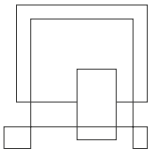


PROJECT REPORT

Group members:

Andrei Enoiu (253668)

Oleg Eni (253977)

Project Supervisors:

Jakob Knop Rasmussen (JKNR)

Kasper Knop Rasmussen (KASR)

IT-SEP4C-S18 ICT ENGINEERING

4TH SEMESTER

07/06/2018

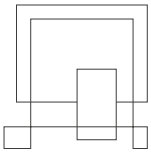
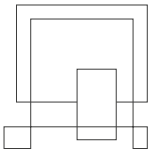


Table of contents

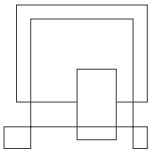


Abstract

The contents of this project report are relevant for the full understanding of the way the memorization training game "Crazy Maze" has been implemented, with all the requirements necessary in building up a program that copes with the stakeholder's desires.

The aim of this project is to provide a "memorization training" activity in a different way. Building a game that serves this purpose and greatly improves the user's short-term memory comprehension is a challenging, but fascinating concept that has attracted our attention and motivated us in many ways during the period of this project.

Our solution to a faster and easier memorization process, the "Crazy Maze" game, was built in the Unity game engine and is compatible for Android mobile devices, as requested by our stakeholders Ensight Games. The main target audience for our game, for the most optimal memorization activity, are children and elders, but we can safely say that anyone can enjoy the game, as the mechanics are as simplified and easy to learn as possible and very addictive for people of all ages.

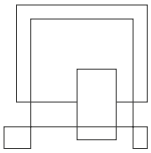


1 Introduction

Background Information:

For as long as scientists have analysed, explored memory, they have strived, and often failed, to improve its significance. Most approaches to improve memory implement strategies, such as creating mnemonic devices or memorising activity courses. However, despite evidence that these techniques improve memory *performance*, they do not completely train the brain and do not target underlying memory *processes*. While they do have some influence on memory systems in the brain, they typically fail to broadly generalize to untrained activities.

Video games are a new concept considering for how long we, humans, have been living on Earth. From this perspective, we can assume that there are a lot of things yet to be explored about concepts and activities that can be improved by playing games. Reaching a level of understanding high enough to build such games can seem difficult at first glance, but by trying to emulate life experiences or activities that may be useful to us in the future, we can achieve an optimal level of practice by just focusing on a game's quests and objectives. . (Memorise - The Original Memory Gym, 2018)



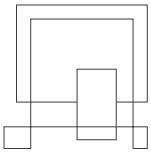
Problem Formulation:

This project's focus is building an interactive mobile game for the purpose of memorization training, with the requirements set by our stakeholder Ensight Games. The game engine which the game is built on is Unity, with compatibility for Android devices, such as phones and tablets.

The main point of interest in this project is finding a reliable way to balance a learning activity with an entertainment activity. We want our player to be put in a position that, in one way or another, simulates an event that may occur in real life. Thus, some questions must be questions about the core mechanics of the game and scalability. The further report will answer these questions and explain the core of the program in advanced details, consisting of the analysis, design, implementation, and testing phases.

Delimitation:

- There will be no mid-session saving as each level will vary in duration from 5 to 10 minutes maximum.
- The application will not be giving feedback on player performance or change difficulty automatically.
- The game is not responsible for any side effects other than improving the player's memorization.



2 Requirements

Functional Requirements:

1. The application must improve the user's ability to memorize something
2. The application must be developed using Unity
3. The application must be developed for phones/tablets
4. The project must contain written documentation (Project & Process Reports)
5. The project must contain relevant diagrams / UML representations

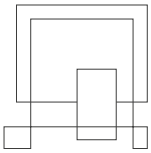
Non-Functional Requirements:

a. The application should be accessible and adaptable to different aspect ratios and OS types

b. The application should provide a level of scalability in order to keep the player interested in the game

c. The UI elements of the game should be reliable and easy to manage / utilise

d. Important aspects of mechanics and user input should be explained shortly (for example User Guide & Readme.txt)



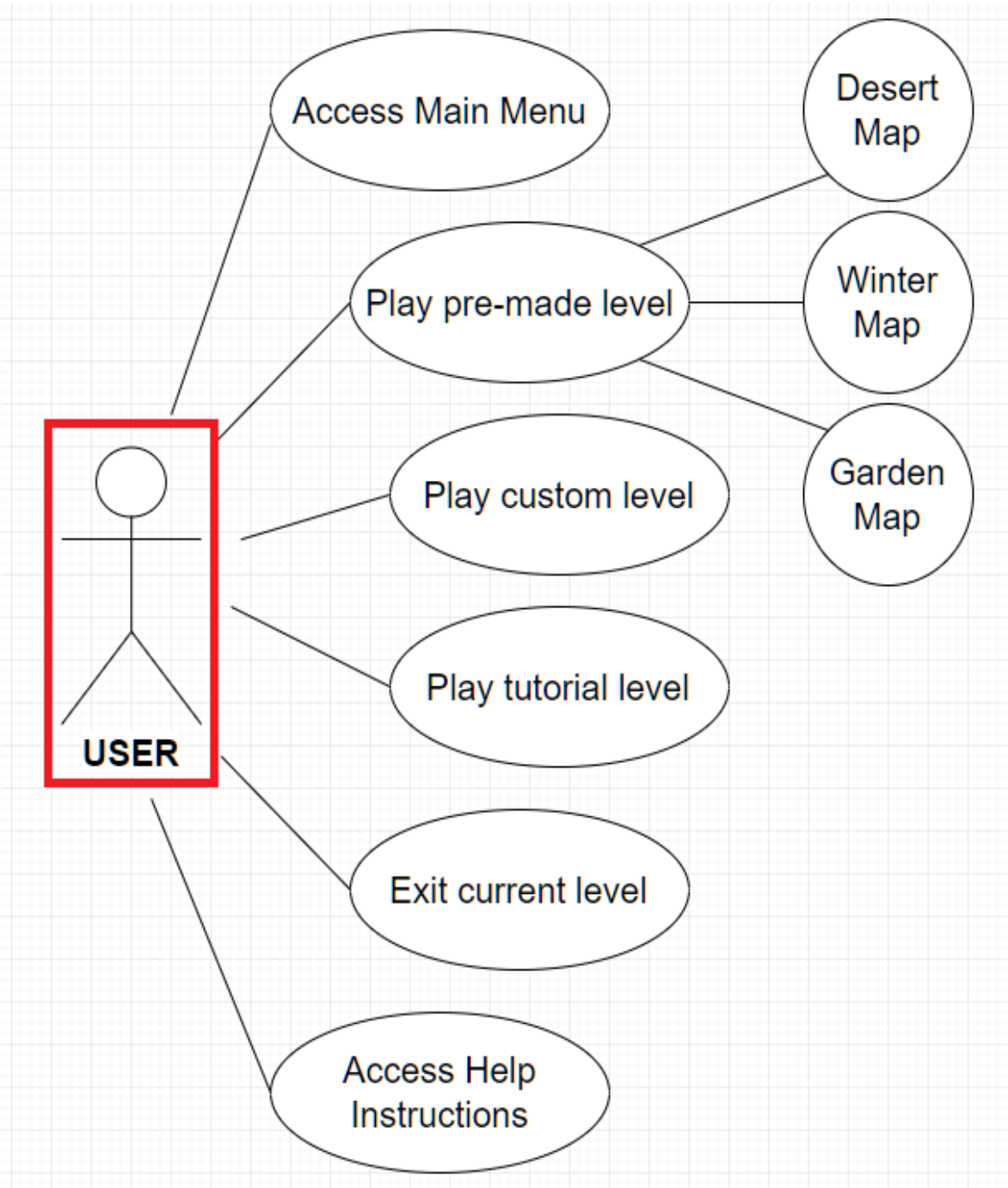
3 Analysis

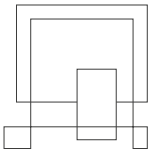
The analysis phase was done by examining and researching the current market of games made for educational purposes, mainly for memorisation. We, then, decided to wisely combine the pieces of information we gained through reviewing with the requirements set by our stakeholder in order to come up with an acceptable and optimal solution.

