Analysis

Project Background

In the last year, I have been getting into game development and I have been invested in advanced computing and I love the idea of design software: Maze Generator Game. This involves creating a searching algorithm to create a path from a set start point to another set finish point, then creating a pathfinding algorithm to create the maze and use a multi-dimensional array to create and limit the background around the random path created so the game always has a completed path from start to finish, then create an SQL database to save the players information depending on the clients needs.

Below is a class diagram for the current system of the software.

Project Outline

As I stated before, the maze generator will randomly create a maze but will not change complexity in terms of levels or harder difficulty. Instead, it will have a guard that will have a selected range and will move randomly around the maze, when the guard sees the player, it will chase him up to the end of his range.

By completing the maze, it will start the maze generator on level 2 and so on, it will increase the number of guards, the range of the guards and their speed so it will get harder every time.

My client for this project is my brother who likes to play role-playing games and would like an original maze game to play as a hobby with his friends.

I intend to make the rules and strategy part of the game easy to understand as my target audience are young teenagers and the complexity of the game will be based in the increase of levels and not the difficulty of playing (functions on various keys or hard moving patterns)

Stakeholder

As noted previously, my stakeholder is my brother (Sergio Estrada) and his friends who will be the main users of the project which are in year 7. My brother has agreed to answer questions through email and to provide me with advice about the project and objectives that he would like to be implemented within the project.

Email from my brother; Client

There are specific comments related to the complication of the game due to many functions, so I am going to

use the basic keys to play the game (up, down, left, and down arrows)

He also mentioned bonuses at each level, I will explain more about these in my research stage but for now I have implemented the idea in my class diagram above.

Objectives

Must:

Have a randomizer algorithm to create the map/maze (No game should be the same map).

Have a Menu Scene with access to all the levels.

Pause Menu accessible whenever the player wants (In Game) through pressing keys in the keyboard.

Pause Menu should stop the game and have labels, buttons.

Door, portal to advance to next level, all levels to be connected.

2D interpretation.

Visual Image of maze, player and the door.

Enemy creation that chases player.

Active Timer that stops when Pause Menu is active.

Score Count directly proportional to the timer.

Path from start (starting point of player) to finish (door/exit/portal).

Be a tractable problem (solvable in reasonable amount of time).

Player Death when colliding with Enemy.

Save highest score and keep it in game so the player can try to beat it.

Should:

Have a maze renderer (software process that generates a visual image from a model).

Have a stop button; to be able to pause the game, a stop button in the screen or by pressing a desired key in the keyboard.

Advance into harder level of complexity every time player goes through a portal/door.

Be the same wall dimensions per scene.

Be the same floor dimensions per scene.

Have the timer to two significant figures.

Have a score count and a variable high score count which changes when the score count increases higher than the current high score.

Could:

Have a resetting function; it should have a button or function in the keys to reset the game.

Positive bonuses at each level.

3D game version of the 2D algorithm.

Have a stats information page.

Definite speed throughout the level and as you advance, increase in speed.

Range of enemy, minimal (on collision with player).

Increase the number of enemies depending on the level.

Limitations agreed with the stakeholder.

Player functions when colliding with enemies:

No need for extra lives; if time allows, I will try to manage it.

The player can respawn at the start.

The timer keeps going until the player reaches the exit.

Guard/Enemy movement and actions:

No need for AI or complex algorithms; if I can, I will try to accommodate AI.

Create a range for the guard to move around at random.

The guard will chase the player if the player is within line of vision.

Stats and player information:

The game will not save all the players stats, since it would require a lot of memory of the CPU.

It will save the top 5 stats of the players.

The stats will save the players highest score and the lowest time for each. (The higher their time, the higher their score)

Research

Old/Current Systems

Ancient Greek myth Theseus and the Minotaur

Story of the half-man, half bull killed by Theseus inside a labyrinth. Due to the Minotaurs huge size, King Minos ordered a craftsman called Daedalus and his son, Icarus, to build a huge maze known as the Labyrinth to entrap the beast. I used the idea of the Labyrinth where there is one path to enter, and that same path leads you out. You make the choice to enter the path and start a journey. I took inspiration from the Greek name of the hero and found it fitting to name my project after the hero of the myth: Theseus.

Unlike the myth, my project design of the maze will alter itself every time the game is started to avoid a monotonous resemblance of the last time played and instead of the hero (player) killing the minotaur (enemy), he will try to escape through the exit door to win.

The design of the maze was as shown below; A nearly straight path towards the centre of the maze, where the minotaur would live, and the entrance where King Minos would put human offerings and lock them inside the maze for the minotaur to eat.

Maze Runner Book/Film

A complex labyrinth with towering walls which move every night, making it impossible to find a way out. An area filled with footways that are separated by walls, hedges, corn plants, or other barriers. The aim of the story was to escape the maze, either by learning the patterns of change or surviving till you get to the exit without getting killed by the enemy.

Since my maze is randomly generated using the backtracking algorithm, it would be very difficult to learn the patterns of change so to make it similar to the film, I made it so that some of the levels will have a change in the camera perspective, basically meaning that, at times, depending on where the player is at, it wont be possible to see the whole maze so that the player cannot just memorize the path from beginning to end and follow it through.

Pac-man game

The algorithm used in the game of Pacman, allows ghosts to only run alongside the corridors of the maze, and to only move to the positions of x and y coordinates. Since my game is in 3D, I also have to take into account the z component of movement for the ghosts enemies, but I am projecting the same idea of the ghost only running alongside the corridors of the maze so I changed the camera positioning and the angle of projection to look at the game from above so it gives it the impression that the player and the enemy are

moving in 2D in the x and y axes but in reality, they are moving in the x and z axis while the y axes is a constant of 0.

Below is part of the mathematics of the algorithm to make the ghost follow Pacman:

Path Finding Algorithms

I have been researching different path finding algorithms and trying to decide which one to use, here are the algorithms I have chosen:

Recursive backtracker Algorithm

It is used to solve a variety of problems. It is a randomized version of depth-first search algorithm, and it uses a stack. A depth- first search works by starting at the root node (the top) of a tree and goes down as far as it is possible down a given branch, then it backtracks until it finds an unexplored path, then it goes to explore it and the last step repeats itself.

A basic example of a depth-first search is:

The root node in this case the node 1, and then it would go to two and after it would go to four.

Once it has explored a full path, it will backtrack the already visited paths (2) and will go to the next unvisited node (5).

This are the steps on how the algorithm works,

You choose an initial cell, and the algorithm checks to see if it can visit its neighbour, it picks a random neighbour and adds the neighbour on the stack, then it marks the path to get to the neighbour, it marks it as visited and POP (removes the last cell added to the stack).

Prims Algorithm

Prims algorithm creates a tree by getting the adjacent cells and finding the best one to travel to next. For this to work in terms of a maze, I will instead take a random cell to travel to the next one. The Prims Algorithm will keep a list of all adjacent cells. An example of Prims Algorithm is:

The algorithm will choose a starting cell at random and mark it as part of the maze, then it will add all adjacent edges to a pending list.

Then the algorithm, it will pick a random edge from the pending list, every edge connects two nodes, if only one of the two is part of the maze, then mark the edge and node as part of the maze.

