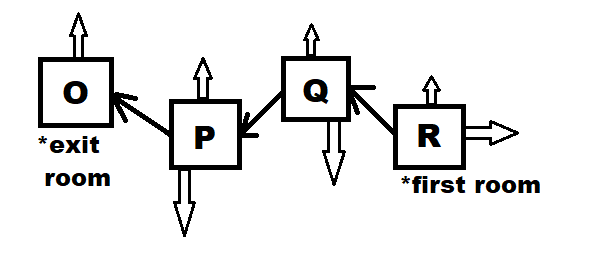


Figure 3: Adding additional exits to rooms O-R

Each room is given a random additional (1, 2 or 3) number of one-way exits.

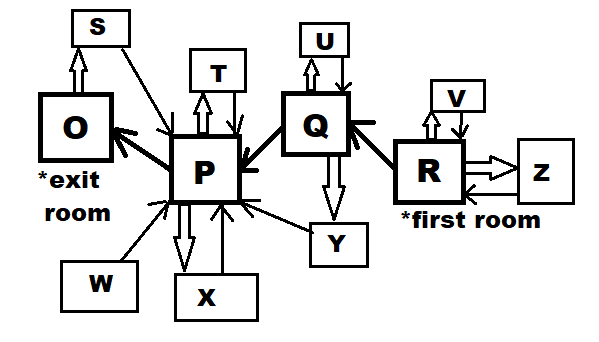


Figure 4: Generating rooms S-Z.

Rooms S-Z are added and they must contain an exit to ‘vital rooms’ P, Q, and R, creating a ‘vital exit’

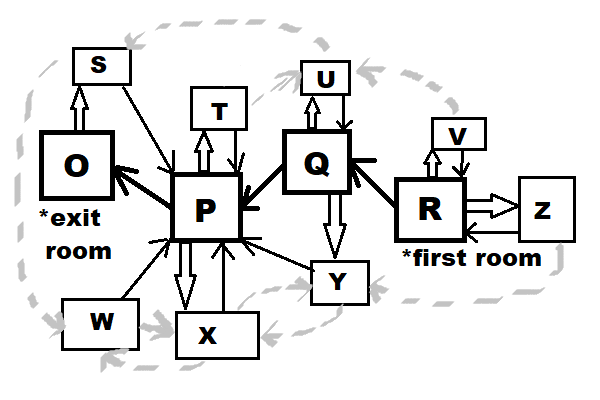


Figure 5:Room S-Z random exits

Exits are generated randomly between rooms S-Z to create 2-4 exits for each room.

## 2.4 Gameplay

At the start of the game, the player is given the option of choosing layout A or layout R to play in. When a layout is chosen, the player is placed in the first room of the layout and is informed of the label of current room and its available exit directions. However, the player must manoeuvre towards out of a room to figure out the next room that the direction leads to. The process showing room label and available exits is repeated until the player arrives at the exit room of the selected layout.

The player is given choices at every stage of the game and it does not progress until it receives a valid player input. The player must respond by typing the suggested characters prompted by the game and then press the enter key to send input. At any stage of the game, the player is able to quit the game by entering ‘Q’, and the player is constantly informed of the option. Completing the game or choosing to quit will halt the program to show an appropriate message before the game is terminated.

# 3.0 Software Design

## 3.1 Class Diagram

An object-oriented approach is adopted to develop The Maze Game. In an object oriented approach, the programmer identifies classes that fit the requirement then creates objects solve the problem. (Davis, 2014). A class diagram is chosen to illustrate the objects within The Maze Game, because it is a diagram that shows different classes and their relationship to each other (Mueller & Cogswell, 2014).

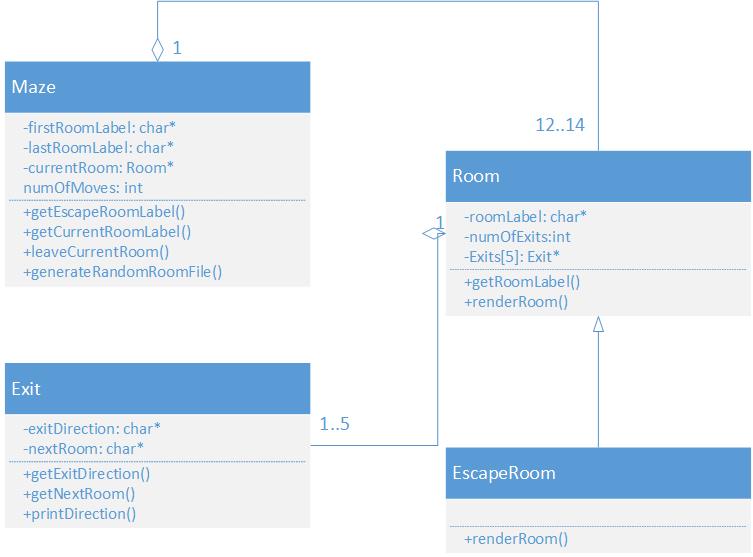


Figure 6: Class diagram of The Maze Game.