In our case, the main source of feedback and interest will come from the "User". He / she is the player, the subject of our game's activities and objectives. Thus, the first point of interest for us was to understand what the best way to efficiently develop memory training is. As described in the previous paragraphs, we want to raise awareness for the fact that memorizing is not as difficult as it may seem if you keep working and training your brain. It requires a certain level of practice and understanding of your mental capacities to fully develop a good memorization mind, and at the same time enjoy the game's core mechanics.

The main concept of our game is the following: the player must make his way through a "natural labyrinth" (a maze in a natural environment) to get from one exit to another. Upon reaching the desired destination and being requested to pick one item of his/her choice, the player will be asked to return to the original starting position in a very limited amount of time. Finally, upon completing the previous three goals in this sequence exactly, the player will have to select the number of the item he previously picked. After completing each step, the player wins and the level ends.

Use-Case Diagram:

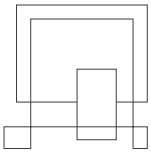




Use-Case Description for Main Menu:

ITEM	VALUE
UseCase	Access Main Menu
Summary	Access the Main Menu of the game by opening and loading the application.
Actor	
Precondition	1. Device is running 2. The "My apps" tab is open 3. The "Crazy Maze" app is installed
Postcondition	1. The user can now access Main Menu buttons
Base Sequence	1. The device is opened 2. The "My apps" tab is open 3. "Crazy Maze" app is opened by tap on icon 4. Loading time for the app to start
Branch Sequence	
Exception Sequence	
Sub UseCase	

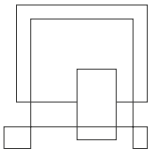
Note: all Use Case Descriptions in the Appendices folder



Use-Case Description for Play Game:

ITEM	VALUE
UseCase	Play pre-made level
Summary	Play a pre-made level from the Play Game button option
Actor	
Precondition	1. The "Crazy Maze" app is installed and running 2. User is in the Main Menu screen
Postcondition	1. The user can now see the Play Game selectionr screen
Base Sequence	1. "Crazy Maze" app is opened by tap on icon 2. Loading time for the app to start 3. User taps the Play Game Button
Branch Sequence	
Exception Sequence	1. The user's device does not suppot this action (rare)
Sub UseCase	

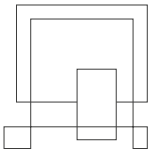
Note: all Use Case Descriptions in the Appendices folder



Use-Case Description for Exit current level:

ITEM	VALUE
UseCase	Exit Current Level
Summary	Exit the current level that is being loaded and return to Main Menu
Actor	
Precondition	1. The "Crazy Maze" app is installed and running 2. User is in any level of the game
Postcondition	1. The user can now see the Main Menu screen
Base Sequence	1. Enter any level of the game 2. Press the EXIT button from the top of the screen 3. Wait for loading of Main Menu
Branch Sequence	
Exception Sequence	
Sub UseCase	

Note: all Use Case Descriptions in the Appendices folder

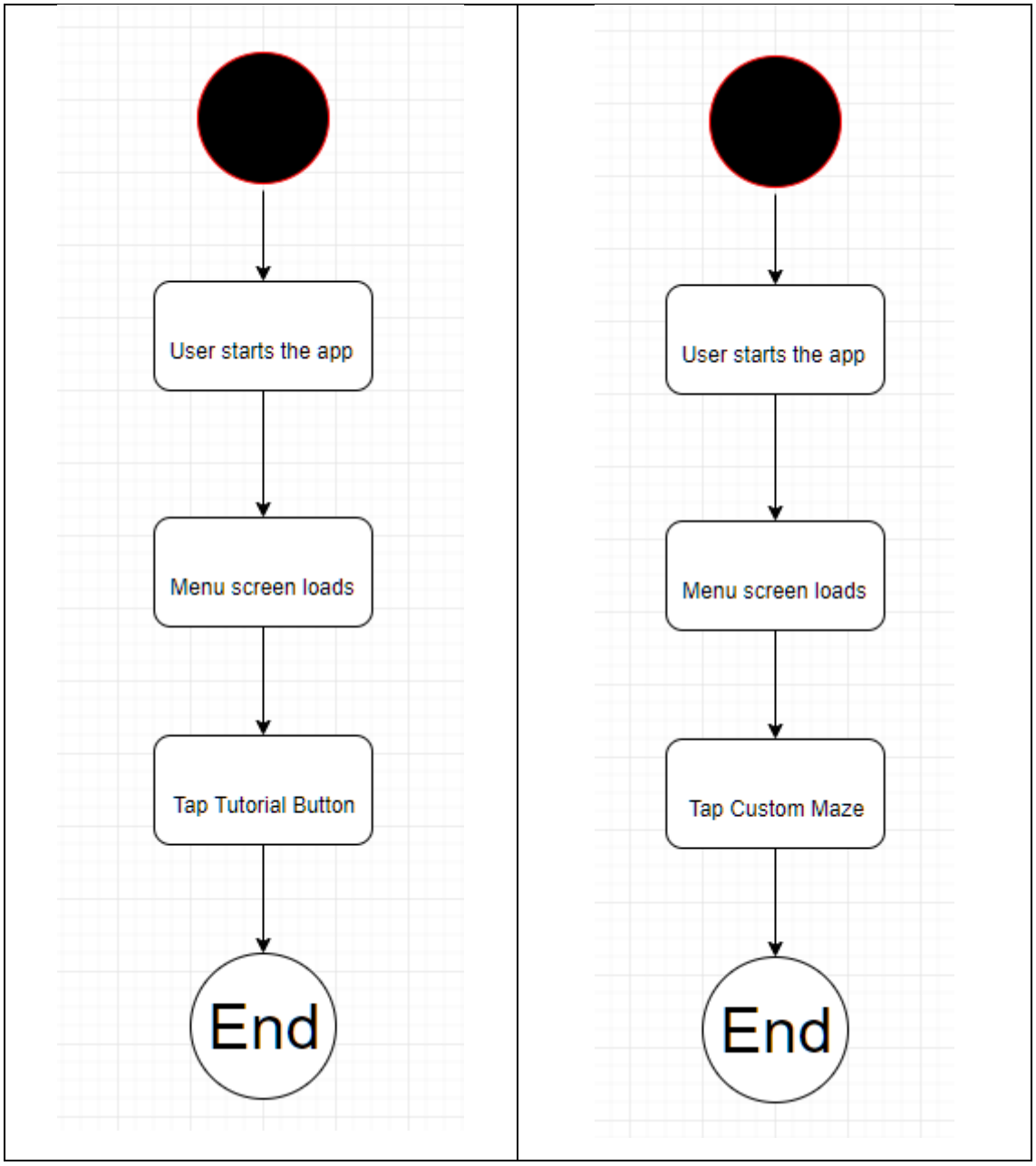


Use-Case Description for Access Help Instructions:

ITEM	VALUE
UseCase	Access Help Instructions
Summary	Access the Help instructions and win / loss conditions by clicking on the Help button in Main Menu
Actor	
Precondition	1. The "Crazy Maze" app is installed and running 2. User is in the Main Menu screen
Postcondition	1. The user can nowread the Help screen instructions
Base Sequence	1. Access the Main Menu screen 2. Tap the Help button at the bottom of the screen 3. To return to Main Menu, press Back button
Branch Sequence	
Exception Sequence	
Sub UseCase	