Lastly, it will add all the new neighbouring edges to the pending list.

Binary Tree Algorithm

It is an exact memory-less maze generation algorithm with no limit to the size of the maze since it builds the entire maze by looking at each cell independently. For each cell in the grid, I would have to randomly decide whether to carve a passage north or west. An example of a Binary tree Algorithm is:

Decision taken after Research.

I have decided that I am going to use the recursive backtracker algorithm, each time a path is tested, if a solution is not found, the algorithm backtracks to test another possible path and so on till a solution is found or all paths have been tested. The algorithm works exactly like the player, it is also trying to find its way out of the maze by trial and error. The problem is solved one step at a time.

I made this choice because its the most efficient algorithm in terms of time taken to create the maze, the binary tree algorithm has a downside which is that two of the four sides of the maze will be spawned by a single corridor since for each cell it randomly decides whether to carve a passage north or west, two passages will be closed.

Prims algorithm also has a downside which is that the algorithm does not allow the me much control over the chosen edges when multiple edges have the same weight. In other words, at every iteration, the algorithm tries to readjust the input to its own convenience.

Database information

SQL is a domain-specific language used for managing data held in a database management system. The language is used to store, create, update, and retrieve data in a relational database. The main use of SQL is that it facilitates retrieving specific information from databases that are further used for analysis. A database is a table that consists of rows and columns that stores organized collection of information or data.

Three examples are:

Supermarket tracking stock in a store.

Contacts list in a mobile phone.

A library list of books and its information.

Compatibility of Users

Compatible with laptops, computers, iPad, and tablets. Not compatible with mobile phones. The minimum and basic system requirements to run Unity 2019.1 version are:

For an operating system, you would need a Windows 7/ macOS 10.12/ Linux; Ubuntu 16.04.

CPU would need a SSE2 instruction set support which is an extension of the IA-32 architecture, based on the x86 instruction set. Therefore, only x86 processors can include SSE2.

The minimum requirement for a GPU is a graphics card with DX10 and shader model 4.0 capabilities. And for apple devices, it needs to be an IOS of 9.0 or higher.

The minimum RAM required to be able to run Unity smoothly without latency is 8GB but 16GB is recommended since you would expect the computer to need more RAM to play the game.

Data Volumes

The system is written in C# using Unity. The hard diskspace that it would take into your computer is shown below, this is accurate to the game alone and no player data already saved within the game. The Unity website also states that it would need at least 100Mb of space on the hard diskspace to run the game properly.

At the start of the game (when the game is run) it has a CPU usage as shown below:

Then, after 2 minutes of processing and the game running the CPU usage changes to:

This is due to the use of UI and physics modelling within the program mostly, and of course, the increase in frames per second:

Then the frames per second changes to:

Programming Options

C# is an object orientated, component-oriented programming language. C# provides language constructs and helps directly support difficult concepts, which makes C# a natural language in which to create and use its software components. Its roots are in the C family of languages making C# familiar to C, C++, and Java.

C# is used for Windows desktop app, enterprise solutions and game development but not to a large scale, hence Unity is used.

If I were to code in C#, it would be easier to prepare, study and learn since I already know how to code, and I would not have to learn a new coding language. On the other hand, it would not have good graphics since it

would be hard to implement this in a forms app, but since the design is not needed, I assumed this as a non-affecting factor.

Java is a widely used object-orientated programming language and software platform. The rules and syntax of Java are based on C and C++ languages, making it easier for me learn the language since it would be the same rules and syntax as C#. Java runs on most operating systems, including Windows, Linux, and Mac OS. Java was designed to be easy to use therefore it is easy to write, compile, debug and learn than other programming languages.

The reason that Java is one of my options for my project, is because Java is platform independent which means that I would not need to install any more platforms as the background.

In the end, I decided to use the development platform/game engine called Unity.

Unity platform is written in C++ but is designed so that the programming language for coding is C#. This means I would not have to learn another programing language since Unity would be compatible with C# and the design. The only thing to consider is that I cannot use any libraries because they take up a lot of code and are used in a simplified version that will not get me any marks and will not be easy to understand for someone else reading and trying to understand the code.

Diary of Research

I have researched a lot of information about search algorithms, UI Design, SQL, random generator algorithms, programming languages, development platforms and how to use Unity.

When researching search algorithms, I used:

Maze Generator Algorithm ( )

Mapping Algorithm ( )

A Comparison of Pathfinding Algorithms ( )

Maze Generator using tree algorithm ( )

UI Design links and websites:

Learn Unity - Beginner's Game Development Tutorial ( )

Make Your FIRST COMPLETE Game in Unity ( )

Dealing damage to AI using hitboxes, ragdolls, and healthbars in Unity ( )

Character Controller With Animations ( )

HOW TO MAKE A SURVIVAL HORROR GAME IN UNITY TUTORIAL ( )

How to Level Up [Unity Tutorial] ( )

Points counter, HIGH SCORE and display UI in your game ( )

On Trigger Enter Collider information ( )

SQL Information and tutorials:

SQLite in Unity simple implementation with UI ( )

Unity 3D C# MYSQL Database Connection ( )

Unity tutorial: High score with SQLite Intro ( )

Unity - How to Keep Score ( )

Using SQL in Your Web Site ( )

Install | MariaDB Tutorial for Beginners ( )

SQL Tutorial - Full Database Course for Beginners ( )

Programming languages information:

Windows Form Application C# Tutorial ( )

Create a C# Application from Start to Finish - Complete Course ( )

C# Tutorial ( )

How to Make a Battleship Game in Windows Form and C# ( )

Code Easy Website ( )

Worlds hardest Game Scratch ( )

Java Algorithm ( )

Learning Unity and its functions:

34 Enemy follower - rb MovePosition in Unity ( )

Points counter, HIGH SCORE and display UI in your game - Score points Unity tutorial ( )

Unity NavMesh Tutorial - Making it Dynamic ( )

C# Basic 2D Game engine from scratch! ( )

Unity Tutorial (2021) - Adding Enemies ( )

Ideas and Innovation:

How to make a Wave Spawner in Unity 5 - Part 1/2 ( )

Nav Mesh Agent follows the player (3D) || Unity Tutorials ep.4 ( )

First person movement ( )

How To Pass a Text From Scene1 Scene2 - Unity C# 2021 ( )

Design

High Level Overview

My project consists of a maze generator algorithm that will randomly create a labyrinth/ maze per level on a set of boundaries and ranges that I have set up previously and predetermined for each level scene. The purpose of the game is for the player to finish all the levels and get the best high score without getting killed by the enemy, also to get the best time possible. Since my target audience are young teenagers, I have made the UI easy to understand and the complexity of the game is the same throughout the levels since there is nothing new to learn, its just about speed and further terrain to cover.

In order to achieve this, I have used recursive backtracker algorithm to create a randomized game visual of the maze/ labyrinth; the recursive backtracker algorithm is made up of a series of data structures that work in unison to effectively unite different parts of UI components and form the map, the main data structures used are a stack, a list and 2d arrays which shall be shown briefly in description of algorithms. This was all coded in C#, within Unity.

