Trust under emergency: A game involving Procedural Generation.

Adam Jennings

955770

Submitted to Swansea University in fulfilment

of the requirements for the Degree of Master of Science

Bachelor of Science



Department of Computer Science

Swansea University

January 14, 2020

# Declaration

This work has not been previously accepted in substance for any degree and is not being con- currently submitted in candidature for any degree.



Signed (candidate)

Date **26/04/21**

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.



Signed (candidate)

Date **26/04/21**

Statement 2

I hereby give my consent for my thesis, if accepted, to be made available for photocopying and inter-library loan, and for the title and summary to be made available to outside organisations.



Signed (candidate)

Date **26/04/21**

# 

# 

# 

# Abstract

The problem faced was a client scenario whereby the client tasked me with developing a first-person survival escape game where users will navigate through a procedurally generated map, pick up items and use those items to try and exit the map and finish the game. The client also asked for implementing a guidance robot with a failure feature where trust can be measured by the user asking for guidance from the robot and choosing to follow or not follow the suggestions.

Procedural Generation is a function or algorithm which adds randomness to an environment in order to have variation in different iterations and instances within a game or other. This is used frequently in mainstream video games however not in conjunction with an element measuring trust of a user towards a non-player character (NPC).

I began by creating a procedural generation algorithm using C# script alongside the Unity games engine software. I ran into problems such as ensuring that a good amount of variation was incorporated with each iteration of the game being launched, increasing optimisation, and ensuring all objectives were met within the given time frame. I bypassed some of these issues by restructuring the algorithm for increased optimisation however this shortened the time frame to focus on other aspects.

Due to time constraints, I decided to prioritise the game design and map generation to ensure a strong foundation to build on in future development.

# 

# Acknowledgements

A special thanks to Muneeb Ahmad who helped me get through to where I need to be. Thanks to all friends and family supporting me through difficult times.

# Table of Contents

[Declaration 2](#_Toc70426896)

[Abstract 3](#_Toc70426897)

[Acknowledgements 4](#_Toc70426898)

[Table of Contents 5](#_Toc70426899)

[Chapter 1 Introduction 8](#_Toc70426900)

[1.1 Motivation 8](#_Toc70426901)

[1.1.1 Objective 8](#_Toc70426902)

[1.2 Overview 9](#_Toc70426903)

[1.3 Outline of the project 10](#_Toc70426904)

[Chapter 2 Background 10](#_Toc70426905)

[2.1 Related Work 10](#_Toc70426906)

[2.2 Game Programming with Unity 10](#_Toc70426907)

[2.3 Procedural Generation (PG) 10](#_Toc70426908)

[2.4 Games to measure trust. 11](#_Toc70426909)

[Chapter 3 Specification 12](#_Toc70426910)

[3.1 Requirement Analysis 12](#_Toc70426911)

[3.2 Language Choices 13](#_Toc70426912)

[3.3 Technical Difficulties 13](#_Toc70426913)

[3.4 Initial Design 13](#_Toc70426914)

[3.4.1 Gameplay 13](#_Toc70426915)

[3.4.2 Player Stats 13](#_Toc70426916)

[3.4.3 Controls 14](#_Toc70426917)

[3.4.4 Art 14](#_Toc70426918)

[3.4.5 Items and Crafting 14](#_Toc70426919)

[3.4.6 Computer 15](#_Toc70426920)

[Chapter 4 Planning the project 15](#_Toc70426921)

[4.1 Detailed steps & milestones 15](#_Toc70426922)

[4.2 Timeline 16](#_Toc70426923)

[4.3 Development model 16](#_Toc70426924)

[4.4 Risk analysis 17](#_Toc70426925)

[4.5 Functional and Non-functional requirements 18](#_Toc70426926)

[Chapter 5 Implementations 19](#_Toc70426927)

[5.1 Platform and License 19](#_Toc70426928)

[5.2 Overview of System and Gameplay 19](#_Toc70426929)

[5.3 Main Game Implementation 20](#_Toc70426930)

[5.3.1 Overview 20](#_Toc70426931)

[5.3.2 Design of the Game 20](#_Toc70426932)

[5.3.3 Procedurally Generating the map 20](#_Toc70426933)

[5.3.4 Collectables 24](#_Toc70426934)

[5.3.5 Robot and Robot Accuracy 25](#_Toc70426935)

[5.3.6 Timer/Saving to a file 26](#_Toc70426936)

[Chapter 6 Future Work 27](#_Toc70426937)

[6.1 Optimisation 27](#_Toc70426938)

[6.2 Time Constraints 28](#_Toc70426939)

[6.3 Next Steps 28](#_Toc70426940)

[6.4 Conclusion 28](#_Toc70426941)

[Bibliography 30](#_Toc70426942)

# Chapter 1 Introduction

This document is intended both as a thesis template and a written tutorial on typesetting a professional looking academic document. The style of the template is designed to mimic an equivalent LaTeX document template that is commonly used for within the Computer Vision and Visual Analytics group here at Swansea. This LaTeX template is itself based on a LaTeX template named Custard.

## 1.1 Motivation

Procedural generation (PG) refers to the algorithmic creation of content. It allows content to be generated automatically and can therefore greatly reduce the increasing workload of programmers. Some of these methods have gradually become more common practice in the video game industry but are limited to specific context and game elements. [2]

The purpose of this is to automatically generate a large amount of content in a short amount of time. Depending on how PG is implemented, there could be the advantage of having a smaller file size with larger amounts of content and randomness for less predictable gameplay.

