Final Year Project Interim Report

Full Unit - Interim Report

Building a Game

Submitted By Sulaimaan Bajwa

Evaluation of my progress so far in the final project critical to the completion of the 3rd year of

Msci (Hons) Computer Science with Software Engineering

Supervised by Hugh Shanahan
EPMS School, Department of Computer Sciences
Royal Holloway, University of London

1 Abstract **4**
1.1 Aims and Objectives 4

2 Background **5**
2.1 Video Games 5
2.2 Roguelikes 5

3 Technology Theory **7**
3.1 Unity and Game Engines 7
3.2 Quaternions 7
3.2.1 Gimbal lock 7
3.2.2 Quaternions For Rotations 8
3.2.3 Application in Top-Down 2D Games 9

4 Product Research **10**
Conclusion 12

5 Software Engineering **13**
5.1 Design Patterns in Video Games 13
5.1.1 Visitor 13
5.1.2 Factory 13
5.1.3 FlyWeight 14
5.1.4 Type Object 14
5.1.5 Game Loop 15
5.1.6 Update Method 15
5.2 Testing a Video Game 16
5.2.1 Unit Testing 16
5.2.2 Identified Issues 16
5.2.3 Current Approach 16

6 Summary Of Completed Work **17**
6.1 Revised Winter and Term 2 Schedule 18

7 Bibliography **19**
7.1 Background 19
7.2 Unity and Game Engines 19
7.3 Quaternions 19
7.4 Product Research 19
7.4 Design Patterns 20
7.5 Testing 20

8 Diary **21**

Project Diary - Building a Game **21**
28/09/2022 21
29/09/2022 21
02/10/2022 21
03/10/2022 22
Meeting notes 22
04/10/2022 22
05/10/2022 22
07/10/2022 22
10/10/2022 23
18/10/2022 23
19/10/022 23
21/10/2022 23
25/10/2022 24
Meeting Notes 24
28/10/2022 24
01/11/2022 24
04/11/2022 24
08/11/2022 24
Meeting Notes 24
12/11/2022 25
22/11/2022 25

1 Abstract

During the course of this project I will be attempting to design, create and deploy a video game. I hope to learn more about Unity as a game engine and how to develop and test with it, with the objective that it will be a strong foundation for my software engineering portfolio. Unity is a game engine rooted in the very foundation of the industry, helping developers create games for nearly two decades, and already contains many of the elements of important software design patterns I intend to use. I have chosen to create a Roguelike game due their current popularity and approachable- but challenging- nature. Roguelikes often have a repetitive cycle consisting of frequently reused modules, perfect for tackling with software design patterns in mind.

1.1 Aims and Objectives

To complete this project, as per the project specification, must have achieved 2 things: creating a video game, and deploying said video game. To do each of these steps will require research and practice, broadening my overall list of objectives.

I must research how to utilise a game engine, how to script the interactions I need for the mechanics of the game to work together, and understand how to generate levels for the player to experience.

I also must choose a platform and find a means of deploying the game, be it on a digital store, or as shareware, and be able to play the game as a standalone product, without the need of the development material on each player's system.

2 Background

2.1 Video Games

Perhaps one of the main forms of software mass produced by teams and solo developers globally, video games frequently feature the newest techniques and sometimes require the frontier of consumer technologies to run. As a result, efficiency within the code of a video game is paramount. In this project, I seek to create a video game using software engineering techniques, creating an efficient and fun game. This requires intimate understanding of both software engineering and the genre or genres I plan to develop within.

There are many pieces of software that enable me to do this, one of which is the game engine Unity, which efficiently handles core elements of a video game, such as the game loop and physics interactions.

2.2 Roguelikes

As a genre, Roguelikes[1] get their name from a game released about 1980 as free shareware. The game was called ‘Rogue’ and was an ASCII art visual game about navigating several floors of a dungeon to find a relic. Many elements of Rogue are still used to this day within new roguelike games. These features include permadeath, items and a room-by-room iterative playstyle. Back then, items were noted as ? symbols, the player character was an @ symbol, and enemies were capital letters, all 26, each of which specified a different creature.