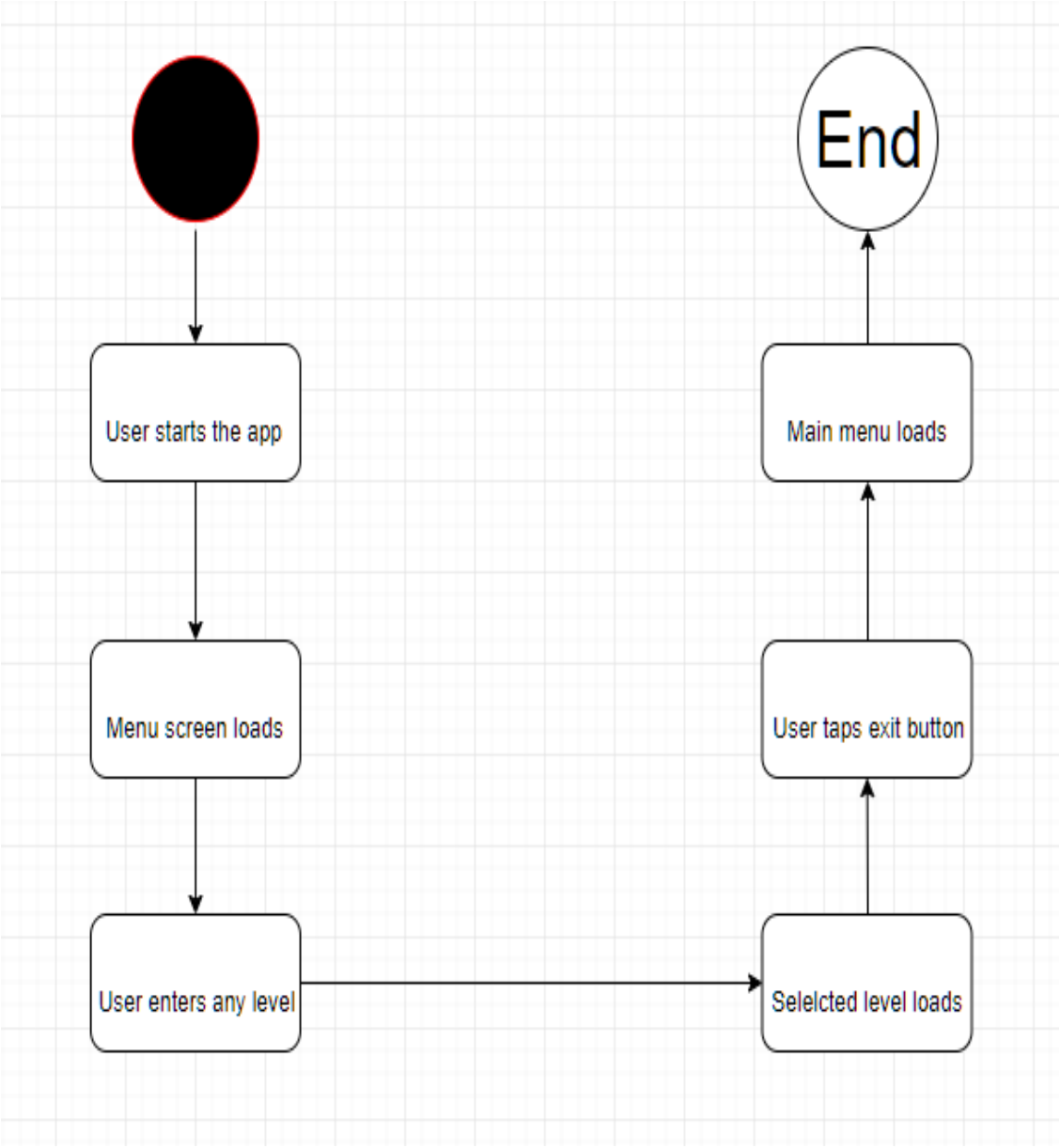
Note: all Use Case Descriptions in the Appendices folder

Activity Diagram for Tutorial and Custom level:



Note: all Activity Diagrams in the Appendices folder

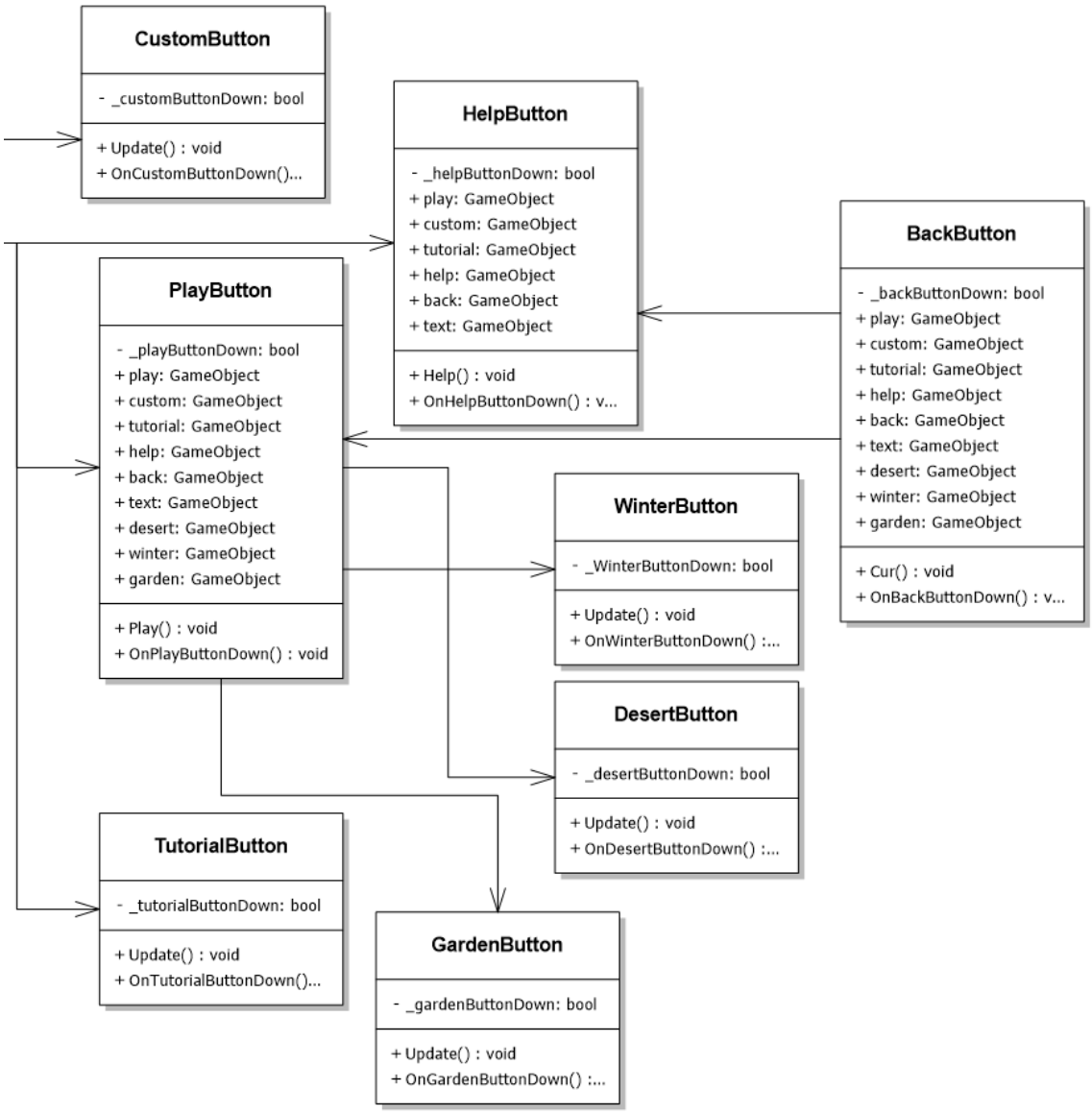
Activity Diagram for Exit level:

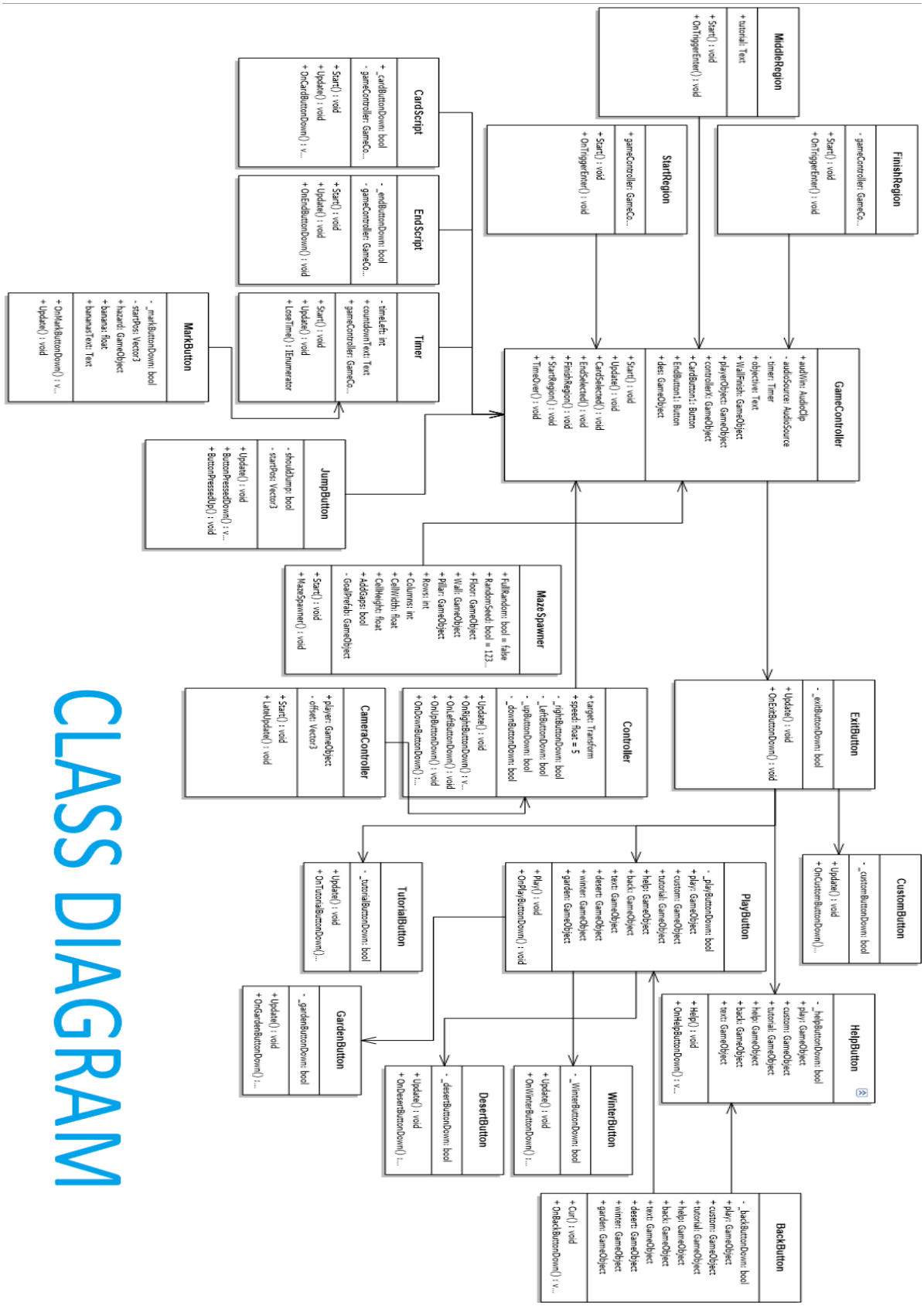


Note: all Activity Diagrams in the Appendices folder

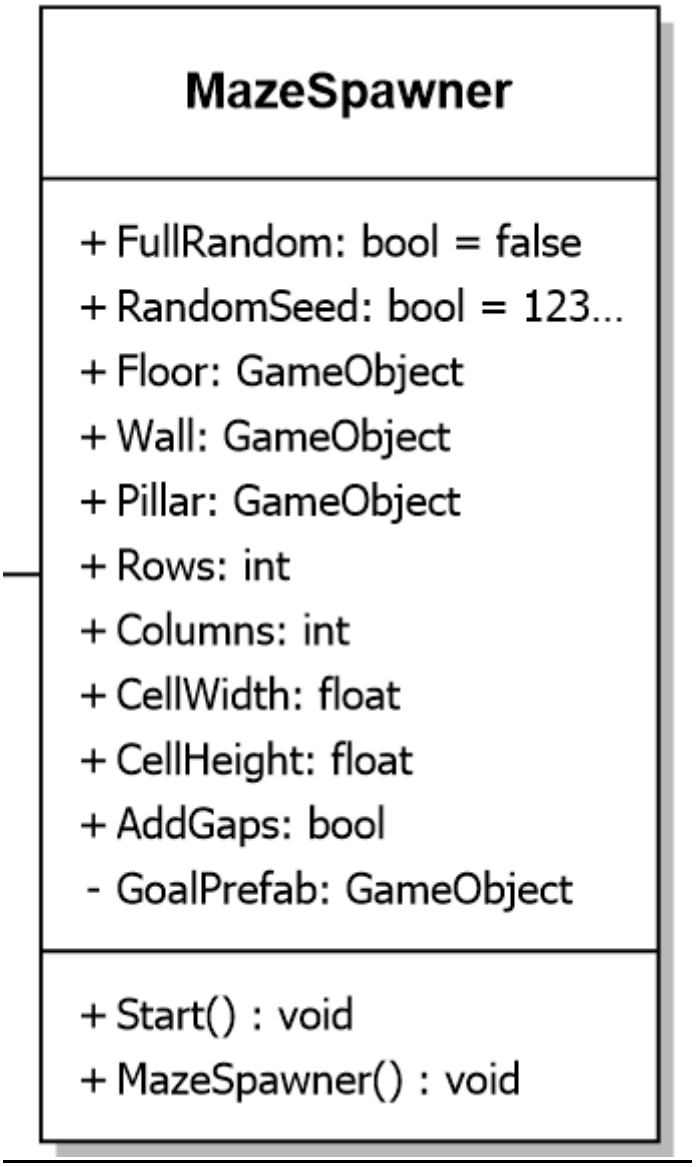
4 Design

The purpose of the Design section is to outline how the system is structured and how we've decided to implement the game mechanics and functionalities described earlier in this project report. Below is the Main Menu class diagram architecture:




CLASS DIAGRAM
MiddleRegion
+ tutorial: Text
+ Start() : void
+ OnTriggerEnter() : void
FinishRegion
- gameController: GameCo...
+ Start() : void
+ OnTriggerEnter() : void
StartRegion
+ gameController: GameCo...
+ Start() : void
+ OnTriggerEnter() : void
GameController
+ audioVoice: AudioClip
- audioSource: AudioSource
- timer: Timer
+ objective: Text
+ WallFinish: GameObject
+ playerObject: GameObject
+ controllerX: GameObject
+ CardButton1: Button
+ EndButton1: Button
+ dest: GameObject
+ Start() : void
+ Update() : void
+ CardSelected() : void
+ EndSelected() : void
+ FinishRegion() : void
+ StartRegion() : void
+ TimeOver() : void
CardScript
+ _cardButtonDown: bool
- gameController: GameCo...
+ Start() : void
+ Update() : void
+ OnCardButtonDown() : v...
EndScript
- _endButtonDown: bool
- gameController: GameCo...
+ Start() : void
+ Update() : void
+ OnEndButtonDown() : void
Timer
- timeLeft: int
+ countdownText: Text
+ gameController: GameCo...
+ Start() : void
+ Update() : void
+ LoseTime() : IEnumerator
MarkButton
- _markButtonDown: bool
- startPos: Vector3
+ hazard: GameObject
+ banana: float
+ bananaText: Text
+ OnMarkButtonDown() : v...
+ Update() : void
JumpButton
- shouldJump: bool
- startPos: Vector3
+ Update() : void
+ ButtonPressedDown() : v...
+ ButtonPressedUp() : void
MazeSpawner
+ FullRandom: bool = false
+ RandomSeed: bool = 123...
+ Floor: GameObject
+ Wall: GameObject
+ Pillar: GameObject
+ Rows: int
+ Columns: int
+ CellWidth: float
+ CellHeight: float
+ AddGaps: bool
- GoalPrefab: GameObject
+ Start() : void
+ MazeSpawner() : void
ExitButton
- _exitButtonDown: bool
+ Update() : void
+ OnExitButtonDown() : void
CameraController
+ player: GameObject
- offset: Vector3
+ Start() : void
+ LateUpdate() : void
Controller
+ target: Transform
+ speed: float = 5
- _rightButtonDown: bool
- _LeftButtonDown: bool
- _upButtonDown: bool
- _downButtonDown: bool
+ Update() : void
+ OnRightButtonDown() : v...
+ OnLeftButtonDown() : void
+ OnUpButtonDown() : void
+ OnDownButtonDown() :...
CustomButton
- _customButtonDown: bool
+ Update() : void
+ OnCustomButtonDown()...
PlayButton
- _playButtonDown: bool
+ play: GameObject
+ custom: GameObject
+ tutorial: GameObject
+ help: GameObject
+ back: GameObject
+ text: GameObject
+ desert: GameObject
+ winter: GameObject
+ garden: GameObject
+ Play() : void
+ OnPlayButtonDown() : void
HelpButton
- _helpButtonDown: bool
+ play: GameObject
+ custom: GameObject
+ tutorial: GameObject
+ help: GameObject
+ back: GameObject
+ text: GameObject
+ Help() : void
+ OnHelpButtonDown() : v...
TutorialButton
- _tutorialButtonDown: bool
+ Update() : void
+ OnTutorialButtonDown()...
GardenButton
- _gardenButtonDown: bool
+ Update() : void
+ OnGardenButtonDown() :...
DesertButton
- _desertButtonDown: bool
+ Update() : void
+ OnDesertButtonDown() :...
WinterButton
- _WinterButtonDown: bool
+ Update() : void
+ OnWinterButtonDown() :...
BackButton
- _backButtonDown: bool
+ play: GameObject
+ custom: GameObject
+ tutorial: GameObject
+ help: GameObject
+ back: GameObject
+ text: GameObject
+ desert: GameObject
+ winter: GameObject
+ garden: GameObject
+ Cur() : void
+ OnBackButtonDown() : v...

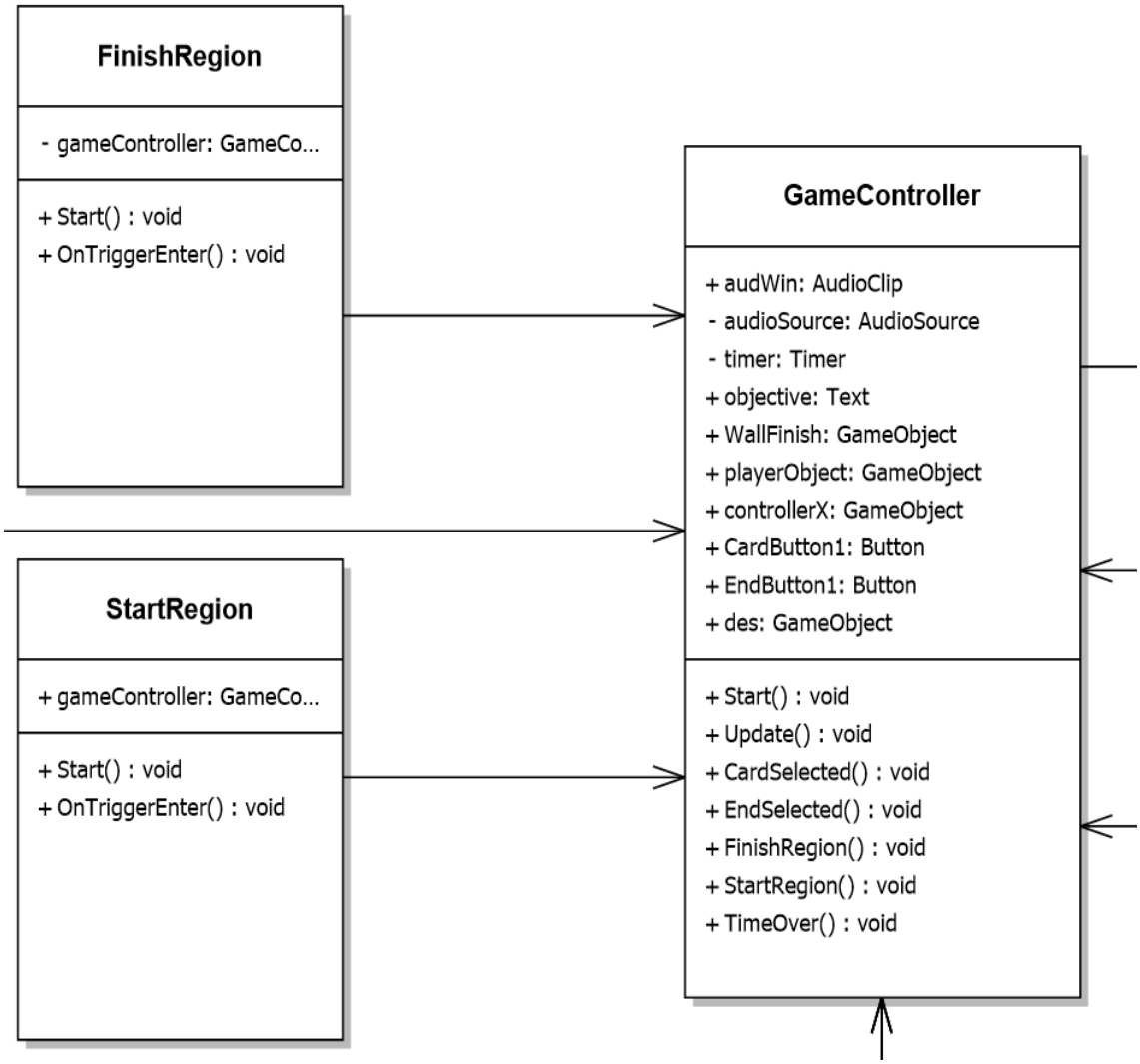
Class Diagram for Maze Spawn:



Public values can be introduced inside Unity Inspector when attached to a game object that spawns the maze at location.