I have always had an interest in game development and have been fascinated with procedural generation within a game environment to allow users to have multiple iterations within the game without coming to the same environment or conclusion. This can make a video game near infinitely replay able without having to play the same specific requirements twice.

### 1.1.1 Objective

With this project, I aim to create a game based on a customer's demands on a survival game with a feature of an entity deciding to choose certain options allowing the player to trust or distrust the actions that the entity will give. Features in this game environment will be procedurally generated such as the size and shape of the building, the amount, and placements of assets like office supplies and keys to open doors, and resources such as collectibles. This will be done in C# language primarily using the free unity engine because I have had previous experience with this language and software. The customer will use this game for a research background to test users if they are willing to trust a non-player character in an emergency. Some freedom in this game can be more towards the players ability which is more explained in the specification. The customer’s requirements are as follows:

*The general idea for the game is to create an environment that represents a building design or any other similar setup. As the game starts, the player finds itself inside the building. The goal for the player is to vacate the building in a minimum amount of time say (300 seconds). The player is accompanied by the agent or a robot to help it get out of the building. The player is presented with two options 1) to ask the robot for help to vacate the building or do it by himself via following a map. Here, for the robot, we implement a failure feature. This means the robot can provide help where it is correct 50% of the time and where it is correct 97% of the time.  Further, the robot uses two modes to provide instructions to help the player, in one, it just tells the player what to do and in another, it tells the player about what to do after giving an explanation. At the start of the game before launch, the game should have a menu to set all these parameters such as failure rate, explanation, or none. The overall idea is to see how much a user (player) trusts the robot in a different situation in the game?  For instance, does the player trust the robot more when the robot is 97% correct and provides an explanation vs when the robot is 50% correct and provides an explanation?*

Another aim of this project is to test if a user is more trusting of a Non-Player Character (NPC) depending on the failure chance and explanation. I Predict that the user will be a lot more trusting given an explanation regardless of its failure feature, but it would be a linear increase of trust given the failure chance being reduced. This will also find out what aspects of a given instruction is more trustworthy or not. Whether that be the explanation or the chance of the NPC being correct.

.

## 1.2 Overview

The remainder of chapter 1 outlines the document structure and the key contribution of this work is organized as follows. Chapter 2 reviews techniques for finding and properly citing external resources from the academic literature and online. In chapter 3 we show examples of how to typeset different types of content, such as internal references, figures, code listings, and tables. And lastly in chapter 4 we summarize the main contributions and key points to take away from this template.

## 

## 1.3 Outline of the project

The main idea of this project is to have a scenario where a customer has asked for a specific game which has certain features. In this case it is to make a survival game with an element of trust in the instance of an emergency. The specific demands are that it must be in a 3D setting and that it must be complete and ready to be given to the customer in the set time lasting the academic year.

# Chapter 2 Background

## 2.1 Related Work

Some video games use a trust system from the Non-Player Character (NPC) to the player. “The Thing” which was a 2002 third person shooter survival horror video game in which they had a “Fear/Trust system”. This was when an NPC would follow the players orders and join them in combat based on certain variables such as if the player gives weapons, ammunition or heals the NPC’s. This will be similar to the game that I will be making as there is a trust feature however it will be based on the player trusting the NPC. The NPC will only be able to gain the player's trust by giving the player an explanation (provided that is toggled on in the menu selection before the game starts).

## 2.2 Game Programming with Unity

There are many different video games which use the Unity engine. Games like the one I intend to make which has used Unity are Slender, Husk, Krampus and many more.

Unity provides many features that help with video game design such as materials, models, mesh, animation importing and input handling. These features will help allow to create a fully working video game in a shorter time frame. With this, it will allow me to go straight to implementing more finer detailed core elements in the game in the back end without dealing with most of the front-end graphics. There are a vast number of tutorials and community supporting members that use unity to help with any problems that need solving as well as an Asset Store to allow me to not have to spend as much time on artistic work within the game.

## 2.3 Procedural Generation (PG)

PG has been a very common factor in cult classic games such as Minecraft (random maps and items inside when you create a new world) and Binding of Isaac (every level has its own map design that is unique). To create a modern game with PG is nothing new. The most common use of procedural generation in modern day use in video games is to be used in looting systems on quest-driven games. Such as Role-Playing Games (RPG) and Massive Multiplayer Online games (MMO). Loot such as weapons or armour can be generated for the player given their character’s level, the level of the given quest, the player’s performance on the quest and other random chance features. An example can be *Borderlands* (2009) which uses PG to create over a million unique guns and other equipment. [4]

To design video games, it's crucial to consider video game design over the past few decades. Most video games as we know them today would have been heavily influenced by and built upon the designs of the past, and so it is important to research into this.

One of the first procedural generation-based games is *Beneath AppleManor* (1978) - a roguelike 2-dimensional game where a warrior would travel through a dungeon. This game would procedurally generate the dungeon to create a different map experience every iteration the user played. Most roguelike, and games with roguelike concepts, allow the development of complex gameplay without having to spend excess time on creating a game’s world environment. [2]

PG was more modernised in games with a game called *.kkrieger*(2004) - a german first-person-shooter which only had a 96 kilobyte executable file which could generate hundreds of megabytes once run worth of 3-dimensional assets and texture data. [1]

///Usually you would not put the URL of the resource you are citing directly in the text like is done previously in section 2.2. The citation for the resource [4] is sufficient to reference it within the text given that full details of its location are then kept neatly within the bibliography at the end of the document.