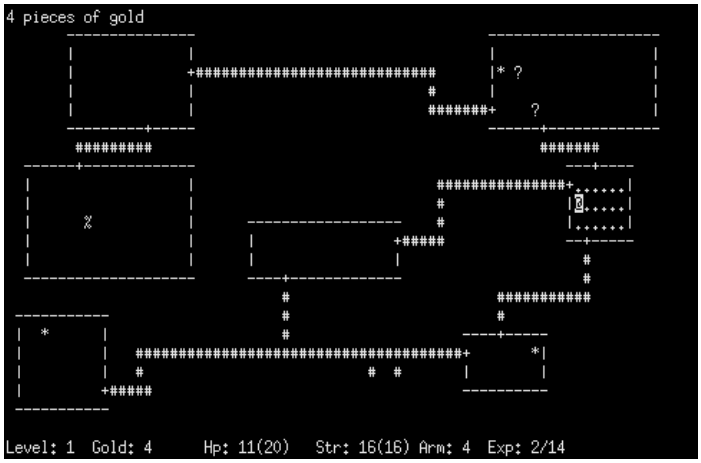

Figure 1: A screenshot from *Rogue (1980)*

The current iteration of the genre sees several of Rogues features still in use, namely a variety of enemies, traversing and descending a multi-layered dungeon, collecting items that enable certain mechanics, and permadeath. All of these mechanics are important elements that define the genre, but some have been altered since their first appearance in Rogue. Permadeath in particular is a simple mechanic that has been altered sometimes on a game-by-game basis. While the definition of permadeath remains the same - the death of a character ends the run, the character cannot be revived to continue or restart the run - there have been more engaging

implementations of it that allow the user to maintain a sense of direction and progression that keeps them playing the game, run after run. If the player dies more frequently then they progress, the player generally loses interest[2].

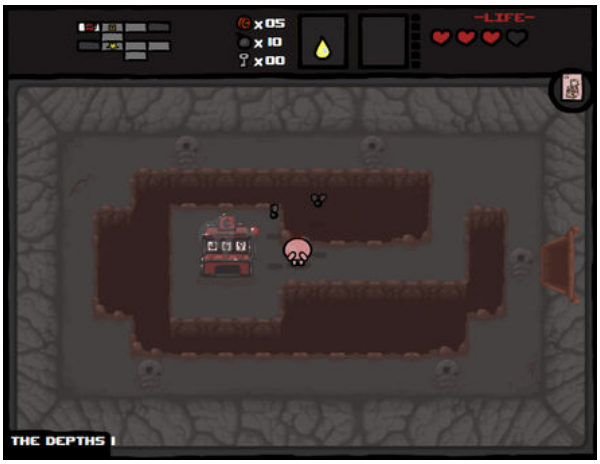

Figure 2: A screenshot from *The Binding of Isaac (2011)*

3 Technology Theory

3.1 Unity and Game Engines

To facilitate production of my game, I opted to use a game engine, ‘a software framework designed for the creation and development of video games’ [3]. These pieces of software form a toolkit that allows for the largely repetitive low-level programming that renders graphics, calculates physics, handles and integrates inputs and scripts, and runs the game loop to be reused incredibly quickly. While parameters within these modules can often be tweaked, video games can be created with these fundamentals untouched. Live-support game engines are frequently updated to bring optimisations for games running on them, as well as features for developers to utilise[4].

Building on top of an engine means that a developer now only has to provide assets and scripts to successfully create a game. These scripts are largely the focus of my project, as I can use software engineering techniques, such as the use of design patterns, to solve common issues, like reducing the technical resource requirements for my game.

3.2 Quaternions

There are two primary variants of 2D games: Top-Down and Side-Scrolling. Most forms of 2D rotation in Unity are designed to support side-scrolling games, while my game is Top-Down. As a result, I have to utilise 3D rotation methods, which include the other axis I need. One such form of rotations are Quaternions, they solve a very specific problem that I encountered during the creation of my game: gimbal lock.

3.2.1 Gimbal lock

Consider the pitch, roll and yaw of an object. These are descriptors used to identify the rotation of an object in 3D space, and while generally they do a good job in doing so, there are very specific cases in which this gimbal setup can lock itself out of proper rotation, physically. In the case that 2 of the 3 rotational axes line up, a gimbal lock occurs, in which one of the rotational axes can no longer be added to the rotational matrix to have an effect. Figures 3 through 6 are taken from an external source for the purposes of simple visual explanation[5].

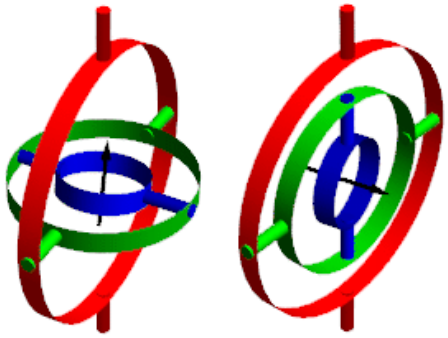

Figure 3: a Gimbal exhibiting Gimbal Lock, where the Blue and Red Axis have aligned, caused by a 90 degree rotation in the Green Axis.

