Progress report marking sheet

Student's name: **Matthew Taylor**	Student's Reg: **100151729**	
Supervisor: **Dr Rudy Lapeer**	Date:	Signed:
2nd marker:	Date:	Signed:
Title: **Third Year Project: Progress Report**		

	Agreed Marks
Description of project: aims, motivation	/10
Description and understanding of issues and problems addressed in the project	/20
Design and planning	/50
Evaluation of progress	/10
Appropriate use of LaTeX	/10
Overall mark	/100

Comments

Report and completed mark sheet should be returned to the LSO.

Third Year Project: Progress Report

Matthew Taylor

registration: 100151729

1. Introduction

1.1. Aim of the project

The project aims to implement a first person video game which uses procedural generation (PCG) to provide a novel and interesting game experience. The project will explore the technical aspects of using PCG as a core feature of a game, specifically, implementation and performance.

It will focus on real-time procedural level generation and providing a compelling, dynamic level structure.

1.2. Motivation

Procedural generation is not a new field, especially in video games, but there are interesting areas that are seldom explored.

The motivation behind the project's aims are to explore new and interesting concepts, namely, using PCG as a core feature in a game and using real-time procedural generation to provide an interesting experience.

2. Issues and problems

2.1. Designing 3D games

Designing 3D games and the assets required to populate the levels created could take a long time, especially if much of the functionality is created from scratch. To enable the project's aims to be achieved in a realistic timeframe, the Unity game engine was chosen. This tool enables relatively rapid production 3D game enviroments, supporting the design of assets and enabling quick prototyping of ideas. Unity supports the use of C# as a scripting language to program game features in. It also provided a lot of documentation, which was useful in learning the framework and the systems it provides. Much of the initial stages of the project focussed on learning Unity.

2.2. Performance

How a game performs is critical to it's success, even at a basic level. If the game cannot be played in a sustainable framerate on common hardware, it could be considered a techncial failure. The prototype stage considered performance at every aspect.

2.3. Evaluation of success

A key issue with a project of this type is how success is evaluated. There are a few obvious candidates, which I will list, along with some more subtle intepretations of the projects success.

2.3.1. Performance

As alluded to previously, performance is one of the key components of the projects success. Performance will be measured in a few ways:

- Frame rate
- Initial loading time
- Smooth level re-generation (no jitter when level changes behind player)

2.3.2. Convincing levels

The levels must appear convincing. This is difficult to quantify, but nevertheless is an important attribute. The levels (at least in the initial design) will be mazes generated in a grid, with the grid sections subdivided in different ways.

This could easily create a boring level, where the grid pattern is discernable and the rooms are boring and procedural to traverse. Successful generation of levels will avoid these issues.

3. Design and planning

3.1. Implementation methodology

The project development operates under an Agile methology. Agile does not require that all requirements (or "user stories") are defined up front, instead it is encouraged that they are developed and refined during the lifecycle of the project.

The project itself is divided into Agile "sprints". It was decided to deliver a sprint every two weeks, as this fit with the development time given to the project and providing updates to the project supervisor.

While all the stories were not required at the outset, as part of the planning, it was decided to plot rough timescales against Agile "epics". These epics provide a high-level view of functionality and are detailed in Table 2 on page 7.

The epics also contain very high-level user stories, which provide a little more detail. These were decided at the project outset, then refined during development. Before any user story can be developed, it is required that more detailed requirements are written. These are then recorded and tracked on Trello.

3.2. Implementation plan

With the high-level design documented in the epics, planning of these epics against timescales was required. This timeline plan is visible in 1 on page 6. Progress was then tracked against this Gantt chart to ensure that progress was being made and that the project could be achieved as planned.

3.3. UML

As the project began, a UML diagram was designed to provide a high level framework to work towards. Unity provides some pre-defined structure, in that all game objects must be part of a Scene object to work. Aside from this, Unity does not prescribe a lot of structure, so having a UML design to begin with was useful. The initial UML design can be seen in Figure 3 on page 8.

3.4. Requirements and prioritisation

3.5. Design

3.5.1. Guaranteed path generation

The first part of the design to be implemented required a guaranteed path through a maze to be generated. This was achieved by using a random walk - a stochastic process implmented in two dimensions to provide a definite route through the maze.

The random walk is implemented using an agent based approach. The agent can choose from three random directions to travel in - left, right or down. These probabilities are detailed in Table 1.

Left	Right	Down
40%	40%	20%

Table 1: A table showing probability of direction random walk agent will take

This skewed probability distribution encourages the path generating agent to meander left and right more than moving down, generating a path that is longer and doubles back on itself more.

When the agent has chosen a valid direction, it randomly selects a valid tile. To aid in creating an interesting maze, the agent can choose from a variety of pre-made tiles with varying exits. For example, it may choose to generate a room with only left and right openings, or a tile with openings on the left, right and down-facing aspects.

The agent has methods of checking where it has been and where it is going next, so that it picks valid tiles that have openings in the directions it has come from and intends to go.

The agent is given bounds and restrictions while it is performing the random walk. These bounds ensure the generation does not stray too far left or right. When the agent reaches the lowest down-bound, it stops generating the level and spawns the level's goal in the final tile it generated.

These bounds are adjusted by parameters, which aid in development and testing and can be adjusted based on performance or desired level size.