In normal usage the purpose of a citation is not to direct the reader away from your thesis, but to justify and back up assertions you are making about the state of the domain. If a reader questions your assertions then they can follow the rabbit hole of papers which will likely also make and justify assertions with even earlier papers from the literature.

In the above case the intention is for the reader of this template to actually go to that resource and read what it has to say directly. The link is therefore shown clearly within the main text to indicate that the reader should visit it. This as opposed to wanting the reader to purely acknowledge that the facts which are within the resource legitimize the points made in this document, in which case a simple inline citation is the best way to back up your assertions.

## 2.4 Games to measure trust.

Similar games which measure trust are primarily trust-tested games within other users in an online multiplayer setting. These are quite common and simple designs where one or more of the users are trying to kill all other users while users who are innocent must find out who are the killers while also completing objectives. Mainstream games with this common objective are *Deceit (2017)* and *Among Us (2017).*

However, it is very difficult to come across any instances of games that have a user measuring the trust of a Non-Player Character (NPC) showing that this idea is unique which may have struggles with any form of research into constructing a game such as this.

# Chapter 3 Specification

## 3.1 Requirement Analysis

In terms of hardware requirements, the requirements to use the Unity engine will be enough to be able to run the game setting. As this game is designed purely on research, there is no need to have high quality graphics or any features which would cause a lot of storage space or computer power. Ideally, I will need to be using Windows as I am more familiar with that operating system however once the game is complete and ready to be released, it should be able to run in any main operating system with Apple and Linux to name as examples. The specifications are as follows:

|  |
| --- |
|  |
| Figure 3.1: screenshot from Unity manual website: <https://docs.unity3d.com/Manual/system-requirements.html> |

## 3.2 Language Choices

Choosing C# in the Unity engine is ideal because C# is a much more versatile and powerful programming language in comparison to other languages that generally create a Unity game, such as Java which does not work as well in developing scene structures in the unity engine.

## 3.3 Technical Difficulties

Although possible, there will be a lot of difficulty in transferring all work from current device to a more portable device such as a laptop. This can be a very severe technical difficulty as I will be using my main desktop computer to create this game although during breaks like the winter holidays, it would not be ideal to transport an entire desktop to different places. To mitigate this, all files that are necessary can be placed on a cloud which can be accessed from any device provided they have the same user login in the Unity Engine.

## 3.4 Initial Design

With a survival game, there are many features that have to be added to the user. Because of this, this section will be broken down into sub sections.

### 3.4.1 Gameplay

1. Player Roles - There are most commonly 3 main roles that a user has to complete in order to progress which will be a core part in the user’s gameplay.
   1. Gathering - Acquiring resources for survival such as picking up keys to go through doors, a crowbar to break through materials etc.
   2. Exploring - Satisfying potential objectives by going further around the map for more resources as well as being able to vacate the building.
   3. Listening - Paying attention to the entity to trust the robot and follow the orders.

1. Player Inventory - With collectibles in the game, there should be the ability to carry more than one item in order to help the player progress. For example, if the user were carrying both a fire extinguisher and a key, they could alternate between what is in their current hand to use (with the fire extinguisher, they can put fires out. With the key, they can open locked doors)

### 3.4.2 Player Stats

In a survival game, player states are very important to encourage the player to progress through the program in order to survive. This section describes the main stat the player will have and how a player can gain or lose each stat. Typically with most survival games, there are player stats like health, stamina, hunger, thirst, and in some rare cases sanity. In this case however, there is no need to add any other state than Health because there is no need to hunger, thirst or sanity when the situation will almost always be to simply vacate the building. Having a difference in health value will affect the user experience.

Health will decrease within the player stats by several different ways; the health can decrease by taking damage from entering a dangerous zone which can be an area on fire, a gas leak or any other event which might require evacuation. Once there is no more health value (if the health bar reads zero) the player will die. Health can increase over time provided they are in a safe zone and the player does not take any damage.

### 3.4.3 Controls

With any form of video game, there is always a general consensus to have very familiar controls to allow the users to have ease of motion and not have to relearn different controls and have an irregular experience as the user. Efficiency and effectiveness of gameplay were taken to consideration when designing a control scheme to make the user feel normal about them. A clear example is to use the motion keys with the “W,A,S,D” keys and to use the mouse for other actions to have a more relaxed feeling. The setup is as follows:

1. Movement

* W - move forward
* A - move left
* S - move backwards
* D - move right

1. Actions

* Left Click - Use item held/ Attack
* Right Click - Interact (Pick up, open, close, etc)
* Tab - Open/Close Inventory spaces (can Left click to equip items in inventory. Can also hold and drag items together in order to craft a new item)

### 3.4.4 Art

As there are no requirements on specific artistic designs of the game as well as time constraints, most artistic assets will be obtained in Unity’s standard free assets in the Asset Store. The Asset Store is a community where other Unity users distribute their art from various different price points. This store will be very beneficial for me as the programmer to fully focus on all back-end creating of a functioning game without having to worry at all about aesthetics or visual designs.

### 3.4.5 Items and Crafting

The crafting and constructing mechanic are a staple part of any survival game, it allows the player to progress and evolve their character to set themselves more challenging and ambitious objectives. This section will go into detail on the design decisions made when creating different items that can be craft able and used in order for the player to vacate the building. This section will also mention the placements of items which are available for a player to craft and then use.