The Maze object has four private attributes and four public functions. ‘firstRoomLabel’ defines the current layout of the maze, whether it is layout A or layout R, while ‘lastRoomLabel’ is used to check if the player has completed the objective of reaching the exit room. ‘currentRoom’ attribute is of data type Room object and is created each time the player travels to across rooms. ‘numOfMoves’ is used to store the statistic of moves taken to escape the maze. ‘getEscapeRoomLabel()’ and ‘getCurrentRoomLabel()’ are getter functions that return ‘lastRoomLabel’ and ‘firstRoomLabel’ respectively. ‘leaveCurrentRoom()’ is called by the program to accept user input and move the player according to the input received. ‘generateRandomRoomFile()’ is called when layout R is chosen; it generates random room files according to constraints and outputs those file into ‘gamedata’ folder.

The Room object has three private attributes and two public functions. ‘roomLabel’, like the name suggests, is used to identify the current room. ‘numOfExits’ is used to count valid lines of data from text input to calculate the number of exits of a particular room. ‘Exits’ is an array that will store data type of Exit object. ‘Exits’ array size is set to 5 to reflect the maximum number of exits of a room. ‘getRoomLabel()’ is a getter function that returns ‘roomLabel’. ‘renderRoom()’ is a virtual function that allows the player to enter exit direction that will loop recursively if an invalid response is received.

EscapeRoom object is an inheritance of Room object that overwrites the ‘renderRoom()’ function to print an appropriate message when the player has completed the game. EscapeRoom object created by Maze Object after ‘leaveCurrentRoom()’ returns a character value equal to ‘lastRoomLabel’.

The Room object is used to store data from text input of format (see *3.2 Text File Input)*. Public functions ‘getExitDirection()’ and ‘getNextRoom()’ respectively return private values of ‘exitDirection’ and ‘nextRoom’ that is used to store the direction of the exit and the room that it leads to. ‘printDirection()’ displays a formatted output of both ‘exitDirection’ and ‘nextRoom’ that is a vital gameplay component because it determines the user input that is accepted by ‘renderRoom()’ function.

Most of the program uses dynamic memory allocation by practising the use of object pointers and variable pointers.

## 3.2 Text File Input

The program builds a maze by reading formatted data from a text file for every room. Rooms of layout A will have its text file pre-defined by the developers then stored in ‘gamedata’ folder. Data is stored in the format of α=β, where α denotes the direction of exit and β represents the next room that the exit leads to. The following series of images show pre-defined values of text files of room A to room N. Notice that in *figure 6*, extra line spacing, spacing between characters and case insensitive data is accepted

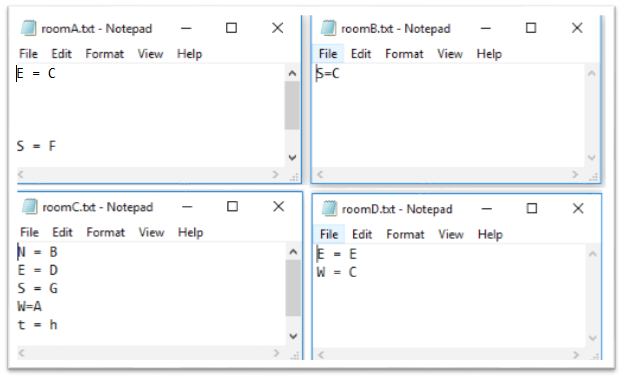


Figure 7: Text files of room A, room B, room C, room D.

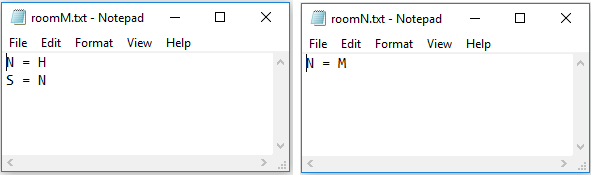


Figure 8: Text files of room M, room N.