This is evident within the mathematical depiction of a rotation using matrices.

$$\begin{pmatrix} C_\alpha C_\beta & -C_\beta S_\alpha & S_\beta \\ C_\gamma S_\alpha + C_\alpha S_\beta S_\gamma & C_\alpha C_\gamma - S_\alpha S_\beta S_\gamma & -C_\beta S_\gamma \\ -C_\alpha C_\gamma S_\beta + S_\alpha S_\gamma & C_\gamma S_\alpha S_\beta + C_\alpha S_\gamma & C_\beta C_\gamma \end{pmatrix}$$

Figure 4: A 3x3 z-y-x rotation matrix, where C_n and S_m are Cos(n) and Sin(m), using angles α, β, and γ, or the Red, Green and Blue Axis respectively

If I were to substitute in π/2 as β, where π/2 is a 90 degree rotation in radians, Cos(π/2) = 0 and Sin(π/2) = 1, resulting in the following matrix.

$$\begin{pmatrix} C_{\frac{\pi}{2}} C_\alpha & -C_{\frac{\pi}{2}} S_\alpha & S_{\frac{\pi}{2}} \\ C_\gamma S_\alpha + S_{\frac{\pi}{2}} C_\alpha S_\gamma & C_\alpha C_\gamma - S_{\frac{\pi}{2}} S_\alpha S_\gamma & -C_{\frac{\pi}{2}} S_\gamma \\ -S_{\frac{\pi}{2}} C_\alpha C_\gamma + S_\alpha S_\gamma & S_{\frac{\pi}{2}} C_\gamma S_\alpha + C_\alpha S_\gamma & C_{\frac{\pi}{2}} C_\gamma \end{pmatrix}$$

Figure 5: The substitution before simplification

$$\begin{pmatrix} 0 & 0 & 1 \\ C_\gamma S_\alpha + C_\alpha S_\gamma & C_\alpha C_\gamma - S_\alpha S_\gamma & 0 \\ -C_\alpha C_\gamma + S_\alpha S_\gamma & C_\gamma S_\alpha + C_\alpha S_\gamma & 0 \end{pmatrix}$$

Figure 6: Simplified substitution, in which there is no longer a β present for further substitution.

Figure 6 shows the result of this substitution in β is an inability to substitute in any more β values, ergo the inability to rotate one of the axes at all. This is a gimbal lock of the Green Axis mathematically. In real models or virtual simulations, without external correction, the lock cannot be undone. Quaternions do not suffer from this issue, and so are a better option for me.

3.2.2 Quaternions For Rotations

The use of quaternions hinges on Euler's Rotation Theorem, where the rotation of an object in 3-dimensional space can be defined by a single rotation θ about an axis defined by a vector from a fixed point. This is directly why a quaternion's notation is q = (w,r), where w is a scalar value representing the angle of rotation, and r is a vector, representing the line around which the rotation occurs. In Figure 7, these values w and r are noted as θ and ê respectively.

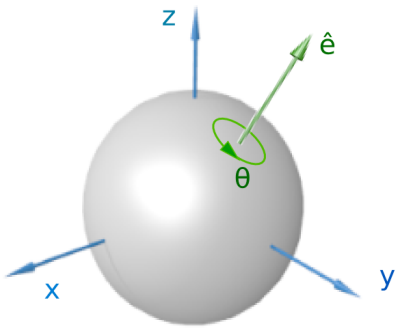

Figure 7: A rotation of scalar θ degrees about the euler axis vector ê.

3.2.3 Application in Top-Down 2D Games

In Unity, some much simpler rotation functions use the gimbal rotation system. As a result, these functions are prone to gimbal lock. I found that as the player character moved directly north or south of an enemy, the enemy would suddenly flip trying to face the player, breaking the 2D feel of the game. Quaternions allowed me to remove this sudden flip, rotating objects smoothly in all cases, as all rotations are done around a single, defined line in any direction, and are not solely dependent on x, y and z axis lines.

4 Product Research

In order to effectively create a game, I must break down other games in the genre to see exactly what the key elements of a roguelike are. From this, I should be able to get some understanding of what I can do to effectively produce a good roguelike. Design patterns can start to be identified as common elements emerge.

While the primary focus of this project is to create a well coded game, using software engineering techniques, and not necessarily to create the most creative or unique roguelike game, it is important that I understand the target genre before creating any game.