The player's inventory will be a main way for the user to pick up and carry items within the game that will be necessary to leave the building. The inventory will have twelve slots meaning that the player can only carry twelve items at a time which should be more than plenty for the ability to vacate the building. The player will have the option to equip any item in their inventory by simply opening up the inventory window and using left click on the desired item.

### 3.4.6 Computer

As said in the game specification, there are a set amount of requirements that the computer must do if allowed from the menu selection. These roles are as follows:

1. Accuracy of decision - This should be a slider in the menu for the user to be able to allow the level of accuracy of the computer to decide the right thing to do. The lower the accuracy, the less likely that it suggests the best option to vacate the building
2. Description - A toggle button within the menu that will let the computer give a description for their choice and reasoning behind it.
3. Communication with the user - In order for the user to be able to trust the computer, there needs to be a way that the computer can deliver the information to the player. This can be portrayed by having a simple chat screen that the user can read. It can also be portrayed as if the player is receiving text messages within a phone environment to create a more immersive experience within a player.

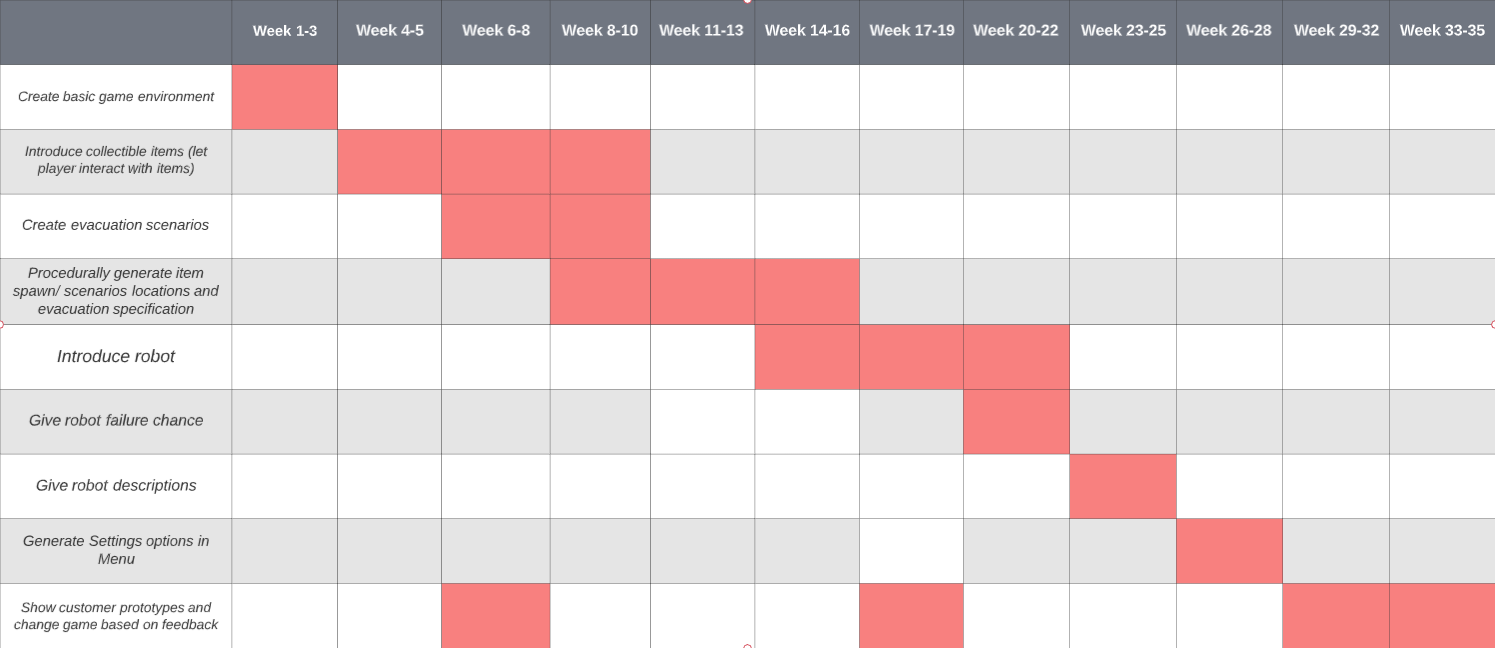
The computer must be implemented into this game in order to have any research purposes and to meet the customers’ requirements.

# Chapter 4 Planning the project

## 4.1 Detailed steps & milestones

Many steps need to be taken in developing this video game. The first crucial and basic step is to just create a game environment using unity where you can have a player moving around a setting and importing assets which a player can walk through or not walk through depending on the object. The next step can be to introduce collectible items which the player can use to interact with the objects already in the game environment. Then, you can make a procedurally generated algorithm to create a random location for the collectible items. The next main steps would be to create the robot or non-player character with their failure features, descriptions and having those options to determine their chance of failure or providing a description on the main menu screen.

## 4.2 Timeline

The best way to understand and to create a good model for me as the programmer to stick to is to visualise the tasks and the time that needs to be taken with each task. A simple Gantt chart will be more than enough to give me the ability to organise what steps need to be taken and at what time. The Gantt chart can be seen below:

As seen by the chart above, the task to show customer prototypes comes in increments. This is to show that I will be using the spiral model for development and will continuously show the customer the product which needs to satisfy their needs.