Note: all Class Diagrams in the Appendices folder

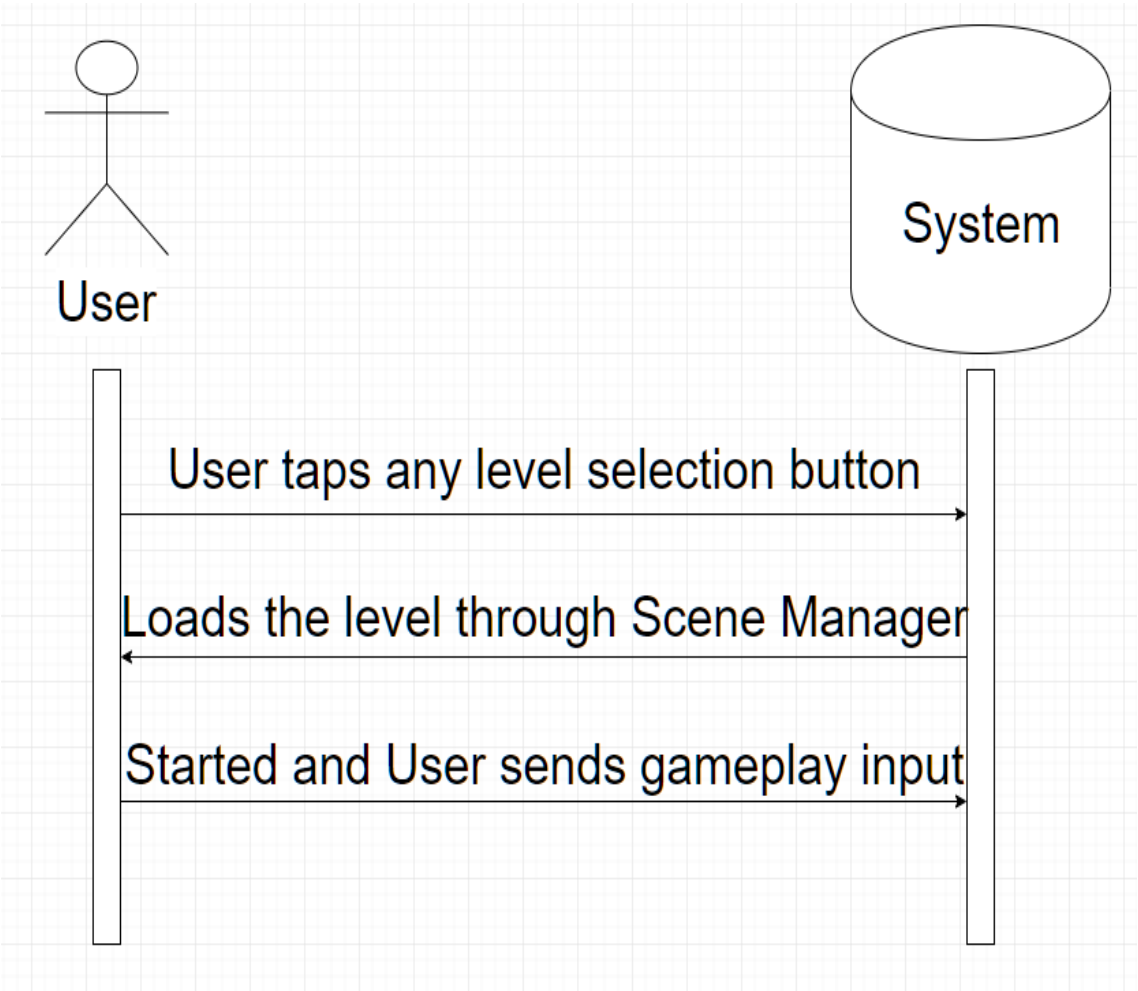
Class Diagram for Game Controller & Regions:



This is a very important part of our "objectives" system in the game levels, as both the Start and the Finish regions have functionalities in GameController class with FinishRegion() and StartRegion() methods. When the OnTriggerEnter() is used, the Game Controller is notified and the respective methods are started.

Note: all Class Diagrams in the Appendices folder

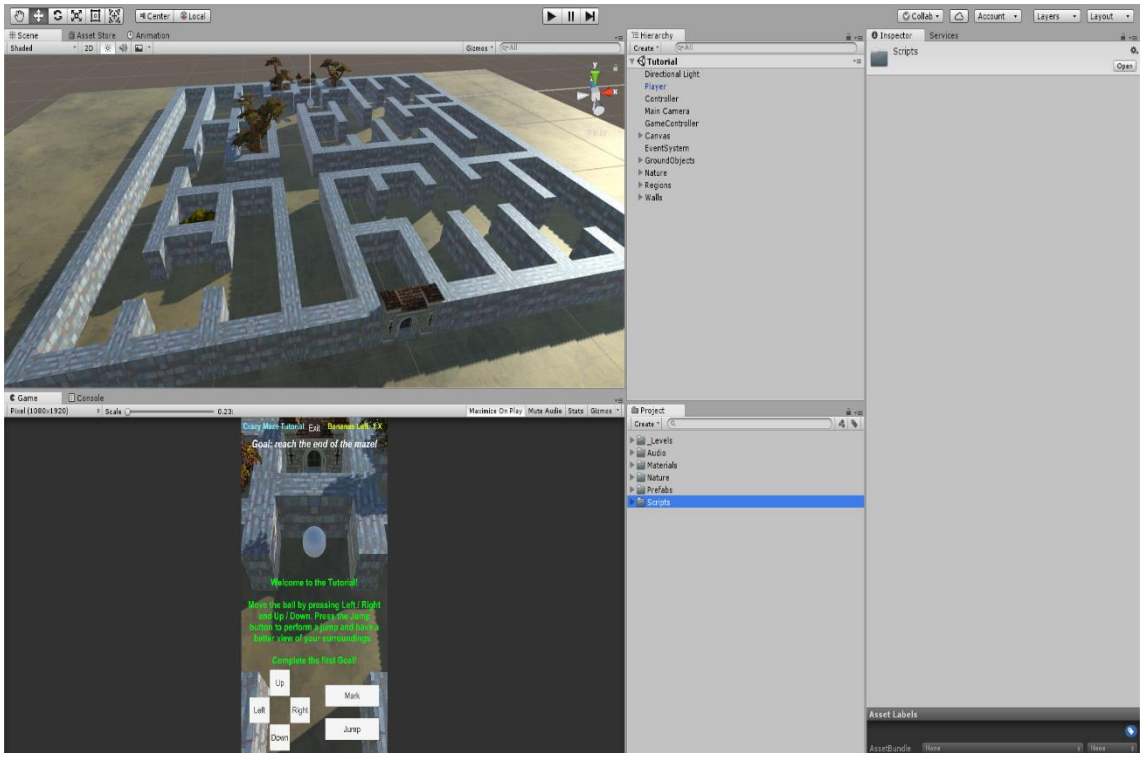
Sequence Diagram for a basic initiation of a game level with actors USER and SYSTEM:



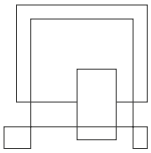
Notice that every time the USER chooses to enter a level by pressing the respective button, the Scene Manager loads the exact level requested and begins the scene. Pressing the Exit button activates the Scene Manager load again, this time sending the user to the Main Menu screen

5 Implementation

The implementation process was, in our group’s view, the most challenging, yet interesting part of our project. We had to balance gameplay aspects with innovative memory learning techniques in a moderate manner that would get the player more and more interested in the game. The code snippets that will be shown in this section are in C# programming language on MonoDevelop-Unity.



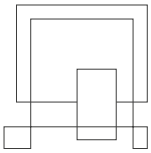
In the picture above, you can notice how we constructed our game scene with the assets we had and structured our Hierarchy and Project windows in a clean and efficient manner.