Game	Similar to Intended Scope	Different to Intended Scope
Binding of Isaac [6]	Top-down 2D	
	Procedurally generated box rooms, forming a level of a dungeon	
	Boss fights	Roughly set final bosses/Unique bosses
	Templated rooms, and enemies	Entirely unique enemies, often with effects the player can obtain through items
	Player descends to progress toward the final boss	Roughly set final bosses
	Simple item system, with active, passive and culminating items	Many item types
Cult of the lamb [7]	Almost top down/2.5D graphics	
	Simple attack gameplay	
	Random generated dungeons	
	Boss fights	
	Generated rooms, with unique features	
	Permadeath	Integrates colonySim genre gameplay elements to delay the repetitive act of dying

Hades [8]	Levelled progression, multiple floors of dungeon	
	Boss fights	Traps/other forms of damage outside of combat
	Randomly generated map	Incredibly large, complex dungeon room generation
	Character ability building in the form of boons	Huge variety of types of weaponry and damage forms
	Permadeath	Permanent upgrades that persist between runs
		Large variety of enemies
		Isometric instead of grid 2/2.5D
Enter the Gungeon [9]	Top down 2D	
	Has a wide variety of weaponry, also works in place of items, generally	Weapons take on the role of essentially all itemisation
	Random generated dungeons, descending floor system	Room are not all 1 size, or shape
	Permadeath	Additional resources to provide safety, and dodge system
		Expansive shop system
		Has a variety of unique enemies
		Targets a sub-genre of roguelikes, called bullet hells, designed to provide an overloading experience of learnable patterns.

Figure 8: Table of major research points.

4.1 Conclusion

Major elements of successful titles that I plan on including:

- randomly generated, multi-levelled dungeons
- Broad array of items
- An array of enemies, and some diversity in bosses
- 2D view of some kind
- Simple, but modifiable gameplay
- Permadeath

These should be considered required elements, unless I can come up with an inventive system that can provide the function of one of these elements, without including the element specifically. Additional, I can identify some areas in which design patterns could be used:

- Factories can be used to create randomly generated dungeons, items and enemies.
- FlyWeights and Type Objects can be used to define and further specify items and enemies.
- Visitors can be used to facilitate shared functions between enemies, items and the player

These are in addition to the design patterns built into, and enforced by, Unity, such as the Game Loop, and Update methods to interact with it. A more thorough insight into these Design Patterns can be found in the Software Engineering Section.

5 Software Engineering

5.1 Design Patterns in Video Games

As one of many forms of good practice, modularity, and therefore decoupling, of code is quintessential to the ongoing work of a software developer. Being able to re-utilise elements of your portfolio again makes you a valuable developer, bringing with you the correct tools for the job at hand. My game is no different, and if I were to develop games in the future, or even allow access for another individual to see my source code, the game would be designed in such a way that elements could be reused. Design patterns also seek to create a readable, maintainable and efficient codebase, such that another person could easily get to grips with the code if need be, and the code itself runs with a small, refined footprint. Below are the efforts I have made to incorporate design patterns into my work, by breaking down how and why I use each one.

5.1.1 Visitor

Visitors[13] exist to decouple a repeated function from every participant of some structure, allowing a single traversing object to assert the function on each object instead. In the case of my game, projectiles act as visitors, in a much less obviously connected structure of enemies to one another. Each enemy exists as a child of the enemy controller, also acting as the FlyWeight. Any projectile sent by a player would effectively traverse this structure.

Projectiles, such as the bullets shot by both enemies and players, operate as visitors, even though they lack the exact visit/accept functions. Unity handles collision between any two objects as, essentially, visiting. When an object collides with a second object, both gain the ability to refer to each other. This allows the projectile to hold all the functions it needs to afflict anything onto any given object it collides with, and potentially multiple objects, if the bullet has the ability to pierce entities. This decouples the objects from each other, utilising visiting projectiles to interact with any given object, so long as it has a health value. Once the purpose of the bullet is served, it can be destroyed to preserve resources.

5.1.2 Factory

The purpose of a factory[13] is to separate the need to mechanism by which you instantiate a set of objects. These objects could be in large numbers, or just rather convoluted to make individually within a script. As a result, a script is written to specifically instantiate a type of object, with minimum required arguments going forward.

All prefabs are generated as children of a class that has a designated factory, spawning in objects, and then passing on all required specifics for that class to operate. A good example of a factory in my code will likely be the enemy factory, which spawns in enemies, and assigns them their Object Types. This enables easy creation of enemies from prefabs, and allows me to quickly summon enemies using debugging tools if necessary.