## 4.3 Development model

With the ability to be in constant contact with the customer, it would seem silly not to go for a more agile approach which can produce more prototypes to ask for customer feedback. The spiral model is a more familiar development model that I have used multiple times where I will create multiple iterations of the video game and once each cycle is complete, it can be reviewed to the customer to see what aspects they wish to change or improve. This is also helpful as with a longer time frame, the customer may change their mind on some aspects that they wish to have or be more lenient to a different idea which I can provide more effectively. The spiral model is also beneficial as with a set time frame as this, if the video game was to not be done in time, there will still be previous prototypes which can be given rather than not having a functioning video game at all which would fail the customer.

## 4.4 Risk analysis

There can be multiple risks when designing and creating a game from scratch by one person. Most game companies that range in sizes will always have a minimum of 5 members working on their own part in a game. With this limit to only 1 member there can be time constraints to create such a large video game idea. Below is a table of potential factors that can affect the work and ability to mitigate the issues that may arise.

|  |  |  |  |
| --- | --- | --- | --- |
| **Problem** | **Severity (1-10, 10 being very Severe)** | **Minimalization (reducing chance of problem)** | **Mitigation (reducing severity of problem)** |
| All files are lost | 10 | Saving the work regularly in multiple ways i.e in a current physical hard drive or allowing it on a cloud-based file system like Google Drive | In order to stop this from happening, files and hard drives must be backed up on a regular basis to be able to start in an earlier version of the game so not as many files are lost |
| Running out of time to complete work | 7 | To be able to have a set timeline and to be vigilant with allowing a set amount of time on each aspect in the game prioritising more serious elements in the game such as the main features that the customer wants | If there is no more time to make any changes to the game, there may be a possibility that the customer may wish to postpone the deadline in order to have a properly functioning game for the study. Also, there could be update releases which can give an initial release of the game but to update and fix any issues found once used by the customer. This is a very common feature in recent video games where they have an issue with the game after release which they update to fix later. |
| Being behind schedule according to the timeline set | 4 | Keep addressing to the timeline and ensure that you are always on track | Depending on the severity of the case(i.e there is more than one day behind schedule or having to spend more time on one specific issue as it is important to the structure of the game) you could either give yourself more time to focus on the work that needs to be done, stop being stuck on one objective and attempt to do other parts of the game, or ask the customer for more time to make the game more functional. |

## 4.5 Functional and Non-functional requirements

In order to measure trust in a PG environment, there needs to be a set of constants which are measured. This is required for keeping a degree of consistency to ensure the data collected is relevant to the objectives of the project. The functional and non-functional requirements are as follows:

|  |  |
| --- | --- |
| **Functional** | **Non-functional** |
| Users should be able to get from the start point to the end point(through doors, moving the player around, etc.) | A first person player controller is implemented which makes users move their cursor to look around as well as W,A,S,D to move around |
| Users should be able to ask for help and are hinted to then have a NPC assist them to try and find collectibles or an exit. | When a button is pressed, a pathfinding algorithm checks the way to the end point and then depending on its failure feature, shows a UI saying either the right door to go to or a random other door which the user will then choose which way to go |
| Map should be procedurally generated with the end point being very different with every iteration. | Procedurally generated the map setting and removing any doors which lead to nowhere |
| Saves data of the run they did with relevant data to measure the trust | Once the exit box collider is triggered, all data collected within the game is stored in a text file in the format of (Time taken - NPC failure rate - number of times trusted) |

# Chapter 5 Implementations

All code implemented in this project can be seen at: <https://github.com/AdamJ335/Dissertation>

Any Unity built-in elements used can be found at:

<https://docs.unity3d.com/2019.4/Documentation/Manual/index.html>

## 5.1 Platform and License

The following implementations were used to make the project in this thesis: Unity (2019.4.21f1), C# (7.3), and Visual Studio Code (1.55.2)]

## 5.2 Overview of System and Gameplay

This game is a 3D First-Person platform maze which contains a robot that follows the user around the map. The map is procedurally generated with set parameters to have a roughly set size and design of the maze in which the player can traverse around. The goal is to find the end point within a maze in which a robot would also find the end point using the A-star pathfinding and when the user asks for a hint from the robot, it will then tell the user either the correct way to traverse or the incorrect way to traverse depending on the failure rate of the robot.

There will be a menu interface at the start of the game which would allow the user to play/exit the application as well as setting the failure rate of the robot in the options menu. Using the basic unity UI package as well as SceneManagement helped in making the imaging of the menu and loading from the menu scene to the game scene.

A timer is implemented and shown in the user interface throughout the entire traversal in the maze. The timer then is stopped once the user exits through the end room and is then saved onto a text file along with the robot failure rate and the number of times the user trusted the robot when they asked for hints. This was implemented using the unity built in function Time.deltaTime with a counter incrementing by deltaTime each frame in the game.

Item Spawn points are set randomly within the map making the user not be able to memorise the location of the collectables. Using the Unity Gizmos package will allow me to ensure that collectables will spawn randomly within a drawn box.

## 5.3 Main Game Implementation

### 5.3.1 Overview

Unity is a free games development platform which allows quick and easy game creation. It works by letting users create game objects that can have colliders, can attach sprites and can have personalised behaviours by attaching scripts written in C#. With all these features, it helps to generate games or similar game related items with ease and in a fraction of the time taken if created from scratch.

### 5.3.2 Design of the Game

With the customer requirements in mind, I knew that they wanted to create a game environment with the design of a building setting. Using the unity asset store to get some free assets would assist me greatly rather than generating all of these assets myself. I used Snaps Prototype Office Assets to generate a basic office room (<https://assetstore.unity.com/packages/3d/environments/snaps-prototype-office-137490>) which helped me generate a visually satisfying environment within a limited time frame.