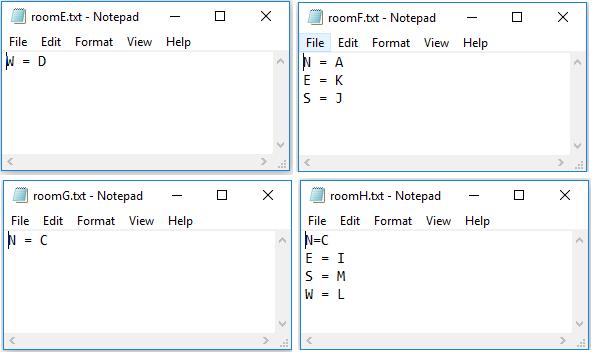


Figure 9: Text files of room E, room F, room G, room H

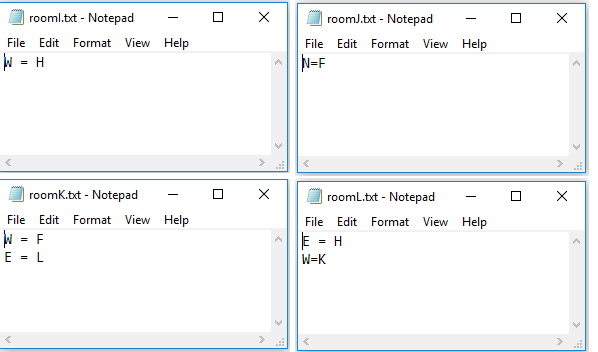


Figure 10: Generated text files of room I, room J, room K, room L.

On the other hand, text files or layout R are created when running the program. The following text files are a sample of a randomly generated output for layout R created on 1 May 2018. Like the text files of layout A, they are stored in ‘gamedata’ folder in the α=β format.

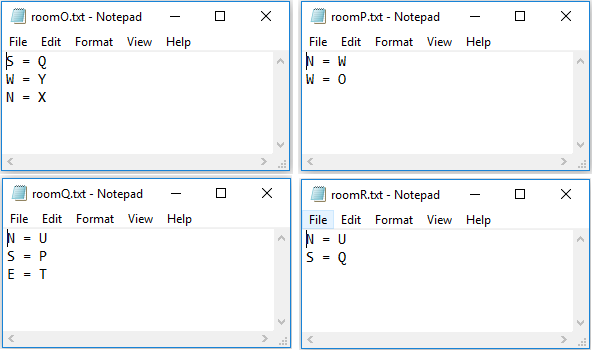


Figure 11: Generated text files of room O, room P, room Q, room R.

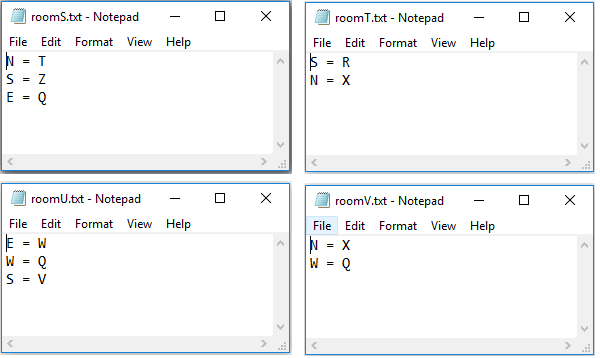


Figure 12 Text files of room S, room T, room U, room V.

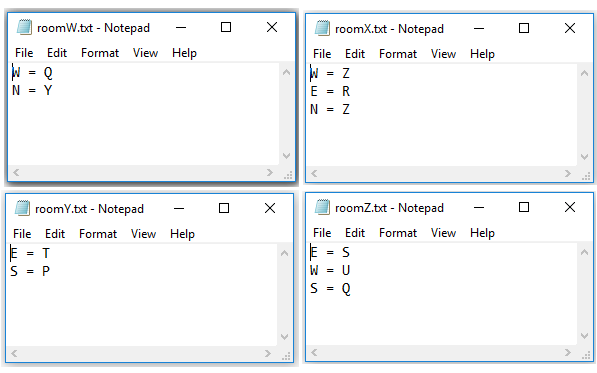


Figure 13: Generated text files of room W, room X, room Y, room Z.

In *figure 10,* ‘vital exit’ requirement of ‘vital rooms’ is created where a connection of R-Q-P-O is formed. In *figure 11* and *figure 12*, ‘vital exits’ exists in the form of an exit to room P, room Q, or room R. All generated rooms for layout R have 2, 3, or 4 exits.

## 3.3 Source Code

a) Libraries and Main Function

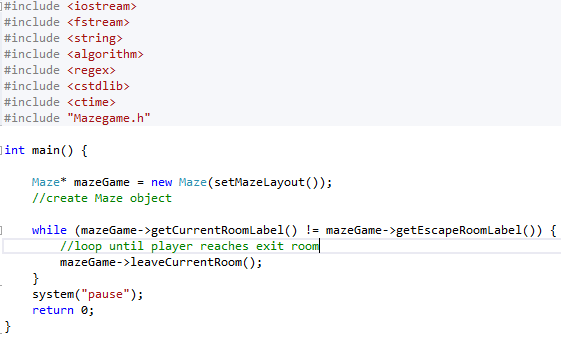


Figure 14: Main program.