At first, when I began writing this project, I was initially planning on using C++ to create generator algorithm and planning to use Prims algorithm instead of recursive backtracker! I also didnt plan on using Unity engine to code my project, it started with me trying to code in visual studio through the use of Windows forms app, I managed to code part of a simulation of the algorithm when I realised that in order to see it in the windows forms I would have to attach it to labels, buttons, etc and I would have to create my own UI and code, after careful consideration, I figured that learning a new coding language (C++) and creating my own UI and engine to make it update to frames per second would not be worth the time and effort so opted for Unity instead which helped to focus more on the code side of things and not UI and engine which doesnt get benefit me in any way since the coding for them wouldnt use any real value techniques.

After I started coding in Unity I have been checking and fixing most errors as I advance throughout the program, in order for me to check what was missing in the game, I had to play it, so every so often, I play the game and I notice the errors that come up as I add new components to each class, for example, when I designed the maze and put the player in the map, it was only able to go at a certain speed, so you werent able to hold the keys because the player would start levitating and leave the designated area. To fix this, I implemented a gravity component and made it the same as the Earths gravitational strength: 9.81, fixed it but also created another issue, the player and enemy started going through walls. I had 3 ways to fix it, make a wave spawner, or create a mini-AI controller for the enemy or increase the width of the walls, I opted for a mini-AI controller which managed to fix the problem and now neither of them goes through walls.

The Recursive Backtracking algorithm code generates a maze. It defines a two-dimensional array of type WallState [,] named "maze" that represents the walls of the maze. WallState is an Enum that represents the state of a wall as a bitwise flag, where LEFT is 0001, RIGHT is 0010, UP is 0100, and DOWN is 1000. The Generate method initializes the maze array with initial walls for each cell, and then applies the ApplyRecursiveBacktracker method to generate the final maze.

The ApplyRecursiveBacktracker method uses a stack of Position structs to keep track of the current position in the maze. It also uses a GetUnvisitedNeighbours method to get a list of neighbouring cells that haven't been visited yet. It then randomly selects a neighbour and removes the wall between the current cell and the selected neighbour. The GetOppositeWall method is used to get the opposite wall to remove. The current position is then updated to the selected neighbour, and the process is repeated until all cells have been visited.

The GetUnvisitedNeighbours method takes a Position struct and returns a list of Neighbour structs representing the neighbouring cells that haven't been visited yet. It checks each neighbouring cell to see if it has been visited, and if not, adds it to the list of neighbours. The Neighbour struct represents a neighbouring cell with its position and the shared wall between the current cell and the neighbouring cell. The GetOppositeWall method takes a WallState and returns the opposite wall, which is used to remove the wall between the current cell and the selected neighbour. It uses a switch statement to return the opposite wall based on the input WallState.

Overall, the code uses stacks and lists to keep track of the current position and neighbouring cells in the maze, and bitwise flags to represent the state of the walls between cells. The recursive backtracking algorithm is used to generate the maze by removing walls between cells in a randomized manner.

Description of Algorithms

Modular Design Diagram

Player Movement

This algorithm moves the player in the x-axis and the z-axis, it saves the input of the user and saves it to either the moveX variable or the moveZ variable then it keeps constant the other axis and gains speed on stated user input.

Enemy

The algorithm below uses an already defined variable Enemy and uses a search algorithm to find the player and then chases the player in the start function. If the update function were not implemented, then the enemy would just go to the beginning place where the player spawns, had the player moved a bit, the enemy would never reach the player. So, to stop this, the update function calls the search algorithm every Time.time is bigger than next action time which is an already predefined variable with a value. In my case, it would call the search algorithm every 5s and would reroute towards the new player position so it would never stop chasing the player.

Array

The array that I used is a two-dimensional array of type WallState [,] that I named maze. Its used to represent a maze by storing the state of each cell in the maze, where each cell can have one of the WallState values: Left, Right, Up, Down and Visited. The maze is generated using the ApplyRecursiveBacktracker method which applies the recursive backtracking algorithm to the maze represented by the WallState [,] array.

Stack

The Stack data structure is used to keep track of the positions to visit in the maze.

List

Uses a List of type Neighbour to store the unvisited neighbours of a given cell in the maze.

The Neighbour struct contains information about the neighbouring cell's position and the shared wall between

the two cells.

Maze Renderer

The MazeRenderer class has several private variables that are serialized so that I can modify them in the Unity editor:

Width: an integer that determines the width of the maze.

Height: an integer that determines the height of the maze.

Size: a float that determines the size of each cell in the maze.

WallPrefab: a Transform that represents the wall prefab to be used for rendering.

FloorPrefab: a Transform that represents the floor prefab to be used for rendering.

The Start () method generates a maze using the MazeGenerator.Generate() method and passes the resulting maze to the Draw () method. The Draw () method uses a nested for loop to iterate through each cell in the maze and render its walls and floor. It instantiates wall and floor prefab and positions them accordingly based on the location of the current cell in the maze.

User Interface Design

This screen is called LevelSelection, this is the main screen that will be displayed when the game is started which will determine which level the player wishes to play. Normally, the player should start from level 1 and every time they beat the level, they would advance to the next level till level 4 which is the highest. The screen was achieved using a canvas which is defined as a Game Object, the canvas is the basic component of Unity UI. It generates meshes that represent the UI components/ elements placed on it, it then issues draw calls to the GPU so that the UI is displayed.

The Canvas holds all the UI components such as the title, the buttons, and the numbers within the buttons hence text as well.

Title

The title was achieved using a component called TextMeshPro Text (UI) with coordinates/ transform position in the X-axis of -55, Y-axis of 144 and Z-axis of 0. Writing the Select Level was written in font size of 36. The dimensions of the title rectangle dont affect the position when creating the title because it will be created inside the Canvas and the Canvas fits itself in the screen size, so it hasnt got a position it needs to be in.

Buttons

The buttons are coded and linked so that when the user presses said button, it will direct the user to stated level. The numbers are on top of the buttons, which means that the button originally did not have text input. I created a text input on top of the buttons and wrote the numbers according to the order of the button.

Pause Menu Scene

The pause menu will only become active when the game has started, it originally wont be visible but by clicking esc on the keyboard, the screen will show a text message saying the game is paused and a list of buttons that will let the user decide options from the Pause Menu like Resume, Menu and Quit buttons.

This is how level 1 is going to look after you click the button:

Data Structures

Player

Below are the script components, but the player object class has some attributes that are not from scripts:

Transform Component To dictate the position in the 3d map (x, y, and z vector components)

Cube (Mesh Filter) What the player is, a cube.

Mesh Renderer Needed for development.

Box Collider Mainly used to collide against objects and enemy but since my map is generated, I have had to code the collider myself. I use the Box Collider for No Friction against the floor.

RigidBody and CharacterController are needed for the player scripts to run so they are used for development.

Enemy

Below are the script components, but the Enemy object class has some attributes that are not from scripts:

Transform Component To dictate the position in the 3d map (x, y and z vector components)

Cube (Mesh Filter) What the enemy is, a cube.

Mesh Renderer Needed for development.