### 5.3.3 Procedurally Generating the map

Creating the initial design of PG, I considered creating a 4 room wide by 4 room long environment in which a critical path is made between the start room and the final exit room then filling the rest of the grid with dummy rooms (irrelevant rooms which some collectibles could be spawned as well as rooms which the NPC would tell the user to go to if the failure rate is high). I did this by having the start room be in a constant position which then random room types would be spawned. These random room types are as follows:

|  |  |
| --- | --- |
|  |  |
| 1. LR(Left-Right): this room would only have entrances through the left and the right of the room. | 1. LRT(Left-Right-Top): Entrances on the Left, Right, and above the room |
|  |  |
| ALL: Entrances in all directions. Top, Bottom, Left, and Right. | 1. LRD(Left-Right-Down): Entrances in the Bottom, Left, and Right. |

Initially, there was the start room, and then from a random direction, another room would then be instantiated in the direction determined by using the Random() built in function in C# and Unity. A room type was also randomly instantiated by using a list of room types and further using the Random() function to then grab a random element in the list. This came across the first problem however, where some rooms would have blocks covering the entrances which would sometimes limit the user from reaching the end point.

To fix this, I set a game object with a collider checking if there was another game object in the way. In this case, it was a wall check and seeing if it was there or not. Once a room was instantiated, it would then check based off the previous room if there is an opening allowing the player to walk through to the other room. If there was not, then remove the current room trying to instantiate and repeat instantiating a different type of room and checking if there was an opening until there was. This ensured that there was a clear critical path and all rooms instantiated would then be able to allow the player to move to the final exit room.

|  |
| --- |
|  |
| Figure 5.1: Initial procedural generation of a map. Critical path from start point to end point established. Very slow to initialize and user can go away from critical path. Furthermore, all within a maximum of 4 rooms by 4 rooms block which limits the possibilities of procedurally generating the path. |

Once the critical path was created, there were still empty points within the grid setting which needed to be filled in with random rooms so that there was no way that a user could stray away from the map set out and break the game. When trying to develop this, a series of bugs and issues started to happen within the game. Furthermore, the role of procedural generation was limited to having a 4x4 room grid which made implementing procedural generation useless as the chances of having the same exact map was very high. With all these factors in play, I decided to rethink the approach of map generation using PG.

With the Second approach, I started with simply making a room with no doors and spawning rooms around it relative to the start position placed at the origin. This was done with spawning the start room then creating a function, called LoadRoom() taking the parameters of the room type and the positioning of where the room would be loaded, which instantiated rooms based on the positioning it wanted. To make these numbers more practical, I would use the factor of the size of the room so whenever I called the function and wanted to place a room next to the original starter room, I could set the positioning to 1 or -1. This was a strong start but much more had to be done in order to create a way for the user to go to each room and also generating a random amount of rooms in random locations.

I then focussed on creating a script which would then randomly make rooms and placing them randomly next to already instantiated rooms. This was done with a function SpawnRooms() which took an IEnumerable of type Vector2Int of a crawler which would instantiate from the start and carry on in random directions the rooms that needed to be made. A crawler is an element of coding that determines the amount of procedurally generated pathways based on the origin (in this case, the start room). This was also a public variable and was set determining the number of crawlers was checking if there was enough variation within the map layout. With 2 crawlers, 2 procedurally generated pathways were made going different directions from the start. By personal preference, 2 crawlers seemed like there was enough variation for procedural generation to take place rather than more or less crawlers which with more than 2 crawlers started to more commonly be like a grid and less of a path and with just 1 crawler just made one critical path which the user can easily find the end point. Upon using this method, I realised that this was more efficient than the previous attempt as the first attempt would take roughly 5 seconds to load just the critical path whereas this attempt would take roughly 2-3 seconds to generate the entire

Now that rooms were being instantiated procedurally and there was enough variation with each iteration, the next step was to introduce doors which the user can go through. To begin with, I added doors to the general rooms, with doors on all different sides. These doors were 3d game object cubes with a box collider which would then when walked through by the player, it would transform them through the door relative to what type of door it is. For Example, if a user wants to go to the room on the right, once a user walks into the door, an OnCollisionTrigger() function is then checking if the door is a type “right” and if it is, then transform the players position positively on the z axis (in this game setting, this transforms as moving right). This was hopeful however I came to an issue where once the map is created, there are rooms going in every direction, even to outside the drawn map.

To sort this, I created a void function called RemoveUselessDoors(), which used a similar collider checker as used in the first iteration of development, checking if there was a room on the other side of each door. If there wasn't, then remove the door game object which would lead the player to go outside the map. This method was only called once the entire map had been drawn because if it was called during map creation, there would have been a series of rooms with no doors being able to allow the user to go through and onto the rooms leading to the end room. Now that the procedural generation of the map is created as well as doors leading the user to only within the drawn map, the final stage for map generation was to have an exit room where the user has to go to complete the run of the user.