An out-of-the-line approach is implemented with a header file, function definitions file and main program. The main program is shown above in *figure 14.*

b) Mazegame.h Part 1

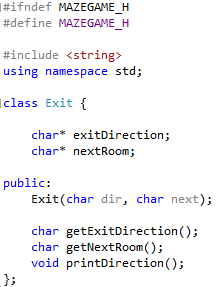


Figure 15: Exit object declaration in header file

c) Mazegame.h Part 2

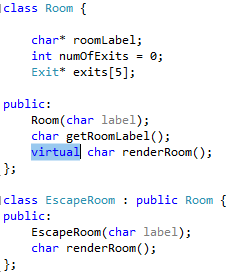


Figure 16: Room object and EscapeRoom object declaration in header file

d) Mazegame.h Part 3



Figure 17: Maze object declaration in header file

e) Mazegame.h Part 4

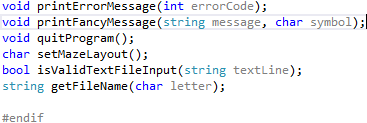


Figure 18: Additional functions to supplement class functions.

Definitions of these functions can be found in *6.0 Appendixes.* The #endif in *figure 18* is paired with #ifndef in *figure 15* that is used to avoid repeated declaration of classes and functions when the header file is included in multiple files.

f) setMazeLayout Function

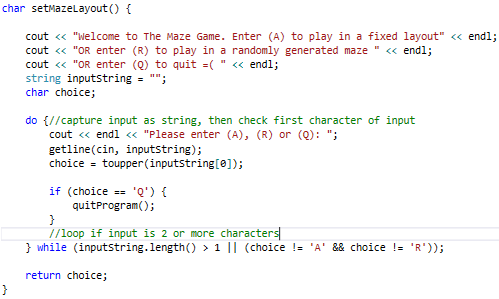


Figure 19: Function definition of setMazeLayout.

Designed to receive user input or repeat itself until a valid input is received, the first function called by the program includes a welcome message while conveying instructions to the player. User input will be used to create maze of the chosen layout, unless the player chooses to quit, then the ‘quitProgram()’ function is called (*Appendix III*).

g) Maze Object Constructor

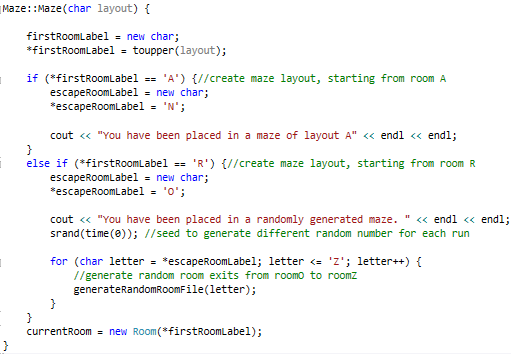


Figure 20: Constructor of Maze object.

Parameter ‘layout’ to create Maze object is returned from setMazeLayout function. Parameter is then stored as ‘firstRoomLabel’ as a corresponding ‘escapeRoomLabel’ is assigned. If layout R is chosen, then the generateRandomRoomFile functionis executed 12 times through a for loop to generate the random layout. Finally, a new room object is dynamically created with a parameter of ‘firstRoomLabel’ and is assigned to ‘currentRoom’.

h) Room Object Constructor

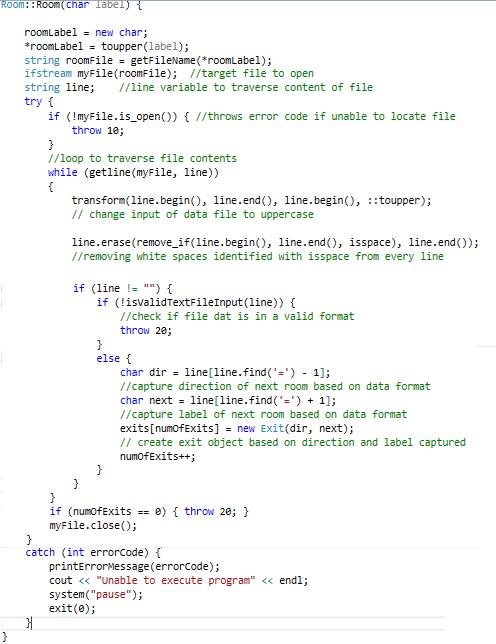


Figure 21:Constructor of Room object.

Program uses ifstream to open file by transforming ‘roomLabel’ to a text file location string with getFileName function (*Appendix V).* If unable to locate file, an error is thrown then the program prints an appropriate error message with printErrorMessage function (*Appendix I*) and gracefully terminates itself. Text data is captured and a regular expression is used to check for data validity (*Appendix IV*), but prior to that, whitespaces are removed and empty strings are ignored. If data is absent or invalid, another error is thrown and program terminates. For each line of valid data in text file, an Exit object is created and ‘numOfExits’ is increased by 1.

i) Exit Object Constructor

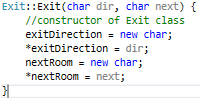


Figure 22: Constructor of Exit Object.