Box Collider Mainly used to collide against objects and enemy but since my map is generated, I have had to code the collider myself. I use the Box Collider for No Friction against the floor.

RigidBody and Enemy Material are needed for the player scripts to run so they are used for development.

Nav Mesh Agent This used to enable change to the enemys attributes like speed, angular speed,

acceleration, and obstacle avoidance.

Array

One of the arrays used is called maze which has parameters of width and height, which are modelled later as [i, j] in terms of its vector composition. The array focuses on creating the actual layout of the maze, dimensions, sizes, and distances. Since its 3D, it also includes the z component, which is the walls thickness, but this is all pre-made using a prefab of dimensions which will be explained below in the Design of Maze.

List

The list has been created to randomize the backtracking algorithm and the stack below.

The list uses the public struct called neighbour and organizes the different options for the neighbour nodes. There are 4 options, up, down, left, and right so the list organizes the walls, in my program called as WallState and identifies if there is a wall between the node its currently in and the neighbour nodes, if theres a wall on the left, it will then check the right and so on. For the list to always have 1 exit so a path is created, a SharedWall is added so that when all 3 walls are covered then the last side of the node will be the continuation of the path, it will then mark the node as visited and go to the next node and so on.

Stack

The stack created is called positionStack its role is to check the width and height of each position and place the floor and wall at random. It then checks the neighbour nodes width and height and checks if it has been visited if it hasnt, then the stack will decrease the count of neighbours that are currently free. This count has been named neighbours. Count it will continue moving to the neighbours' nodes at random and creating walls and the floor by the Push method of current node.

Design of Maze

Since the game is a maze generator, there is no terrain before starting the game so the objects and the player will drop due to gravity when the game is run. To stop this from happening, I have transformed the position axis of the X and Y component from the player and the objects such that they are directly above the maze so when the maze is run, the change in the Y axis will be miniscule and will not affect the game.

The game is seen and played in 2D, but the design of the maze generator is 3D and the objects and the enemies are also 3D which means that I am coding it for 3D functions, this can be seen in the actual game by

the use of shadows, since its in 3D, the walls have shadows which make the game more realistic. The player and the enemies will move on the board and the players movement will be the up, down, left, and right keys. The player is a 3D cube of dimensions 0.4x0.4x0.4, the dimensions of the player are directly proportional to the walls dimensions. The walls dimensions are 1.0 on the X- Axis, 1.5 on the Y-Axis and 0.1 on the Z- Axis.

By coding my project in 3D, I changed the main cameras angle and position so that the game is seen from above to give it a 2D look.

The player has a component of a RigidBody, and gravity implemented to stay on the board of the maze and not pass-through walls or fall below the board.

As an extra objective, I could change the cameras settings and amend the code to make an extra level in 3D but this an extension if I have the time.

Prefabs

Prefabs are pre-built game objects that can be reused multiple times throughout a game. They are essentially templates for creating multiple instances of the same object. I have to design them myself and save it as a prefab in order to reuse it, I have 5 prefabs in total; Button, canvas, floor, wall2D and wall3D and with the help of coding, I can implement them with UI and construct, in my case, a maze.

The 5 prefabs are:

Database Design

I had been looking to code in MySQL or azure (Microsoft) or SQLite, I will show why I chose SQLite at the end to do my database in. The reason why I want to use a database with Unity is to store and manage game data such as player information, game progress, and scores. By using a database, you can store this information in a centralized location that can be accessed by multiple players and devices. Also, a database can be used to create a management system that is commonly used for storing, organizing, and managing large amounts of data, in my case I would use it to save the scores, usernames and high scores of the players.

MySQL

MySQL is an open-source relational database management system that is mainly used for storing and managing large amounts of data. It is a good choice for online games and multiplayer games that require a

centralized database to manage game state and player information. MySQL is well-suited for handling complex queries and has good scalability, making it a good choice for games with large amounts of data.

Azure

Azure is a good choice for developers who want to take advantage of cloud computing and do not want to manage their own database infrastructure. It also offers integration with other Azure services, making it a good choice for games that require additional cloud-based functionality. Its a fully managed relational database service that offers high availability, scalability, and security.

SQLite

SQLite is a lightweight, file-based relational database management system that is commonly used in mobile and desktop applications. It is a good choice for games that do not require a centralized database or online functionality. SQLite is easy to use, has good performance, and requires minimal setup and configuration.

Decision

In the end I decided to use SQLite since it has simpler data management requirements and its easy-to-use, also easier to incorporate within Unity. For my purpose of the use of my database, I dont need access to the cloud or integration to other services, and my game is not online nor needs a complex centralized database hence I decided cancel MySQL and Azure and use SQLite instead.

ER Diagram

Some of the code used for the connection of the database is shown below:

Usability Features

Intuitive user interfaces that are easy to navigate and understand for the user/player, the game will be played using the up, down, left, and right keys to make it simple for every age player. The high score and the lowest time are to 2sf (significant figures) degree of accuracy to make it easier to understand, to remember and to compare, etc... since competitivity is always best.

Technical Solution

Objectives Data from Analysis

Code Listing

To write and document my code, I shall be using the names of the class being documented and the numbers

alongside the code since I cannot fit all my code into a single screenshot.

Maze Generator Class Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
//Enum which is named WallState, represents the state of a wall.
//Each wall can have one of the following states: LEFT, defined as 1, RIGHT, defined as 2, UP, defined as 4,
DOWN, defined as 8, or a combination of these states.
//The VISITED state is defined as 128.
//Used to indicate that the values of the Enum can be combined using the bitwise OR operator.
[Flags]
public enum WallState
{
//0000 -> NO WALLS
//1111 -> LEFT, RIGHT, UP, DOWN
LEFT = 1, //0001
RIGHT = 2, //0010
UP = 4, //0100
DOWN = 8, //1000
VISITED = 128, //1000 0000
}
//Fields X and Y of type int represent the position of an element in a two-dimensional array.
public struct Position
{
public int X;
public int Y;
```

```
}
//Position is an instance of the Position struct representing the position of a neighbouring element.
//SharedWall is an instance of the WallState Enum representing the shared wall between the two neighbouring elements.
public struct Neighbour
{
public Position Position;
public WallState SharedWall;
}
public static class MazeGenerator
{
//GetOppositeWall takes WallState as a parameter and returns the opposite WallState.
private static WallState GetOppositeWall(WallState wall)
{
switch (wall)
{
case WallState.RIGHT: return WallState.LEFT;
case WallState.LEFT: return WallState.RIGHT;
case WallState.UP: return WallState.DOWN;
case WallState.DOWN: return WallState.UP;
default: return WallState.LEFT;
}
}
//Two-dimensional array of type WallState[,] named "maze"
//The maze is generated using the ApplyRecursiveBacktracker method
//Which applies the recursive backtracking algorithm to the maze represented by the WallState[,] array.
private static WallState[,] ApplyRecursiveBacktracker(WallState[,] maze, int width, int height)
```