Implementation code for the Mark ability:

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class MarkButton : MonoBehaviour
{

    bool _markButtonDown;
    Vector3 startPos;
    public GameObject hazard;
    public float banana;
    public Text bananasText;


    // Update is called once per frame
    void Update ()
    {
        bananasText.text = ("Bananas Left: " + banana + " X");
        if (_markButtonDown)
        {
            if (banana != 0)
            {
                var player = GameObject.FindGameObjectWithTag ("Player");
                startPos = player.transform.position;
                Quaternion spawnRotation = Quaternion.identity;
                Instantiate (hazard, startPos, spawnRotation);
                banana = banana - 1;
            }

        }
    }

    public void OnMarkButtonDown (bool down)
    {
        _markButtonDown = down;
    }

}
```

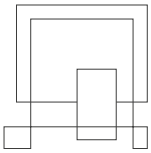
As it can be seen from the code, when the Player presses the Mark button, the bool that is checked in the IF statement from Update() method is down, which means that a GameObject clone is spawned at the location of the Player Object. The bananas left text is also updated accordingly.



Implementation code for the Countdown Timer:

```
30      // Update is called once per frame
31      void Update()
32      {
33
34          if (timeLeft <= 0)
35          {
36              StopCoroutine("LoseTime");
37              countdownText.text = "Out of time!";
38              countdownText.color = Color.red;
39              gameController.TimerOver ();
40          }
41      }
42
43      IEnumerator LoseTime()
44      {
45          while (true)
46          {
47              yield return new WaitForSeconds(1);
48              timeLeft--;
49              countdownText.text = ("Time Left: " + timeLeft + " sec");
50          }
51      }
52  }
```

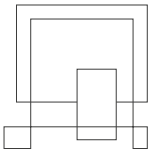
This code belongs to the Timer.cs class and has the purpose of counting down time and check if the timer has reached the number 0. The LoseTime() method is called from the Game Controller class when the player has to return to the start of the maze, and counts down the time left once every second, while also updating the UI to let the player know how much time he has left. The Update() method checks the IF statement for when the timer reaches 0. When that happens, the method from Game Controller called TimerOver gets triggered.



Implementation code for the player Controller:

```
using UnityEngine;
using System.Collections;

public class Controller : MonoBehaviour
{
    public Transform target;
    public float speed = 5;

    bool _rightButtonDown;
    bool _leftButtonDown;
    bool _upButtonDown;
    bool _downButtonDown;

    // Update is called once per frame
    void Update ()
    {
        if (_rightButtonDown) {
            float moveRight = (Time.deltaTime * speed);
            target.Translate (moveRight, 0, 0);
            target.transform.rotation = Quaternion.identity;
        }
        if (_leftButtonDown) {
            float moveLeft = (Time.deltaTime * speed);
            target.Translate (-moveLeft, 0, 0);
            target.transform.rotation = Quaternion.identity;
        }
        if (_upButtonDown) {
            float moveUp = (Time.deltaTime * speed);
            target.Translate (0, 0, moveUp);
            target.transform.rotation = Quaternion.identity;
        }
        if (_downButtonDown) {
            float moveDown = (Time.deltaTime * speed);
            target.Translate (0, 0, -moveDown);
            target.transform.rotation = Quaternion.identity;
        }
    }
```

This class has the Player object attached to the Transform target and is moved when the movement buttons are pressed. Each frame, the object’s position is changed according to its desired direction.

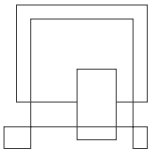


Implementation code for the Camera Controller

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class CameraController : MonoBehaviour {
5
6     public GameObject player;
7
8     private Vector3 offset;
9
10    void Start ()
11    {
12        offset = transform.position - player.transform.position;
13    }
14
15    void LateUpdate ()
16    {
17        transform.position = player.transform.position + offset;
18    }
19 }
```

In the code snippet above, the Main Camera in our scene is controlled through the CameraController script attached to it. At the start of the mission the camera offset will be calculated by the position of the camera object minus the position of the player object. Later on, the camera will follow the player object at the same distance and rotation at which it was set at the start of the level.

Note: All Classes & Scripts in Assets\Scripts folder.



6 Testing

Video games require a considerable amount of testing in order to eliminate bugs that are caused by a faulty code or bad Inspector management. One easy way to know if our code has any issues with the scene objects or vice versa was to check the Console window inside Unity, which allowed us to quickly find solutions.

Another way to test our game was to use the Unity Test Runner, a tool that tests your code in both Edit mode and Play mode. We used this method at the start of the project when we had to make sure that player object movements and camera movements are working properly.

```
[UnityTest]
public IEnumerator GameObject_WithRigidBody_WillBeAffectedByPhysics()
{
    var go = new GameObject();
    go.AddComponent<Rigidbody>();
    var originalPosition = go.transform.position.y;

    yield return new WaitForFixedUpdate();

    Assert.AreNotEqual(originalPosition, go.transform.position.y);
}
```

The most efficient way for us to identify bugs or code errors later on was to “simply” play test our game and try every single possible way to complete the game, either by winning or losing. The goal was to put our application to test in casual conditions.

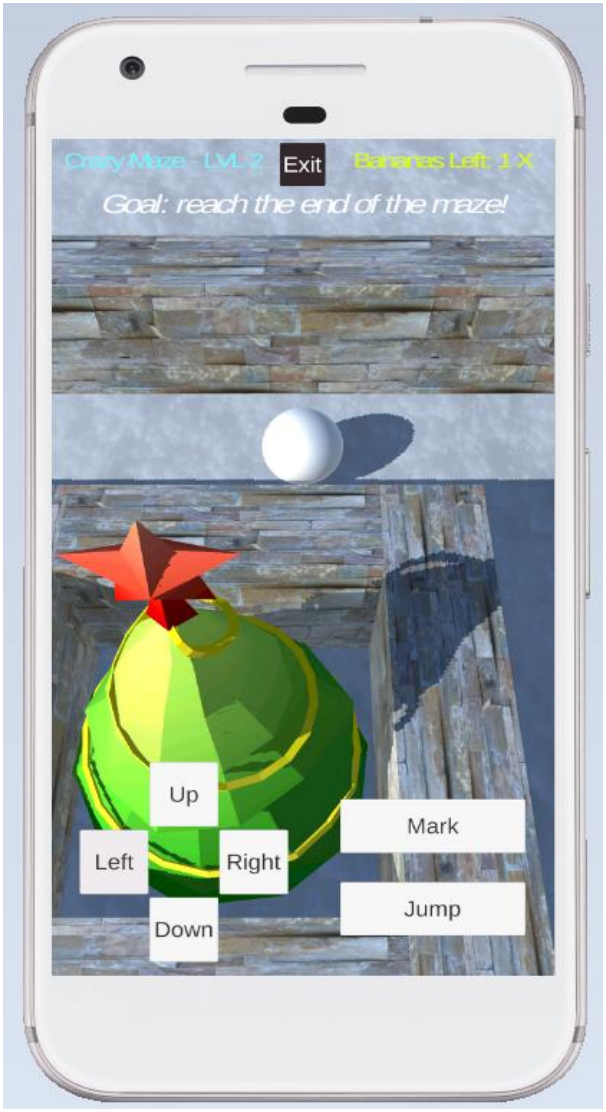
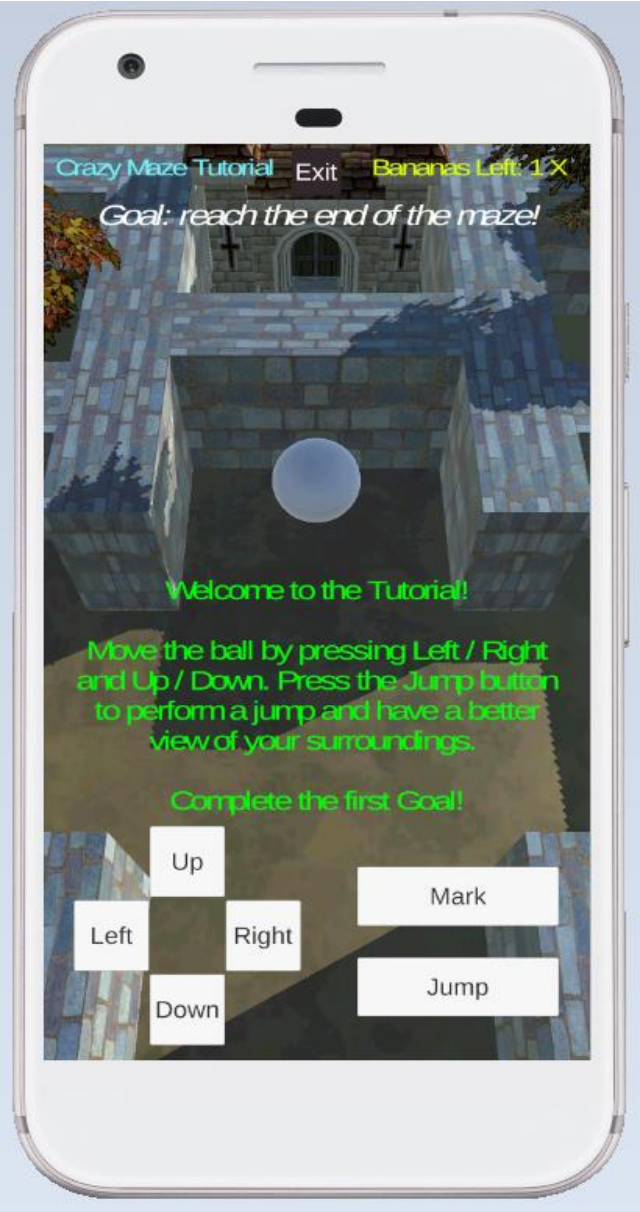