Dynamic memory allocation is used to store variables of the Exit object.

j) leaveCurrentRoom Function

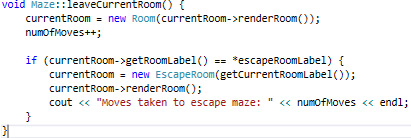


Figure 23: Function definition of leaveCurrentRoom.

A new dynamic Room object is created through ‘renderRoom()’ virtual function. Upon completion of ‘renderRoom()’ the value of ‘numOfMoves’ is increased by 1.This function will be repeatedly executed by main function until ‘getRoomLabel’ (*Appendix VIII)* is equal to ‘escapeRoomLabel’. Then, a new EscapeRoom object is created with parameter returned from ‘getCurrentRoomLabel()’ (*Appendix VI)*, then the renderRoom() function, overwritten through inheritance, is called. Finally, the statistic of number of moves taken to complete the game is shown.

k) renderRoom Function of EscapeRoom Object

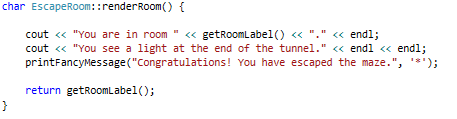


Figure 24: Function defintion of renderRoom in EscapeRoom object..

Program output when player has complete the objective. Definition of ‘printFancyMessage()’ is available in *Appendix II*.

l) renderRoom Function of Room Object

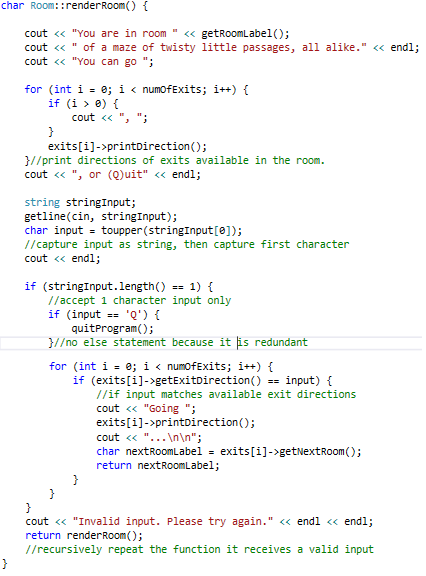


Figure 25: Virtual function definition of renderRoom of Room object.

‘renderRoom()’ is a virtual function to capture user input that will recursively repeat itself until a valid input is received. Function definitions of ‘getRoomLabel()’, ‘printDirection()’, ‘getExitDirection()’ and ‘getNextRoom()’ is shown in *Appendixes VIII, IX, X, XI* respectively. The program outputs the current room label and available exit directions and it expects player input that matches available directions. If the player chooses to quit, then the program will call ‘quitProgram()’ (*Appendix III*), otherwise it will check for a valid exit direction then returns ‘nextRoomLabel’ of the given direction that will be used by Maze object to update its 'currentRoom’ object.

m) generateRandomRoomFile Function

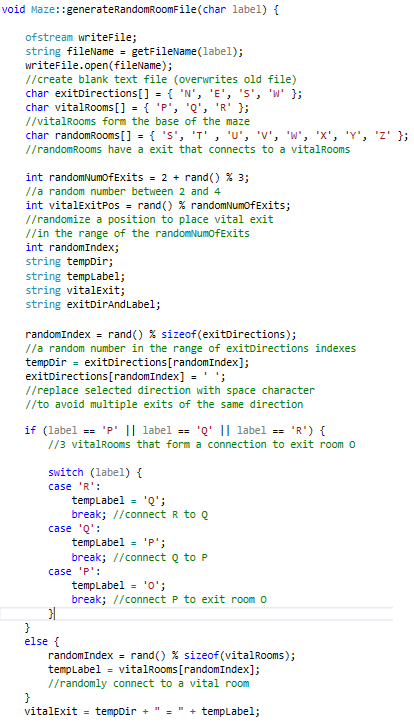


Figure 26: Function definition of generateRandomRoomFile, part 1.

Using ofstream to write output in conjunction with ‘getFileName()’ (*Appendix V*), the program writes generated exits for each room into a different text file. Firstly, a ‘vital room’ will have its fixed connection paired with a random direction, while any member of the ‘randomRooms[]’ array will have an exit direction paired with a randomly selected index of ‘vitalRooms[]’. Secondly, a randomly index of ‘exitDirections[]’ and another random index of ‘randomRooms[]’ will generate an exit. Any object selected from any array will be replaced with a space character to avoid repetition. Thirdly, the program selects a random number of 2, 3 or 4 as the number of exits and another random number of ‘vitalExitPos’ within the range of the random number of exits.