```
{
//The code below uses depth-first search.
//It creates a new instance of the Random class, which is used to generate random numbers.
//It creates an empty stack to keep track of the visited positions in the maze.
var rng = new System.Random();
var positionStack = new Stack<Position>();
var position = new Position { X = rng.Next(0,width), Y = rng.Next(0,height)};
maze[position.X, position.Y] |= WallState.VISITED; //1000 1111
positionStack.Push(position);
//A loop that will continue until all positions in the positionStack have been visited.
while (positionStack.Count > 0)
{
var current = positionStack.Pop();
//GetUnvisitedNeighbours is assumed to be a separate function that takes the current position, the maze array, and the width and height values as arguments.
//It will then return a list of neighbouring positions.
var neighbours = GetUnvisitedNeighbours(current, maze, width, height);
//Checks that there are no unvisited neighbours.
if (neighbours.Count > 0)
{
positionStack.Push(current);
var randIndex = rng.Next(0, neighbours.Count);
var randomNeighbour = neighbours[randIndex];
var nPosition = randomNeighbour.Position;
maze[current.X, current.Y] &= ~randomNeighbour.SharedWall;
maze[nPosition.X, nPosition.Y] &= ~GetOppositeWall(randomNeighbour.SharedWall);
maze[nPosition.X, nPosition.Y] |= WallState.VISITED;
```

```
positionStack.Push(nPosition);
}
}
return maze;
}
//GetUnvisitedNeighbours method takes a Position object, a 2D array of WallState objects, and two integers representing the width and height of the maze.
//The method first creates a new empty list of Neighbour objects.
private static List<Neighbour> GetUnvisitedNeighbours(Position p, WallState[,] maze, int width, int height)
{
var list = new List<Neighbour>();
if (p.X > 0) // LEFT
{
if (!maze[p.X -1, p.Y].HasFlag(WallState.VISITED))
{
list.Add(new Neighbour
{
Position = new Position
{
X = p.X - 1,
Y = p.Y
},
SharedWall = WallState.LEFT
});
}
}
if (p.Y > 0) //BOTTOM
```

```
{
if (!maze[p.X, p.Y - 1].HasFlag(WallState.VISITED))
{
list.Add(new Neighbour
{
Position = new Position
{
X = p.X,
Y = p.Y - 1
},
SharedWall = WallState.DOWN
});
}
}
if (p.Y < height -1) // UP
{
if (!maze[p.X, p.Y + 1].HasFlag(WallState.VISITED))
{
list.Add(new Neighbour
{
Position = new Position
{
X = p.X,
Y = p.Y + 1
},
SharedWall = WallState.UP
});
```

```
}
}
if (p.X < width -1) //RIGHT
{
if (!maze[p.X + 1, p.Y].HasFlag(WallState.VISITED))
{
list.Add(new Neighbour
{
Position = new Position
{
X = p.X + 1,
Y = p.Y
},
SharedWall = WallState.RIGHT
});
}
}
//the method returns the list of neighbouring cells that have not been visited.
return list;
}
//Generate method initializes the maze array and returns the final maze
public static WallState[,] Generate(int width, int height)
{
WallState[,] maze = new WallState[width, height];
WallState initial = WallState.RIGHT | WallState.LEFT | WallState.UP | WallState.DOWN;
for (int i = 0; i < width; i++)
{
```

```
for(int j = 0; j < height; j++)
{
maze[i, j] = initial;
}
}
return ApplyRecursiveBacktracker(maze, width, height);
}
}
```

Maze Renderer Class Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class MazeRenderer : MonoBehaviour
{
[SerializeField]
[Range(1,50)]
private int width = 10;
[SerializeField]
[Range(1,50)]
private int height = 10;
[SerializeField]
private float size = 1f;
[SerializeField]
private Transform wallPrefab = null;
[SerializeField]
private Transform floorPrefab = null;
//The Start() method generates a maze using the MazeGenerator.Generate() method and passes the
```

resulting maze to the Draw() method.

```
// Start is called before the first frame update
void Start()
{
//Generates a visual representation of the maze using prefabs that represent the walls and floor of each cell.
var maze = MazeGenerator.Generate(width, height);
Draw(maze);
}
//The Draw() method uses a nested for loop to iterate through each cell in the maze and render its walls and floor.
//It instantiates wall and floor prefabs and positions them accordingly based on the location of the current cell in the maze.
private void Draw(WallState[,] maze)
{
var floor = Instantiate(floorPrefab, transform);
//Change the value of 0 to 1 to change the whole floor to green.
floor.localScale = new Vector3(width, 1, height);
//Nested for loops.
//They determine the cell's position in 3D space and checks its WallState value to determine which walls should be present.
//It creates a game object which is what will change position.
for (int i = 0; i < width; i++)
{
for(int j = 0; j < height; j++)
{
var cell = maze[i,j];
var position = new Vector3(-width/2 + i, 0, -height/2 + j);
```

```
//The method instantiates a new wall GameObject using the wallPrefab argument and positions it at the top of the current cell.
if (cell.HasFlag(WallState.UP))
{
var topWall = Instantiate(wallPrefab, transform) as Transform;
topWall.position = position + new Vector3(0,0, size/2);
topWall.localScale = new Vector3(size, topWall.localScale.y, topWall.localScale.z);
}
// Positions it at the left of the current cell.
//It sets the localScale and eulerAngles properties to match the size value and rotate the wall 90 degrees on the Y-axis.
if (cell.HasFlag(WallState.LEFT))
{
var leftWall = Instantiate(wallPrefab, transform) as Transform;
leftWall.position = position + new Vector3(-size / 2, 0, 0);
leftWall.localScale = new Vector3(size, leftWall.localScale.y, leftWall.localScale.z);
leftWall.eulerAngles = new Vector3(0, 90, 0);
}
if (i == width - 1)
{
//Positions it at the right of the current cell.
if (cell.HasFlag(WallState.RIGHT))
{
var rightWall = Instantiate(wallPrefab, transform) as Transform;
rightWall.position = position + new Vector3(+size / 2, 0, 0);
rightWall.localScale = new Vector3(size, rightWall.localScale.y, rightWall.localScale.z);
rightWall.eulerAngles = new Vector3(0, 90, 0);
```

```
}
}
if (j == 0)
{
//Positions it at the bottom of the current cell.
if (cell.HasFlag(WallState.DOWN))
{
var bottomWall = Instantiate(wallPrefab, transform) as Transform;
bottomWall.position = position + new Vector3(0, 0, -size / 2);
bottomWall.localScale = new Vector3(size, bottomWall.localScale.y, bottomWall.localScale.z);
}
}
}
}
}
//Update is called once per frame
void Update()
{
}
}
```

Player Movement Class Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayerMovement : MonoBehaviour
{
//Variables
```

```
[SerializeField] private float moveSpeed;

[SerializeField] private float walkSpeed;

private Vector3 moveDirection;

private Vector3 velocity;

[SerializeField] private bool isGrounded;

[SerializeField] private float groundCheckDistance;

[SerializeField] private LayerMask groundMask;

[SerializeField] private float gravity;

//References

private CharacterController controller;

//initializes the reference to the CharacterController component.

//Which is the one used for moving the player.

private void Start()

{

controller = GetComponent<CharacterController>();

}

//Move method handles the player movement.

private void Update()

{

Move();

}

//It checks if the player is on the ground using a Physics.CheckSphere method.