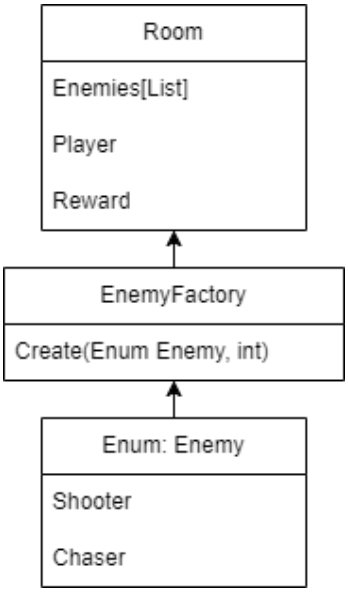

Figure 9: the EnemyFactory found within my game, in which the room may summon any number of a single type of enemy at any time.

5.1.3 FlyWeight

In order to significantly streamline all entities spawned into the scene from prefabs, I will need to remove all the shared components and implement them within some parent spawning class, working as a means to have just a single instance of unchanging data, rather than spawn every entity with its own hard copy of identical data. This is called a FlyWeight[11]. It follows from the idea that duplicate code is wasteful, so by passing a shared, single instance of the code required, I remove the necessity for many other instances of identical code. An example of which within my game will likely be enemies, of which there are a few types, but they all will share common variables, such as damage values, the location of the target, and their movement speed.

5.1.4 Type Object

In order to specialise objects of the same type, such as enemies, or projectiles, I can use the Type Object[11] design pattern, providing an additional script to each object of a given type, importing all the specific functions designed for that subtype of object. There are currently 3 types of enemies planned for my game, a melee chaser, a shooter, and a stationary shooter. Each of these types will be specified by their own script.

Figure 10: The EnemyShooter GameObject, in which the HealthController and EnemyMotion scripts are both FlyWeights, used by many different entities with common features, and the EnemyShooter script asking as a Type Object script, only used by ‘shooter’ variant enemies.

5.1.5 Game Loop

Game loops[12] are one of the most critical of game design patterns to have been formed. It seeks to solve the issue of the broad variety of hardware that game developers often need to accommodate for. Most hardware is designed for many purposes, and as a result, to be able to deal with the differences in processor clock-speeds, the game executes instruction on a unified ‘Tick’ generally occurring a certain number of times a second. This way, the game doesn’t actually increase/decrease in pace based on the speed of the processor running the game. The game loop system is provided by the game engine itself, and I can tie instructions to it using the update methods available to me.

5.1.6 Update Method

In order to tie scripts to game ticks, I must use the update functions. There are a few available to me that allow me to control the order in which scripts are executed. The first, most prioritised function is the awake function, which happens as soon as the object is instantiated, before its initial start call. The start call then runs, frequently setting up initial script relationships with the controller, and any other required objects. The update[12] and fixedUpdate functions are then called every tick; while update is tied to the graphical frame rate of the game, which can change due to processing limitations, fixedUpdate is always executed every tick. Every class within my game operates alongside the Game Loop, and so each class must implement at least one of these Game Loops updater.

5.2 Testing a Video Game

5.2.1 Unit Testing

Unit testing[14] seeks to take a small, manageable element of code and provide it data to process, asserting that the result from that processed data is what was expected. In a typical environment, Unit testing is automated, to ensure that a developer does not need to manually test individual aspects of their code every time a change is made to a pathway that uses that section of code. This is a highly effective and efficient way of testing code, and frequently squashes bugs resulting from incorrect inputs, incorrect outputs, or incorrect processing.

5.2.2 Identified Issues

In order to conduct unit testing, test inputs have to be created in the manner that data is expected to be received by the function. Writing an identical function that takes in a differently formatted data could be seen as an approach to combat having to write incredibly complex data, however, you are then testing a unique piece of code, as it has to handle the test data differently. As a result, this is a flawed approach. If constructing valid test data in an identical format is an issue, it becomes impossible to unit test. That is unfortunately largely the case with my game thus far.

5.2.3 Current Approach

To effectively test functionality along a similar idea to unit testing, I can create a series of testing maps/environments within the game, allowing me to engage each element of a script in an orderly way. I must avoid attempting to test several features at once, as I may struggle to narrow down the culprit lines within my scripts. For every piece of functionality I write, I will need to write tests for all interactions between that function and other game objects or functions. This will be particularly important, I foresee, for the intended item system.