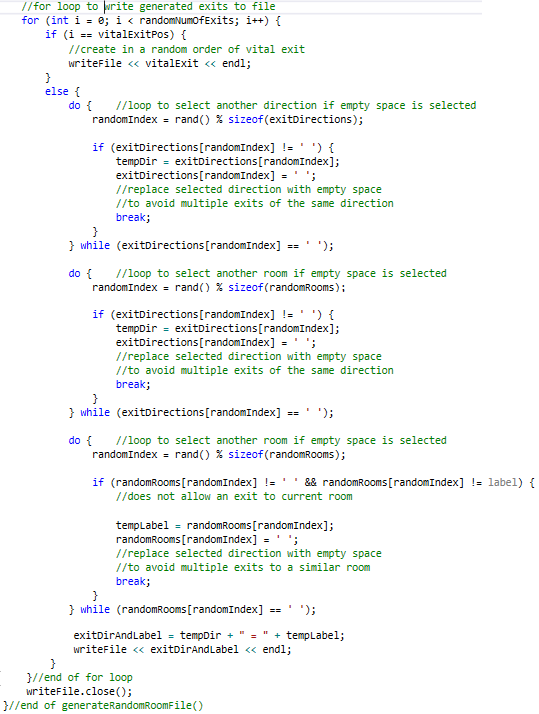


Figure 27: Function definition of generateRandomRoomFile, part 2

If the random number generator chooses an index of a space character, meaning the direction or room label has already been chosen, another random index will be chosen. ‘Vital exit’ is randomly positioned within the text file because it is written when the index of for loop matches ‘vitalExitPos’.

# 4.0 Software Testing

## 4.1 Program Instructions

1. Check if texts files of rooms A-N is placed in ‘gamedata’ folder and if the data within the text files are in α=β format.

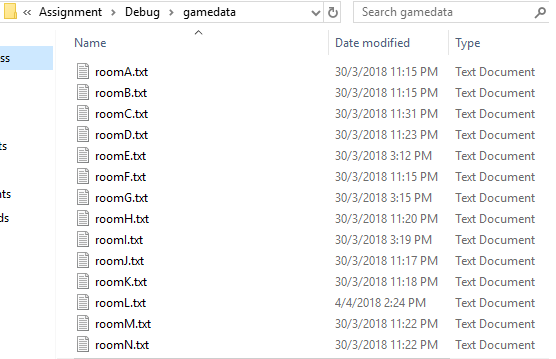


Figure 28: Contents of 'gamedata' folder.

1. Run assignment.exe in Visual Studio project folder/debug

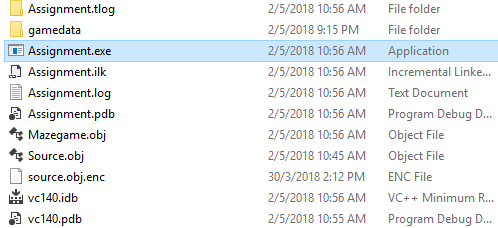


Figure 29: Contents of debug folder.

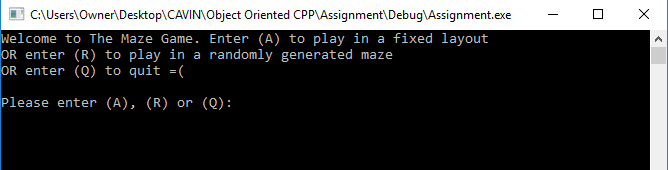


Figure 30: Main menu of The Maze Game.

1. Key in valid responses according to program output to navigate through the game.

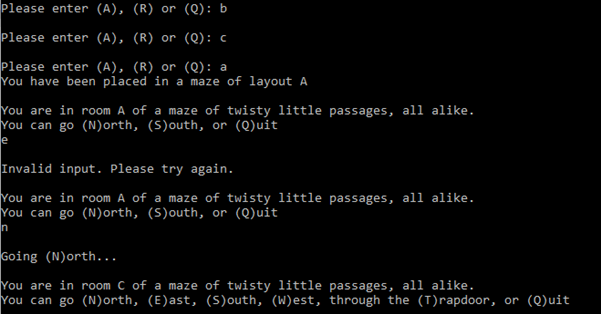


Figure 31: Playing The Maze Game.

1. Repeat until game is completed or player chooses to quit.

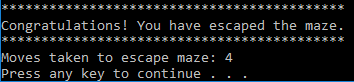


Figure 32: Completing the game.

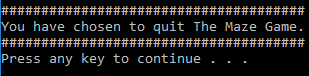


Figure 33: Quitting the game.