In order to instantiate the final room(exit room) I created a class which was a controller class for all rooms(RoomController). This controller was in charge of all instantiations and locations of every room. In this class, a function is called to SpawnEndRoom(). This class sets a timer to wait half a second then checks if there are no more rooms to be instantiated. Once there are no more rooms and the map is already created, it grabs a random already instantiated room, removes it, then instantiates the exit room in the exact same position that the random room was removed. Once all of this has occurred, RemoveUselessDoors() is called to ensure that the exit room also has no doors leading to nowhere.

|  |
| --- |
|  |
| Figure 5.2: Second and Final iteration of Procedurally generating path. Creates a critical path with a few random rooms. Doors only lead to other rooms and user cannot leave the map drawn. Good variation of map generation and runs in less than 2 seconds. Opposed to Figure 5.1 which took roughly 10 seconds to create. |

This finalised the map generation. There is now procedural generation used in developing the map, there is no way that a user can leave the map, and there is a start/end point. I am happy with how this was implemented however iterating more than once made me use up a lot of valuable time which was already limited.

### 5.3.4 Collectables

To begin with, I decided that there would be collectables which were required to collect from the user and once they have all the collectables, they would then be able to leave in the exit room. I started creating a Gizmo which would be the borders of where a collectable would be able to be spawned(using instantiate()). Using a prefab of a key in the unity asset store would be used as my collectible, and once the user walks onto it, the gameobject would be destroyed and therefore being collected. Although this is already suitable and the main functionality is already implemented, there should be more indication to the user that an item has been picked up. This can be done using sound and adding a User Interface element.

In order to incorporate sound, a class CollectKey which is responsible for when the box collider is triggered by the user, the key will disappear (destroying the game object) and sets a boolean check “isKeyCollected” to be true. A Unity built-in class AudioSource has a Play() function which then plays an mp3 sound (I recorded the sound of keys falling on a hard surface then added the sound to my unity project) which is called once the user has collided with the collectable. This helped the user understand using an audio queue however, there should be some update in a visual queue as the user might not know how many items need to be collected at any given time. A user interface element has to be implemented.

Using Unity UI Text, I anchored some black text on the top left of the users screen (ensuring that no matter what aspect ratio the user plays in, it will always be on the top left of the screen). This would read *“KEYS LEFT: X*” with X being an integer variable which started from a number representing how many collectables were in the game environment and decrementing by 1 for each item collected. Once the value was at 0, it would indicate that the user has nothing else to collect and could therefore proceed to the exit of the map.

### 5.3.5 Robot and Robot Accuracy

With the map generation and the Non-Player Character(NPC) being the main features within this project. A lot of time has to be spent on both of these aspects. Furthermore, they are very important to be implemented well enough to not have any bugs and no corners can be cut in terms of design and implementation. Due to time constraints, I was unable to complete this element of the game to a suitable degree. This will be elaborate further in section 6 where I will outline my objectives set for the future of this project.

Had I had more time, I would have implemented the robot element using the A\* algorithm to locate the exit room, using a UI element which is activated by the user to tell them where they need to go from the robots calculations, and having a failure feature. Each element will be explained in detail below:

**A\* Algorithm**

The A\* searching algorithm is a shortest path algorithm which builds on the principles of Dijkstra’s shortest path algorithm to provide a faster solution when trying to find the shortest path between 2 nodes [5]. In a video game setting, this algorithm is very common. In a game such as Age of Empires, a classic real-time strategy game, it uses grids to represent map locations each with their own node. To move a unit of characters(for example, a military unit) the A\* algorithm is applied in order for the unit to be sent to the correct place ordered by the user. Not only is the A\* algorithm applicable to moving game objects within a map, but with implementing as enemies chasing a player around but in my case, having a pathfinder in the map implemented to the game object that is the Robot. I would address the start room and the exit room as start and end nodes respectfully and to have every other room as a transfer node which the algorithm can move to these nodes in order to calculate the location of the end node and the path needed to traverse. With a path generated, there would be a list on which direction the user will have to go. If the next node positioning is positive on the z axis, then it's indicated that the next room to go is right(negative meaning left) and positive on the x axis indicating the next room is above(negative meaning down). This list then is updated if the user has gone to another room which will not be in the correct critical path by calling the algorithm with the start node being at the room the user is currently in. A user should be able to press a button, in this case “H”, in order to show on their screen information determined by this algorithm

**User Interface**

With the user pressing the “H” key, there should be a text pop-up appearing on the right hand side of the screen which would have a suggestion text generically *“I suggest you take the X door”* with X being the door leading to the next node in the traversal to the exit node. This pop-up would only last for a total amount of 5 seconds and would then disappear. To measure the trust, robot error has to be introduced.

**Robot Error**

To introduce this, there has to be a setting of a slider in the options section in the menu scene of the game which allows the user to change the failure rate of the robot. This will be stored as a variable and when the user calls for help, a function would be defined in order to determine if the robot has failed in giving the correct suggestion. Using Unity’s Random.Range() function would have a random number which is then compared to the failure rate set by the user. If the random number is greater than the failure rate, the robot will suggest the correct path to take, otherwise the robot will suggest an incorrect and random door to traverse to. This will also be checked if the user uses the door that was suggested with the box collider already in place on the doors. Eventually, the user will find the exit and will then start saving data within a file.

### 5.3.6 Timer/Saving to a file

As soon as the game starts, a timer will start. This is displayed using UI Text in unity anchored on the top centre in green text displaying *“Time XX:YY”*with X marking the minutes since the timer starts, and Y meaning the seconds.