6 Summary Of Completed Work

6.1 Research Enumeration

Of the reports, I have completed:

Initially Planned

- Current Major Titles

Unplanned

- Design Patterns
- Testing
- Quaternions
- Refactor Plan

I have partially completed:

- List of required Algorithms and Structures - Partially complete
- Design Document - Partially complete
- Major Elements of a Roguelike Game - Partially complete

6.2 Technical Achievements

Of the proof of concepts, I have completed:

- Player Motion
- Enemy and Damage System

I now know how to manipulate GameObjects, and can write scripts to do so using the Unity Libraries in C#, a language which shares many traits with Java, which I know well. I have constructed several persisting objects that I can integrate with my scripts, called Prefabs, which can be instantiated and modified by factories with ease. Many of the key elements of Unity game development have been learned in preparation for the more algorithmically complex stretch of work in term 2. As a result I have a prototype which currently consists of a room of 4 enemies - all prefabs - which the player must destroy, utilising projectiles, before the enemies destroy the player. Eventually, even the room will be a prefab, allowing them to be stitched together into floors of a dungeon, creating the whole level using a randomised algorithm for procedural generation.

Of the Testing Tools, I have not yet created any of them as I have not needed to use such tools as of yet, but intend to still produce each testing tool in the near future. Testing player and enemy motion is not particularly viable with tools, as opposed to simply engaging with the game myself and moving the player, and watching the enemies path towards the player intelligently.

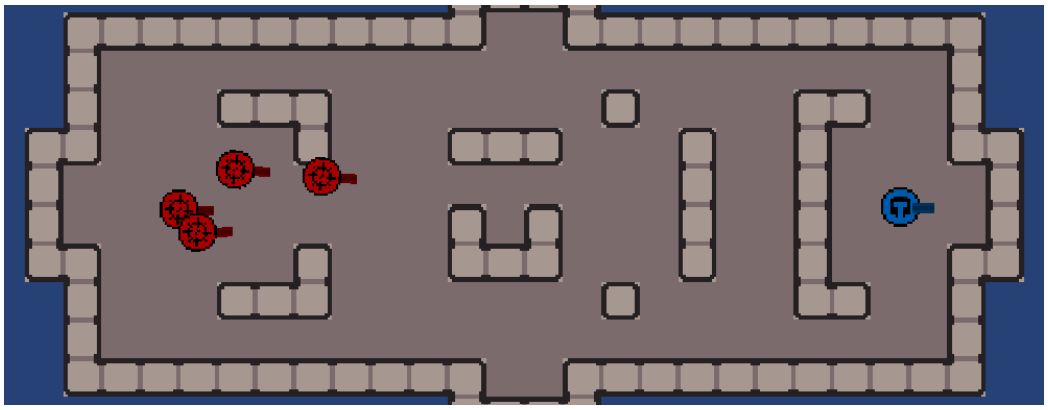

Figure 11: A Screenshot of my game, featuring 4 enemies, red, pathing towards the player, blue. The player must destroy the 4 enemies to beat the room, and in theory, move onto the next room.

6.3 Revised Winter and Term 2 Schedule

(Academic) Week	Milestones	Specific Notes
Winter Break		
12/12/2022	Construct GUI diagrams	Part of missing required elements
	Start creating UI elements within Unity	
19/12/2022	Link GUI to the beginning and escape menu of the game	Likely need to sever update() functions
26/12/2022	Build item proof of concept	
02/01/2023	Start Research on random generation	
Term 2		
Week 17 & 18 09/01/2023 & 16/01/2023	Create algorithm that randomly generates map	Major complexity of project, see report on PCG
Week 19 23/01/2023	Create template rooms to insert into randomised maps	Bulk of final programming, allowing me to tunnel-vision and complete the code, mitigating a lack of report detail. Large majority of this section will fall under the crucial design patterns section of my final report, important to constantly reflect in my diary.
Week 20 30/01/2023	Finish templates, add them to generation algorithm	
Week 21 06/02/2023	Work on character visual modification algorithm	
Week 22 13/02/2023	Finish character visual	
	Work on enemy set templates	
Week 23 20/02/2023	Integrate the enemy randomisation into algorithm, using both regular and boss enemies	
Week 24 27/02/2023	Draft up Final Project Report	Important for feedback on major elements of the final report
Week 25 06/03/2023	Start finalising and testing programming side of project for submission	Squash major bugs, ensure code runs as expected, on target systems.
Week 26 13/03/2023	Finish Final Report draft and start work on feedback	Crucial for an optimal Final Report
Week 27 20/03/2023	Finish report and project for submission	March 24th Final submission for all project documents

Figure 12: Table of key goals following the submission of this interim report.

7 Bibliography

7.1 Background

[1] Wichman, Glenn. “Rogue History.” *Rogue History*, 1997, https://web.archive.org/web/20150217024917/http://www.wichman.org/roguehistory.html.