4. Evaluation of progress

- Changes to aims/scope/work plan

References

A. Gantt charts

B. Diagrams

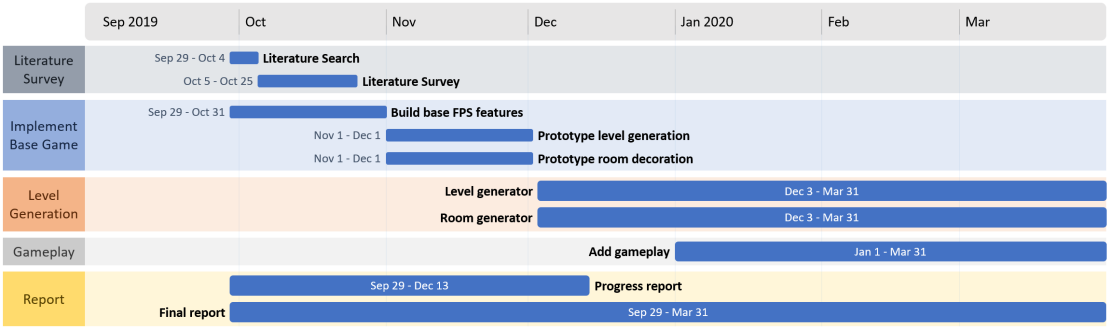

Figure 1: Gantt chart of original plan

Epic	Stories
Build base "first person" 3D game features	
	Provides a basic level design
	Proves the concepts of vision and movement
Prototype level generation	
	Prove a method of generating a guaranteed path through a maze
	Prove a method of filling in non-guaranteed paths through the maze
Prototype room generation	
	Prove a method of generating room interiors
	Room generation must not interfere with guaranteed path
Prototype level re-generation	
	Design method to procedurally re-generate level sections
Add gameplay elements	
	Decide on how level is finished by the player
	Provide some interest and threat when playing
	Provide means of assisting navigation
Implement final designs	
	Use non-prototype textures
	Ensure performance meets goals

Table 2: Agile-style epics and associated user stories

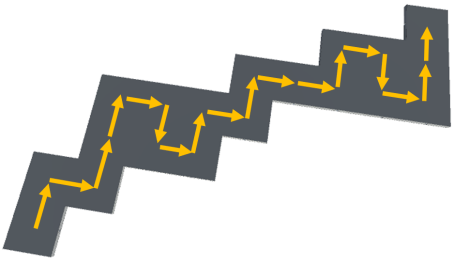

Figure 2: A top-down view of the guaranteed path generation

C. Tables

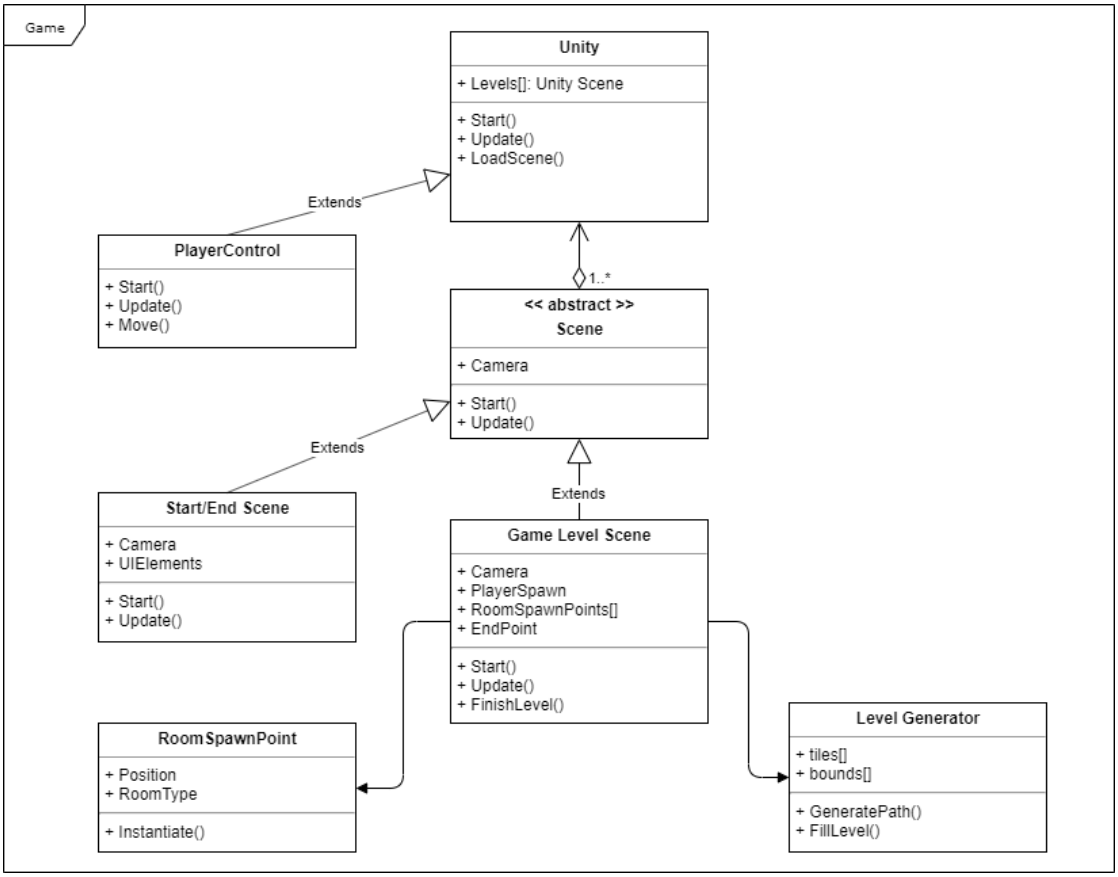

Figure 3: Initial UML diagram