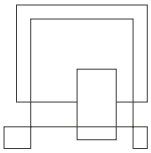
Image above is a screen shot taken during one of our play tests where we were identifying errors in the Winter Map level. The ball was unable to move, but this bug was later fixed.

In this screenshot, the respective buttons had no functionality when the user tapped them. This was due to an error in the Scene Manager loading (misspelled name of levels etc).

The Tutorial level was a good place for us to quickly test complex mechanics such as the Mark or Jump abilities.



We managed to fix and tweak most of our mechanics in order to favour memorization training in a way that wouldn’t turn the player away or make him bored after 1-2 completed levels. Android mobile compatibilities were also modified to suit the game for this platform during this testing session.

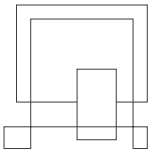


7 Results and Discussion

After evaluating and analysing the core functionalities of our game, together with the results of our testing and bug fixing, we can say that the group is pleased with the results. Not only it's the first mobile application of this scale that we have created, but we have also learned a lot of things regarding aspect ratio and optimisation for better Android performance.

The outcome of this project is an exciting and rewarding video game that serves the purpose of being a memorization training activity, especially for people of young ages or elders. The game keeps the user in control of the game, with each decision the user takes being vital in the completion of the game. Each goal has a meaning in the "road" to a better understanding of what is important to remember and what is of less importance.

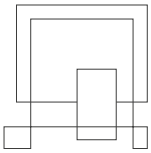
- ✓ The application must improve the user's ability to memorize something
- ✓ The application must be developed using Unity
- ✓ The application must be developed for phones/tablets
- ✓ The project must contain written documentation (Project & Process Reports)
- ✓ The project must contain relevant diagrams / UML representations



Taking each requirement set by our stakeholder Ensight Games and supervisors one by one, it can be said that our application and project as a whole fulfils every demand and necessity. In our view, the most important requirement for our game was to be a learning experience. Helping the user memorise natural and environmental elements to help him find his way through a maze, while at the same time being required to remember an item of his choice, proves to be an efficient way of increasing your short-term memory and train your brain to complete the same process in real life situations, such as being lost in a city or having to fulfil a multi-tasking activity which revolves around and requires memorization of specific elements.

The game's scalability is assured by the Custom Maze level, which generates a random maze every time the level is loaded. This will keep the player interested in the game even after finishing the tutorial level and pre-made levels. We believe that by also making the game user-friendly but challenging at the same time will give the player a sense of recognition and pride each time he successfully completes a level. Each mechanic and ability the player can use has been tested and balanced so that, in order to tick each goal and deploy the win condition, you must be capable enough to use everything in the game to your advantage.

For a better comprehension of our game, please check the User Guide that we have created for "Crazy Maze" inside the Appendices folder. It offers an in-depth guide to the game modes and controls available, as well as a description of the rules and goals of the game, win conditions etc.

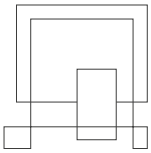


8 Project future

We understand that, as in any video game, there may be bugs or errors that we have not been able to discover yet. We have, though, fixed every bug that we have encountered and we are confident that the gameplay experience is as smooth and error-free as it can possibly be for a game like this. All in all, the best way to eliminate this issue entirely is to play-test the game and let the user experience the game as much as possible in order to "patch" and fix remaining bugs, as well as tweaking the game mechanics better and more efficiently.

From a technical point of view, we think that the choices we made code-wise and the assets we've selected were essential in creating a worthy experience for the player. If we had to start over again, we probably would go for the same design and type of game to fulfil the project's requirements. In our view, we should give more attention to documentation and reports, to explain our code and decision-making better etc.

As a group, we would have loved to implement more features, such as a Leaderboard that gets updated scores every time the player finishes a level by winning or losing. This was though, not possible due to the time limit. A fourth level was also something which we wanted to put on the board, but the scene was unfinished and not worthy enough to make it in the final build of the game. We, though, believe that the game, as it is, would be suitable for release and production at a professional level, after, of course, a period of intensive and fulfilling "beta testing.

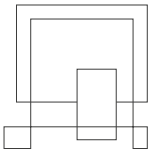


9 Conclusions

Having discussed and elaborated every part of analysis, design, implementation, and testing of our project, it can be said that we have managed to satisfy every requirement and necessity set at the start of the project period. A lot of work was put into making sure that every user interface element and game mechanic was properly tested and balanced to offer the best performance possible to our user.

Our main goal was to have a completed application that serves the purpose of helping the player / user memorize and understand how to use this stored memory to complete certain tasks within the game, like he would do in a real-life situation when put to the test. The game "Crazy Maze" is a refreshing and rewarding activity that does exactly that, and not only represents just a video game that entertains the player and just that. We wanted to make the player understand how to use surrounding elements and develop a higher spacial awareness. By playing and trying to complete the levels, the user can obtain such qualities and abilities that may be a little harder to acquire elsewhere.

Summing up the features and the usefulness of the application, the game will be a valuable asset in memorization training, as it offers a fun and rewarding activity that has fulfilled and delivered all the requirements set at the start of the Semester Project period.



10 Sources of information

Every source of information other than from our own work will be listed here. These may include book chapters, reports, patents, pdfs, standards, interviews, dissertations, conference proceedings and peer reviewed papers in scientific journals. Due to questions often asked about objectivity, newspaper articles, brochures and web addresses are often used sparingly as well.

Anon 2015. *The Art of Memorization - Memorise*. [online] Available at: <http://memorise.org/memory-training/art-memorization> [Accessed 12 May 2018].

Anon 2018. *Try Your Mind at a Memory Challenge*. [online] Available at: <http://www.readersdigest.ca/health/healthy-living/try-your-mind-memory-challenge/view-all/> [Accessed 13 May 2018].

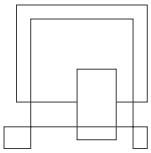
Anon 2018. *Unity - Scripting API*: [online] Available at: <https://docs.unity3d.com/ScriptReference/> [Accessed 16 May 2018].

Anon 2018. *Memorise - The Original Memory Gym*. [online] Available at: <http://memorise.org/> [Accessed 18 May 2018].
Mendeley.com, 2016. *Homepage | Mendeley*.

Unity Assets Store (Mendeley access denied)
https://assetstore.unity.com/

SGM PowerPoint Slides from Studienet Course Material

Project Report & Project Description Guidelines from Studienet



11 Appendices

The Appendices folder contains all the files that were not added in the Project / Process reports. This folder can be found within the "Crazy Maze" game hand-in .ZIP file. The contents found in the Appendices folder are the following:

- ✓ **Use Case Diagram**
- ✓ **Activity Diagrams**
- ✓ **Class Diagrams**
- ✓ **Use Case Descriptions**
- ✓ **Project Description**
- ✓ **References**
- ✓ **User Guide**
- ✓ **Group Contract**
- ✓ **Readme.txt**