//It checks if there is an object of the specified layer within the specified distance of the player's position.

private void Move()

{

isGrounded = Physics.CheckSphere(transform.position, groundCheckDistance, groundMask);

if(isGrounded && velocity.y < 0)
```

```
{
velocity.y = -2f;
}
//Gets input from the player in the x and z directions using the Input.GetAxis method.
float moveX = Input.GetAxis("Horizontal");
moveDirection = new Vector3(moveX, 0, 0);
moveDirection *= walkSpeed;
//Moves the player in the direction of the moveDirection vector multiplied by Time.deltaTime.
controller.Move(moveDirection * Time.deltaTime);
float moveZ = Input.GetAxis("Vertical");
moveDirection = new Vector3(0, 0.0f, moveZ);
moveDirection *= walkSpeed;
//Moves the player in the direction of the moveDirection vector multiplied by Time.deltaTime.
controller.Move(moveDirection * Time.deltaTime);
}
}
```

FPS Controller Class Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//An attribute that makes sure the game object this script is attached to has a CharacterController component.
[RequireComponent(typeof(CharacterController))]
public class FPSController : MonoBehaviour
{
//Variables
public Camera playerCamera;
public float walkSpeed = 5f;
```

```
public float runSpeed = 10f;

public float jumpPower = 5f;

public float gravity = 10f;

public float lookSpeed = 2f;

public float lookXLimit = 90f;

Vector3 moveDirection = Vector3.zero;

float rotationX = 0;

public bool canMove = true;

CharacterController characterController;

// Start is called before the first frame update

// It Initializes characterController and sets the mouse cursor to be locked and invisible.

void Start()

{

characterController = GetComponent<CharacterController>();

Cursor.lockState = CursorLockMode.Locked;

Cursor.visible = false;

}

// Update is called once per frame.

//Contains the main logic of the FPS controller.

void Update()

{

//Calculates the forward and right vectors relative to the player's rotation.

#region Handles Movement

Vector3 forward = transform.TransformDirection(Vector3.forward);

Vector3 right = transform.TransformDirection(Vector3.right);

//Press Left Shift to run

bool isRunning = Input.GetKey(KeyCode.LeftShift);
```

```
float curSpeedX = canMove ? (isRunning ? runSpeed : walkSpeed) * Input.GetAxis("Vertical") : 0;
float curSpeedY = canMove ? (isRunning ? runSpeed : walkSpeed) * Input.GetAxis("Horizontal") : 0;
float movementDirectionY = moveDirection.y;
//Calculates the current speed of the player in the x and y directions.
moveDirection = (forward * curSpeedX) + (right * curSpeedY);
#endregion
//Checks if the jump button is pressed and the player is on the ground.
#region Handles Jumping
//If both conditions are true, moveDirection.y is set to jumpPower.
if (Input.GetButton("Jump") && canMove && characterController.isGrounded)
{
moveDirection.y = jumpPower;
}
//Otherwise, the y velocity of the player is preserved.
else
{
moveDirection.y = movementDirectionY;
}
//It checks if the player is in the air.
//If true, moveDirection.y is decreased by gravity multiplied by Time.deltaTime to simulate gravity.
if (!characterController.isGrounded)
{
moveDirection.y -= gravity * Time.deltaTime;
}
#endregion
//It moves the player in the direction and with the speed calculated earlier.
#region Handles Rotation
```

```
characterController.Move(moveDirection * Time.deltaTime);

if (canMove)

{

//rotationX += -Input.GetAxis("Mouse Y") * lookSpeed;

//rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);

//Rotates the camera up and down based on rotationX.

playerCamera.transform.localRotation = Quaternion.Euler(rotationX, 0, 0);

//Rotates the player left and right based on the horizontal mouse movement.

transform.rotation *= Quaternion.Euler(0, Input.GetAxis("Mouse X") * lookSpeed, 0);

}

#endregion

}

}
```

On Collision Class Code

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

//Namespaces provide access to Unity's game engine and scene management systems.

public class OnCollision : MonoBehaviour

{

//Detects collisions between the attached object and the player, and reloads the current scene if a collision occurs.

void OnTriggerEnter(Collider col)

{

if(col.CompareTag("Player"))

{
```

```
SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
}
}
```

Enemy AI Simple Class Code

```
//Simple implementation of an enemy AI script in Unity using the NavMeshAgent component
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class EnemyAISimple : MonoBehaviour
{
//Used to store a reference to the enemy AI's navigation agent component, which is used for pathfinding.
public NavMeshAgent Enemy;
public Transform Player;
private float nextActionTime = 0.5f;
public float period = 5.0f;
// Start is called before the first frame update
//Used to set the initial destination for the enemy AI to the player's position.
void Start()
{
Enemy.SetDestination(Player.position);
}
// Update is called once per frame
//Used to update the enemy's target location every period seconds.
void Update()
{
```

```
if (Time.time > nextActionTime)
{
nextActionTime = Time.time + period;
Enemy.SetDestination(Player.position);
}
}
}
```

Pause Menu Class Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class PauseMenu : MonoBehaviour
{
//Variables
public static bool GameIsPaused = false;
public GameObject pauseMenuUI;
//Update is called once per frame
//Checks if the Escape key has been pressed using the Input.GetKeyDown function.
void Update()
{
if (Input.GetKeyDown(KeyCode.Escape))
{
if (GameIsPaused)
{
Resume();
}
```

```
else
{
Pause();
}
}
}
//It hides the pause menu, sets the time scale back to normal (1f), and sets GameIsPaused to false.
public void Resume()
{
pauseMenuUI.SetActive(false);
Time.timeScale = 1f;
GameIsPaused = false;
}
//Shows the pause menu, sets the time scale to 0f (pausing the game), and sets GameIsPaused to true.
void Pause()
{
pauseMenuUI.SetActive(true);
Time.timeScale = 0f;
GameIsPaused = true;
}
//Sets the time scale back to 1f, loads the "LevelSelection" scene.
public void LoadMenu()
{
Time.timeScale = 1f;
SceneManager.LoadScene("LevelSelection");
Debug.Log("Loading menu...");
}
```

```
//Logs a message to the console and calls Application.Quit(), which quits the game.
public void QuitGame()
{
Debug.Log("Quitting game...");
Application.Quit();
}
}
```

The Database Mystery Code

This is where I started having problems, I tried different ways of connecting the database to Unity, I installed SQLiteStudio which is a management app for a database which is used online and on an online server and coded the connection to Unity. I coded in SQLite in three different ways, of which none were successful, then I changed the management service DB Browser but no luck. After a few days of trying, I then switched to My SQL and downloaded the package and phpMyAdmin which is like DB Browser for SQLite but for My SQL, I coded queries in the management app for My SQL and even tried writing a DLL file directly to Unity to solve the issue but couldnt solve it. After a week, I tried to use MySQL Connector/NET library to establish a connection to the database, the code was working perfectly but I still had errors in Unity, so it wasnt connecting. So, after careful consideration and imploring how this would affect the project, I spoke with the Stakeholder of the project to see his opinion.