[2] Ahn, J. K. (2016). Mechanism of Permanent Death in Rogue-like Games. *Journal of Korea Game Society*, *16*(1), 33-42.

7.2 Unity and Game Engines

[3] Technologies and Innovation: Second International Conference, CITI 2016, Guayaquil, Ecuador, November 23-25, 2016, Proceedings. Germany, Springer International Publishing, 2016. Page 146.

[4] F. Messaoudi, G. Simon and A. Ksentini, "Dissecting games engines: The case of Unity3D," 2015 International Workshop on Network and Systems Support for Games (NetGames), 2015, pp. 1-6, doi: 10.1109/NetGames.2015.7382990.

7.3 Quaternions

[5] Hughes, Mark, “Don't Get Lost in Deep Space: Understanding Quaternions - Technical Articles.” *All About Circuits*, 2017, https://www.allaboutcircuits.com/technical-articles/dont-get-lost-in-deep-space-understanding-quaternions/

7.4 Product Research

[6] Edmund McMillen. (2011, September 28). *The Binding of Isaac on Steam*. Steam. https://store.steampowered.com/app/113200/The_Binding_of_Isaac/

[7] Massive Monster. (2022, August 11). *Cult of the Lamb on Steam*. Steam. https://store.steampowered.com/app/1313140/Cult_of_the_Lamb/

[8] Supergiant Games. (2020, September 17). *Hades on Steam*. Steam. https://store.steampowered.com/app/1145360/Hades/

[9] Dodge Roll. (2016, April 5). *Enter the Gungeon on Steam*. Steam.

https://store.steampowered.com/app/311690/Enter_the_Gungeon/

7.5 Design Patterns

[10] Ampatzoglou, A., & Chatzigeorgiou, A. (2007). Evaluation of object-oriented design patterns in game development. *Information and Software Technology*, *49*(5), 445-454

[11] Nordeus, Erik. “Game programming patterns in Unity with C#.” *Habrador*, https://www.habrador.com/tutorials/programming-patterns/

[12] Nystrom, Robert. “Game Programming Patterns.” *Game Programming Patterns*, 2009, http://gameprogrammingpatterns.com/

[13] Gamma, Erich. Design Patterns Elements of Reusable Object-Oriented Software. 37th printing. ed., Addison-Wesley, 2009.

7.6 Testing

[14] Huizinga D Kolawa A. Automated Defect Prevention: Best Practices in Software Management. Hoboken New Jersey: Wiley-Interscience; 2015. doi:10.1002/9780470165171.

8 Diary

Project Diary - Building a Game

28/09/2022

This diary has been created to keep myself in line with my goals, as well as actually being able to outline them here over time. Currently, i have:

- A project plan draft with all the key sections
- This repo cloned for editing .md files in VSCode
- Some small collection of papers/books/games for research and analysis
- A proof of concept for auto-tiling rooms correctly in Unity
- A loose game design document
- ideas i haven't written out, which i'm gonna do below

What i am doing right now:

- The second draft of my project plan, tuning back the abstract, because its a page long
- Figuring out how i want to format this diary
- Getting unity to work with this gitlab repo

Final goals:

- A game with some form of PCG, a character to navigate with, and enemies to defeat
- A detailed report on all the complexities of the above task, especially noting all the Design Patterns

---

29/09/2022

I have modified my initial draft of my project plan, replacing the risk table for a more verbose risk assessment section. I have inserted a gantt chart, and further broken down my almost biweekly/triweekly time plan into a weekly breakdown, bar 2 weeks, in which coding large sections of the final game may take much more time and effort.

02/10/2022

Handed in my second draft of my project plan to my supervisor, with much more detail on what I will be using to do my research, what proof of concepts, reports and test tools I will be writing, and have written notes next to each of my milestones as to why each one is important. I will commit it to my Gitlab once it has been looked over tomorrow, in my meeting with Hugh.

03/10/2022

Meeting notes

Two things to note

Abstract:
- Acts like a pre-summary, 2 paragraphs
- Sentence 1: purpose of the paper
- Sentence 2: what it contains

Bibliography:
- References and bibliography could serve as 1

04/10/2022

Uploaded my current unity files to the repo, realised that it was trying to sync my Library files, which unity generates itself, to the repo as well, which was trying to commit 10k file changes, which is not ideal, so a .gitignore was also added to make sure there was no issue. Was quite scary to see 10k additions. Turns out it was only about 150 files instead, and I don't think i even needed to commit my logs, or user settings folders, but figured since nothing online mentioned it, i would simply commit them anyways, i can remove them later if they become an active issue.