## 4.2 Test Cases

**Test Case: Text Files**

Table 1: Test script of text files

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Condition** | **Test Result** | **Comments** |
| A1 | roomA.txt is removed | fail | Program halts at room A and prints error message |
| A2 | roomB.txt is renamed to B.txt | fail | Program halts at room B and prints error message |
| A3 | roomC.txt is changed to roomC.dat | fail | Program halts at room C and prints error message |
| A4 | roomD.txt is unchanged. | PASS | Program execution is successful |
| A5 | roomE.txt is unchanged. | PASS | Program execution is successful |
| A6 | roomF.txt is unchanged. | PASS | Program execution is successful |
| A7 | roomG.txt is unchanged. | PASS | Program execution is successful |
| A8 | roomH.txt is unchanged. | PASS | Program execution is successful |
| A9 | roomI.txt is unchanged. | PASS | Program execution is successful |
| A10 | roomJ.txt is unchanged. | PASS | Program execution is successful |
| A11 | roomK.txt is unchanged. | PASS | Program execution is successful |
| A12 | roomL.txt is unchanged. | PASS | Program execution is successful |
| A13 | roomM.txt is unchanged. | PASS | Program execution is successful |
| A14 | roomN.txt is unchanged. | PASS | Program execution is successful |

The above test script has been performed on 1 May 2018. It is shown that the text files should not be altered for the program to work properly.

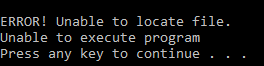


Figure 34: Error message received in tests A1, A2, and A3.

**Test Case: Text File Data**

Table 2: Test script of text file data

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Condition** | **Test Result** | **Comments** |
| B1 | roomA.txt contains many empty lines between data ‘N = C’ and ‘S = F’ | PASS | Program execution is successful |
| B2 | roomB.txt contains ‘===’ | fail | Program halts at room B and prints error message |
| B3 | roomC.txt contains ‘W=A’ | PASS | Program execution is successful |
| B4 | roomC.txt contains ‘t = h’ | PASS | Program execution is successful |
| B5 | roomF.txt contains ‘f = f’ | fail | Program halts at room F and prints error message |
| B6 | roomJ.txt contains ‘N = !’ | fail | Program halts at room J and prints error message |
| B7 | roomL.txt contains ‘W=K’ | PASS | Program execution is successful |
| B8 | roomM.txt is empty | fail | Program halts at room M and prints error message |

The above test script has been performed on 1 May 2018. It is shown that the data within the text files should match the regular expression in *Appendix IV*.



Figure 35: Error message received in tests B2, B5, B6, and B8.

**Test Case: User Input on Main Menu**

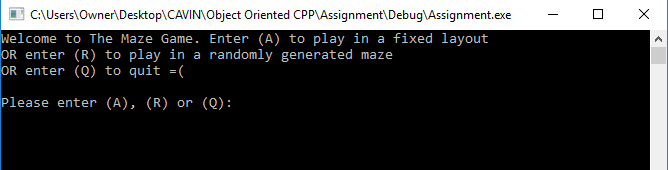


Figure 36: Main menu.

Table 3: Test script of main menu.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Condition** | **Test Result** | **Comments** |
| C1 | User input ‘A’ | PASS | Input is accepted |
| C2 | User input ‘a’ | PASS | Input is accepted |
| C3 | User input ‘aaa’ | fail | Input is rejected |
| C4 | User input ‘r’ | PASS | Input is accepted |
| C5 | User input ‘R’ | PASS | Input is accepted |
| C6 | User input ‘RR’ | fail | Input is rejected |
| C7 | User input ‘q’ | PASS | Input is accepted |
| C8 | User input ‘Q’ | PASS | Input is accepted |
| C9 | User input ‘!QoWO’ | fail | Input is rejected |
| C10 | User input ‘123’ | fail | Input is rejected |
| C11 | User input ‘n’ | fail | Input is rejected |
| C12 | User input ‘E’ | fail | Input is rejected |
| C13 | User input ‘S’ | fail | Input is rejected |
| C14 | User input ‘w’ | fail | Input is rejected |
| C15 | User input ‘t’ | fail | Input is rejected |

The test script above has been performed on 1 May 2018 to prove that only valid input is accepted in the main menu.

**Test Case: User Input in Rooms**

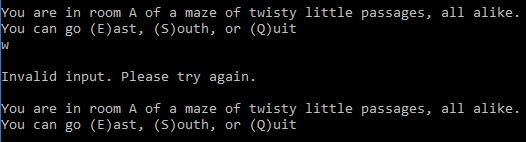


Figure 37: Room A after an invalid input.

Table 4: Test script of room A.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Condition** | **Test Result** | **Comments** |
| D1 | User input ‘E’ | PASS | Input is accepted |
| D2 | User input ‘e’ | PASS | Input is accepted |
| D3 | User input ‘e!3’ | fail | Input is rejected |
| D4 | User input ‘s’ | PASS | Input is accepted |
| D5 | User input ‘S’ | PASS | Input is accepted |
| D6 | User input ‘SS’ | fail | Input is rejected |
| D7 | User input ‘q’ | PASS | Input is accepted |
| D8 | User input ‘Q’ | PASS | Input is accepted |
| D9 | User input ‘!QoWO’ | fail | Input is rejected |
| D10 | User input ‘123’ | fail | Input is rejected |
| D11 | User input ‘E’ | fail | Input is rejected |
| D12 | User input ‘w’ | fail | Input is rejected |
| D13 | User input ‘t’ | fail | Input is rejected |

The test script above has been performed on 1 May 2018 to prove that only valid input is accepted in rooms.