Solution

After my chat with the stakeholder, we agreed that the high score will be saved but will have to restarted manually. Doesnt change the game or the end aim, its just that it doesnt have a visual table to demonstrate the values so the players would have to remember the values themselves.

Below its some of the code that I used to try make the connection to Unity, but all were unsuccessful. I was using my ER diagram as a template, but I didnt quite manage to advance since the first scripts that were vital for the database to work werent working.

SQLite

```
//Try 1
```

```
//Try 2
using System.Data;
using Mono.Data.Sqlite;
public class HighscoreManager : MonoBehaviour
{
private string connectionString;
private IDbConnection dbConnection;
void Start()
{
connectionString = "URI=file:" + Application.dataPath + "/Highscores.db";
dbConnection = new SqliteConnection(connectionString);
dbConnection.Open();
}
void OnApplicationQuit()
{
dbConnection.Close();
}
public void GetHighscores()
{
IDbCommand dbCommand = dbConnection.CreateCommand();
dbCommand.CommandText = "SELECT * FROM Highscores ORDER BY Score DESC LIMIT 10";
IDataReader reader = dbCommand.ExecuteReader();
while (reader.Read())
{
string playerName = reader.GetString(0);
int score = reader.GetInt32(1);
}
```

reader.Close();

}

}

I would get these errors even though both namespaces and the DLL connector were correctly attached.

My SQL

//Try 1

CREATE TABLE highscores (

id INT(11) NOT NULL AUTO_INCREMENT,

player_name VARCHAR(50) NOT NULL,

score INT(11) NOT NULL,

PRIMARY KEY (id)

);

//Try 2

using UnityEngine;

using UnityEngine.UI;

using MySql.Data.MySqlClient;

public void SaveHighscore(string playerName, int score)

{

string connectionString = "Server=localhost;Database=Highscore;Uid=PlayerName;Pwd=Happy;";

MySqlConnection connection = new MySqlConnection(connectionString);

connection.Open();

string query = "INSERT INTO Highscore (player_name, score) VALUES (@playerName, @score)";

MySqlCommand command = new MySqlCommand(query, connection);

//Error Handling code

command.Parameters.AddWithValue("@playerName", playerName);

command.Parameters.AddWithValue("@score", score);

command.ExecuteNonQuery();

```
connection.Close();
}
```

Score Manager Class Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class ScoreManager : MonoBehaviour
{
//Used to display the current score and high score in the game's UI.
public Text ScoreText;
public Text HiScoreText;
//Will store the current score and high score values.
public float scoreCount;
public float hiScoreCount;
//Rate at which the score increases.
public float pointsPerSecond;
public bool scoreIncreasing;
// Start is called before the first frame update
//It loads the high score value from PlayerPrefs and assigns it to the hiScoreCount field.
void Start()
{
if (PlayerPrefs.GetFloat("HighScore") != null)
{
hiScoreCount = PlayerPrefs.GetFloat("HighScore");
}
}
```

```
// Update is called once per frame
//scoreCount value is compared to the hiScoreCount value, and if it is greater, hiScoreCount is updated, and
the new value is saved to PlayerPrefs.
void Update()
{
if (scoreIncreasing)
{
scoreCount += pointsPerSecond * Time.deltaTime;
}
scoreCount += pointsPerSecond * Time.deltaTime;
if(scoreCount > hiScoreCount)
{
hiScoreCount = scoreCount;
PlayerPrefs.SetFloat("HighScore", hiScoreCount);
}
ScoreText.text = "Score: " + Mathf.Round(scoreCount);
HiScoreText.text = "High Score: " + Mathf.Round(hiScoreCount);
}
}
```

Score Per Second Class Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class ScorePerSecond : MonoBehaviour
{
public Text scoreText;
```

```
public float scoreAmount;

public float pointIncreasedPerSecond;

// Start is called before the first frame update.

//The scoreAmount and pointIncreasedPerSecond are initialized to 0 and 1.

void Start()

{

scoreAmount = 0f;

pointIncreasedPerSecond = 1f;

}

//scoreAmount is increased by pointIncreasedPerSecond multiplied by the time that has passed since the last

frame.

// Update is called once per frame

void Update()

{

scoreText.text = (int)scoreAmount + "Score";

scoreAmount += pointIncreasedPerSecond * Time.deltaTime;

}

}
```

Timer Class Code

```
using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Timer : MonoBehaviour

{

public Text timerText;

private float startTime;
```

```
// Start is called before the first frame update
//startTime is initialized to the current time using Time.time.
void Start()
{
startTime = Time.time;
}
// Update is called once per frame
//The minutes and seconds are concatenated into a string in the format "mm:ss" and assigned to the timerText.text variable, which updates the text displayed in the UI Text component.
void Update()
{
float t = Time.time - startTime;
string minutes = ((int) t / 60).ToString();
string seconds = (t % 60).ToString("f2");
timerText.text = minutes + ":" + seconds;
}
}
```

Exit Door Class Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
public class ExitDoor : MonoBehaviour
{
public int iLevelToLoad;
public string sLevelToLoad;
public bool useIntegerToLoadLevel = false;
```