05/10/2022

Managed to complete a proof of concept, I have all basic motion and collision sorted, I now just need to focus on research and further proof of concepts until I have a prototype. The prototype should be super required until my interim, but the proof of concept is good for my next meeting with Hugh. Next bit of research I will focus on is likely the major elements of a roguelike game, just so I can have everything I need in front of me when developing proofs of concepts, because in theory all proofs can be derived from that bit of research in some form.

07/10/2022

Made some tweaks to accommodate for a slight bug in the collision on my proof of concept, and I understand now why the bug existed too.

10/10/2022

I have done some product research that I will upload very soon, as it's almost finished. I am currently also working on a document that would outline the theoretical algorithms and structures I would be able to apply to my project, allowing me to place my thoughts onto a single document that I can go back to whenever I need. I may have both of these documents converted into markdown, for a similar format to what I have in this diary here. These bits of research can later be translated to LaTeX, but i currently use VsCode to write this diary, which syncs to the gitlab repo for me, and if i'm writing my history into that repo, writing my research bit by bit is doing exactly that, rather than writing into google docs and transferring it over in pdf form when im finished.

18/10/2022

After 8 days of little to show for myself I have reminded myself I need to actually update this diary and the gitlab in order for progress to be made. Doing things on my own machine doesn't mean much if the history can be seen, so I am uploading the current state of notes into the notes section. I may have to reorganise them later if I add them in the future.

19/10/022

I have finalised the rotation method for enemies to invoke, in order to track the player. It was very difficult, and required me to learn about quaternions, which are really not very fun to learn about either, i will need to explain these in my vive, but roughly, its a method to rotate a 3D object using 4 dimensions, x,y,z, and an additional w imaginary number. It is not directly linked to Euler rotation with degrees, and I need to convert Euler rotations to Quaternions to use them on my transforms.

21/10/2022

I have begun work on a damage system for the projectiles that the player and the enemies can shoot. Main issue right now is that the projectiles will also be able to collide, and i have realised that my code is less than ideal modularity wise, so i will need to a small refactor ro make sure i am not depending on single classes for many functions, this is not how Dave taught me, and i intend to correct for this.

25/10/2022

Meeting Notes

Hugh mentioned that I have no real evidence of formal testing, which is true. I need to outline how I am doing my testing, as I have actually been doing it, but it isn't very obvious at all. Essentially, since I cannot isolate my scripts from the game environment, I will need to write tests that I can perform in the game, therefore acting as unity tests. I do also need to start writing my document in which i breakdown my required design patterns, a UML may go well with this.

28/10/2022

I have finished the damage system, and finally gotten around to updating my diary, again, as I had forgotten. The refactor is next on the list to do, which I should be able to do very soon. I will need to work on a simple item system within the next 2 weeks. This is a very achievable task.

01/11/2022

I had to take a brief break due to personal matters, but I have started to construct the documents requested of me, beginning with the testing document, outlining what I need to perform in order to ensure elements of my code that I cannot test automatically work.

04/11/2022

I am currently working on several courseworks, so work on this project has been slow, but I have written most of my testing document, and now am writing my design patterns document. The resources stipulated by the brief have been very useful.

08/11/2022

Due to other courseworks, my attention had been diverted to other important submissions, but I can now continue with the design patterns document for my meeting later today with Hugh. I have created the shell of my interim report, placing the headers where I think they should be.

Meeting Notes

I should divide my paragraphs into theory and application to stand the best chance of describing what I intend to describe concisely. I have a habit of being excessively verbose when I don't think I've put the meaning of what I've written across very well. The design patterns document could do with some diagrams and some areas of the test document need clarity. I also need to focus on referencing everything I can, re-digging up any websites or papers I have read.

12/11/2022

I have decided it is likely important to push back my item system into the second term/winter break in order to complete my interim submissions. This time will be addressed in my evaluation of my planning. I need to start working on diagrams for my design patterns, but for now I am writing up all the theory I have learned for this project. I will likely be tight on time, but I will be able to manage.

22/11/2022

I have been writing my interim report non-stop for over the last week, from the 14th-ish, and it has been read by Hugh, who I had a meeting with today to discuss it.

Meeting notes:

- There are a few sections I do not need right now, I need to focus on aligning my time with the actual mark scheme.
- I need to reference everything i write, nowhere near enough references
- Software engineering section is good, but needs references and how I'm using each pattern clearly.
- Need to clearly mark what i've actually achieved
- I need to stop focussing on game design as opposed to software design.
- Reformat my product analysis notes into a table, more useful that way
- Need to paste in this diary.
- Need to re-explain quaternions, and why i use them
- label every image and table
- provide definitions from literature to all precise terms