**Test Case: Random Maze Text File Data Output**

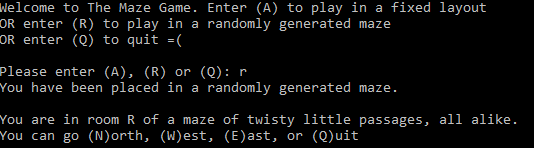


Figure 38: Choosing layout R..

Table 5: Test script of random maze generation..

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Test Condition** | **Test Result** | **Comments** |
| E1 | Layout R is selected | PASS | Text files of rooms O-Z is created in ‘gamedata’ folder |
| E2 | Layout R is selected | PASS | Text file data of rooms O-Z follow the α=β format |
| E3 | Layout R is selected | PASS | Generated text files contain 2-4 exits. |
| E4 | Layout R is selected | PASS | At least 1 exit is created that leads to room P, room Q, or room R for rooms S-Z |
| E5 | Layout R is selected | PASS | Room R is connected to Room Q, with additional random exits. |
| E6 | Layout R is selected | PASS | Room Q is connected to Room P, with additional random exits. |
| E7 | Layout R is selected | PASS | Room Q is connected to Room O, with additional random exits. |
| E8 | Layout R is selected | PASS | There are no repeated directions or next rooms within any text file |

The test script above has been performed on 1 May 2018 and is repeated 5 times to prove that the random maze generation consistently performs as intended.

# 5.0 References

Davis, S.R., 2014, *Beginning Programming with C++ For Dummies*, Wiley, Somerset. [online] Available at: ProQuest Ebook Central. [Accessed 1 May 2018].

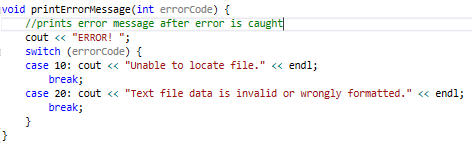
McGonigal, J., 2011. Reality is Broken: Why Games Make Us Better and How They Can Change the World. *New York*, [online] p.402. Available at: <http://vk.com/doc195058859\_332475945?hash=c90a762ba7b55ce05b&dl=696221d531cca93dff>.

Mueller, JP, & Cogswell, J 2014, *C++ All-in-One For Dummies*, Wiley, Somerset. [online] Available at: ProQuest Ebook Central. [Accessed 1 May 2018].

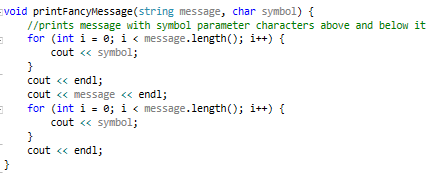
Robson, K., Plangger, K., Kietzmann, J.H., McCarthy, I. and Pitt, L., 2015. Is it all a game? Understanding the principles of gamification. *Business Horizons*, [online] 58(4), pp.411–420. Available at: <http://dx.doi.org/10.1016/j.bushor.2015.03.006>.

# 6.0 Appendixes

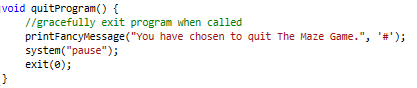
## Appendix I - Print Error Message Function



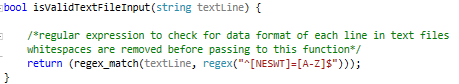
## Appendix II - Print Fancy Message Function



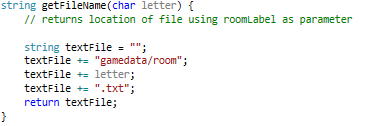
## Appendix III - Quit Program Function



## Appendix IV - Is Valid Text File Input Function



## Appendix V - Get File Name Function



## Appendix VI - Get Current Room Label Function of Maze Object



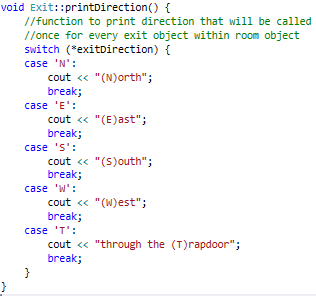
## Appendix VII - Get Escape Room Label Function of Maze Object



## Appendix VIII - Get Room Label Object of Room Function



## Appendix IX - Print Direction Function of Exit Object



## Appendix X - Get Direction Function of Exit Object



## Appendix XI - Get Next Room Function of Exit Object