```
//It checks if the colliding game object has the name "Player", and if so, it calls the LoadScene method.
private void OnTriggerEnter(Collider collision)
{
GameObject collisionGameObject = collision.gameObject;
if (collisionGameObject.name == "Player")
{
LoadScene();
}
}
//The LoadScene method checks the value of "useIntegerToLoadLevel" and loads the scene specified by "iLevelToLoad" if true or by "sLevelToLoad" if false.
void LoadScene()
{
if (useIntegerToLoadLevel)
{
SceneManager.LoadScene(iLevelToLoad);
}
else
{
SceneManager.LoadScene(sLevelToLoad);
}
}
}
```

Testing

I finished coding the program and will now start testing all the classes, functions, algorithms, and settings used in my scripts and in Unity including the UI coded and the UI set up through Unity functions. All the tests are in the tables written below, the first table shows the objectives from the analysis that were specified and

agreed at the beginning with the stakeholder, the last table shows the classes and methods and some of the visuals that the program is supposed to have when the code is ran. Some of the components of the scripts and settings attached to Unity will not be visible due to a few scripts act as a background algorithm. In my case, the maze generator algorithm script cannot be seen visually as its being created, as a viewer we can tell it works because the maze is correctly configured, and its being created randomly hence the algorithm works the way its supposed to. Most of the testing of the program will be done either in the Unity Editor or in the IDE (Integrated Development Environment) to make sure everything works in order.

Test Plan Results

Objectives Results (Compared to the Analysiss objectives)

The objectives data plan is at the start of the technical solution to help orientate on what I need to achieve and what I can add according to time and ability. This section is the results after the code has been finished and I compare it to the original objectives.

Typical Data

Typical Data Results

The classes and the data required for the game to work and be playable. As stated in the objectives, every single result in this section should pass since this is standard requirement of the clause of the game agreement.

Player Movement()

Timer()

Enemy()

Pause Menu()

Maze Generator()

Maze Renderer()

Score Manager()

Exit Door()

FPS Controller()

Main Menu Scene()

Evidence

Maze Generator Evidence

Maze Generator script shown in the technical solution attached to unity in my project.

In the screenshots below, I show the floor and wall prefabs which make the maze. The maze is always random, and this shall be shown in later images as well.

The image below represents the view of the maze generator in game from the unity editor view, how I edit the dimensions and the camera view as well. The rest of the images are of levels of the game showing that every time the game reloads, its a completely different game. This will also be shown in the demonstration video.

Maze Renderer Evidence

The maze renderer uses the maze generator script algorithm and unites the prefabs it changes the size of the maze in terms of width and height.

Player Movement Evidence

The player moves from the first position in the beginning to the 2nd position, as you can see, the mazes are identical since the photos are both from the same game just 15s later.

In the photos underneath, it shows the attributes of the player class that is connected to the game object defined as player. Most of the attributes are predefined by Unity like Transform and RigidBody, but I still must change its dimensions according to the size/ distance of the maze.

Other attributes like the Character Controller and Player Movement Script, I added to make the game object move and to connect it to the keys, also to specify its variables which form part of the game. (Gravity, Ground, Move and Walk)

FPS Controller Evidence

Level 5 uses the same backtracking algorithm and the same classes; the only difference is that the player has a script of FPS controller (for 3D movement) instead of the player movement script which is for 2D movement. Since the maze was originally designed in 3D, the only thing to create the 3D maze was to unite the camera to the player and change the angle of projection to the players vision instead of looking at the maze from above like in the other levels.

Below is the player and the camera view:

Enemy AI Evidence

The cube below is a game object representing the enemy AI, without the code and its attributes, the cube would just fall below into nothingness. In the photos below, I show the area (green floor) which the enemy uses to mobilize since it doesnt recognise the floor created by the algorithm since its random and only created when the game is started. This was fixed by adding an enemy floor which basically is the navigation area for the enemy where he chases the player if in that area, since the maze is smaller than said area, the player will always be as well so the enemy will chase the enemy no matter what.

Pause Menu Evidence

Only appears when Esc (escape) is pressed.

Score Manager Evidence

Timer Evidence

Exit Door Evidence

Final Settings and Details

Results of Testing

All the objectives that were a must, were met, all the objectives in the should were met as well and almost all of the could objectives were completed. The main functions and the working algorithms and classes were very successful.

Evaluation

Objectives Comparison

I have reached the end of my program; this is my final version of my game which contains most of the features and specifications required by the user and meets almost all of the objectives predefined, I will now explain the requirements and objectives, which ones were met, and which ones werent.

Have a randomizer algorithm to create the map/maze (No game should be the same map).

This was met to its full extent; I created a recursive backtracking algorithm that worked with the prefabs designed in Unity. The maze generator script connects to the maze renderer and randomises the map created.

Have a Menu Scene with access to all the levels.

This was met to its full extent; the menu scene connects to all the levels, and it is formatted to be the first scene to appear when you play the game so you can directly choose which level you want to go into.

Pause Menu accessible whenever the player wants (In Game) through pressing keys in the keyboard.

This was met to its full extent; The pause menu can be called from any scene level by pressing Esc key in the keyboard, the functions within the pause menu work fully but the quit function only works in the phone, it doesnt work in the unity IDE.

Pause Menu should stop the game and have labels, buttons.

This was met fully to its extent; it has 3 buttons, resume, menu and quit, all with fully working functionalities. The physics and time components of the game stop when esc is pressed.

Door, portal to advance to next level, all levels to be connected.

This was met to its full extent; the door is fully connected to all the levels and the last level has 2 doors as well!

2D interpretation.

All of the levels have a 2D level interpretation except the bonus 3D level which I had time to include.

Visual Image of maze, player, and the door.

This was met to its full extent; all of the game Objects are fully visible and can be easily differentiated due to the contrast in colours in the creation of the project.

Enemy creation that chases player.

This was met to its full extent; The enemy player has an A.I method and functioning that basically chases the player on a certain area, since the area is the whole maze, it will constantly chase the player.

Active Timer that stops when Pause Menu is active.

This was met to its full extent; the timer is coded to stop included with the physics and movement of the enemy when the pause menu function is called.

Score Count directly proportional to the timer.

This was met to its full extent; As the time increases, so does the score count.

Path from start (starting point of player) to finish (door/exit/portal).

This was met to its full extent; The recursive backtracking algorithm has a searching algorithm within it that

makes sure there is always a path from the start of the maze to the end.

Be a tractable problem (solvable in reasonable amount of time).

This was met to its full extent; The game is a tractable problem since if you cant complete the maze, then the enemy will basically kill you and you will have to restart the level.

Player Death when colliding with Enemy.

This was met to its full extent; the player dies and the level restarts, the score also restarts when enemy collision happens.

Save highest score and keep it in game so the player can try to beat it.

This was met to its full extent; the high score is saved using Player Prefs and successfully stays saved when open.

Have a maze renderer (software process that generates a visual image from a model).

This was met to its full extent since a maze renderer is connected accordingly to the maze generator and fully creates the dimensions of the maze.

Have a stop button; to be able to pause the game, a stop button in the screen or by pressing a desired key in the keyboard.

This was met to an extent since the game stops with the pause function, but it doesnt have a button labelled as stop. But the functionality is still the same.

Advance into harder level of complexity every time player goes through a portal/door.

This was met to its full extent. The dimensions increase and the area of game play also increases to make it more complex to make it to the door.

Be the same wall dimensions per scene.

This was met to its full extent; since the dimensions do not change when playing the game nor do they change when you die and respawn.

Be the same floor dimensions per scene.

This was met to its full extent; since the prefab is predetermined, so are their dimensions.

Have the timer to two significant figures.

This was met to its full extent since the timer is coded to only show 00:00 hours and seconds.

Have a score count and a variable high score count which changes when the score count increases higher than the current high score.

This was met to its full extent; The scenes have 2 boxes on the top left and top right for the scores and the high score saving which increases if score passes high score.

Have a resetting function; it should have a button or function in the keys to reset the game.

This was met to an extent since there isnt a resetting function or button, but you can pause the game click main menu and click the same level/scene and you have reset the game!

3D game version of the 2D algorithm.

This was met to its full extent since I created a full working game of a maze in 3D where the player works as if he is in an FPS shooter game except the guns of course.

Definite speed throughout the level and as you advance, enemy increase in speed.

This was met to its full extent, since the enemy does increase in speed when the level gets bigger.

Range of enemy, minimal (on collision with player).

This was met to its full extent since the range is very small (0.05x dimensions) so hence the player can evade the enemy even if the player goes into a blocked passage.

Unmet Specification Requirements

Most of the objectives and what we agreed upon was completed except the stats information scene. Like I explained in Technical Solution in real time, I tried and put a lot of effort in trying to figure out a way to complete it, but it ended up being unsuccessful. I contacted the Stakeholder and told him about what happened, and his response was a bit dry, and he seemed annoyed, but he accepted it since after all, the program does save the high score just cannot compare it.

Stakeholder Response.

Added Bonus and Limitations that were accomplished.

The code for the Enemy, uses AI, not complex but AI nonetheless, it uses an On Collision script instead of a range and the line of vision is infinity.

Stakeholder Views on the finished project