To create a class for this, named Timer, there needs to be a Text variable timerText(this will be displayed in the UI element), a float startTime, and a boolean check timeStopped. With the start() function built into unity when the first frame is created in the game setting, the startTime is defined as the current overall runtime of the game meaning that it will always be zero. Then, in the update() function also built into unity where it is called when every frame is rendered within the game, it checks if the time has stopped (timeStopped is true) and breaks from the update function with no alterations in variables. If the timer has not stopped, then a float is defined as t which is the built-in unity Time formatted in a number of milliseconds. This is then passed into strings “minutes” and “seconds” where minutes divides all the milliseconds by 60 and seconds takes the modulus of 60 on t. Both variables are defined as integers or floating point values so using another unity built in function toString() ensures that they are outputted as string values. Then, the timerText then places the minutes and seconds in one text variable which is then called on to be displayed within the User Interface. Once the user exits, this information is passed through to the exit controller.

Saving Data in a text file is very straight forward with Unity. A Unity package named System.IO implements an Input and Output system whereby information can be stored locally as a file by using File.WriteAllText(). This function takes in a data path followed by a string of what should be stored into the local file. Initially, I used a local data path with “C:/Users/AdamJ/Desktop/Project/..” which worked well however if the game was sent to other users for testing, this would have not worked as they might not have a user named AdamJ let alone have the project saved on their respective Desktops. Fortunately, Unity’s built in “Application.persistentDataPath” allows the data path to always be stored locally in the same folder as the unity game as long as they are running in a Windows operating system. This allowed multiple users to have their data saved into a text file named simply “save.txt”. Although this is limited to only windows users, most subjects involved in this investigation would be using Windows which made it less of a problem to fix.

# Chapter 6 Future Work

## 6.1 Optimisation

With all code-based programs, optimising code for performance is key. Not only would having perfectly optimised code run more smoothly for users who have more than capable hardware, but would also allow users to run programs with very low requirements thus increasing the availability of usage. With this game in mind, the most hardware straining and what has highest importance for optimisation is procedurally generation of a map.

This is important to optimise because with the map generation function, this particular function is called multiple times to merely instantiate the game environment. This would be called once every frame until the map is created. Each time a function is called, a sizable chunk in memory is used. This means that because a function is called multiple times, there is a lot of memory usage which in some machines may be too costly and in some rare cases crash their game or computer. With this in mind, all unnecessary code must be removed in map generation. Over the coming months, I plan to review multiple procedural generated games to determine which is the most efficient way that is implemented in map generation. Due to time constraints, which will be outlined in the next section, this was a task which I was unable to complete given the time frame.

## 6.2 Time Constraints

In industry, most video game development, such as this one, would typically be developed with a larger time frame than allocated in this project and with more developers focussing on different aspects of the game. This project was quite ambitious considering the components(procedurally generating a map, spawning collectables, having a Robot instruct the user how to navigate the map, and having the user walk around and interact with objects) and the time frame given to complete it.

I will be continuing this project in the coming months so that I can see it through to completion. I anticipate that this will take me roughly 6 months considering data gathering from user studies and analysing the data. This will be outlined in more detail in section 6.3.

## 6.3 Next Steps

As mentioned, I will be continuing this project in the future. My first step will be developing and eventually implementing the Robot component as outlined and explained in section 5.3.5. After this element has been implemented, I will have a fully functional prototype with which I can start user testing immediately.

During user testing, I will focus on optimization of code to ensure that all user studies will not be affected by performance to maintain the integrity of the data gathered. Once all data is gathered, I will create data visualisations to analyse user patterns and trends. With analysing the data, I will attempt to identify the answers to the following questions:

1. Research Question - Do users trust the embodied robot in close proximity as compared to on screen.
2. Do error-rates impact the perception of trust in collaborative games.
3. How can we devise an online metric of trust in real-time (situations in a game when a user asks for help, if they follow the help, or potentially other situations.)

## 6.4 Conclusion

To conclude, this is an ambitious project. Despite time constraints I have still demonstrated an advanced ability to produce a procedurally generated game from scratch. Even though the project has yet to be completed, I am on track to finalise implementation and analysis of data gathering in future work(mentioned in section 6.3).

Through this document, I have outlined the problem, research and background for different components, planned and implemented various aspects in the game, and laid out the foundation for future development within this project.

Overall, I believe I have developed and honed my skills as a computer scientist through this project. Understanding grammar for C# code as well as gaining more knowledge on procedural generation, game design, and bug-fixing. I am fully confident in my abilities to continue this project in future work and to learn new skills in future development.

# 

# Bibliography

[1]  Blog, R. (2012, April 8). *Metaprogramming for madmen*. The Ryg Blog. [Online]. Available: <https://fgiesen.wordpress.com/2012/04/08/metaprogramming-for-madmen/>

[2] Hatfield, T. (2013, January 21). *GameSpy: Rise Of The Roguelikes: A Genre Evolves - Page 1*. Pc.Gamespy.Com. [Online]. Available: <http://pc.gamespy.com/pc/ftl-faster-than-light/1227287p1.html>

[3] Van der linden, R., Bidarra, R., & Lopes, R. (2014). *Procedural generation of dungeons*. Research Gate. [Online]. Available: <https://www.researchgate.net/publication/260800341_Procedural_Generation_of_Dungeons>

[4] Kuo, R. (2012, April 19). Why Borderlands 2 Has the Most Stylish Guns in Gaming. *Wall Street Journal*. [Online]. Available: <https://www.wsj.com/articles/BL-SEB-69805>

[5] Isaac Computer Science(2021) [Online]. Available: <https://isaaccomputerscience.org/concepts/dsa_search_a_star>