Exploration into player engagement

Master's Dissertation project

Will Bennett 20014262k
B014262k@student.staffs.ac.uk, Shaun Reeves

Contents

Figures

Glossary

PCG – procedural content generation

GDD – games design document

IRDC - International Roguelike Development Conference

MUDs - Multi-User Dungeons

PENS - The Player Experience of Need Satisfaction

AC - armour class

Key words

Roguelike,

Procedural Content generation,

Player Engagement,

Replayability,

Game mechanics.

1. Introduction

Procedural content generation (PCG) is a key tool within the games industry with the benefits of its implementation being seen throughout the industry. This project is a conclusion to a series of explorations into the topic of PCG. Previous projects focused on creating a viable framework based off the concept of graph rewriting and cellular automata to create a populated dungeon environment. This framework will be used to create a game that sits within the roguelike genre, this is to achieve the aim of exploring key concepts of player engagement. Especially with the importance of replayability within said genre. The video game created will aim to implement these fundamentals and to analyse the importance and impact upon the player base to weigh the extent that players actually notice the reasons that they are playing the game.

1.1 Aim

Producing a game product which uses a previous procedural content generation tool prototype, this tool has been developed in previous modules.

1.2 Objectives

- Explore the fundamentals of what makes a game fun
- What are the key elements of player engagement?
- How effective are these strategies of achieving their goal?
- Apply these strategies to the game.

2. Project Methodology

This project will be managed through the use of agile methodologies (Wrike, N.D). GeeksForGeeks (2024) defined agile methodologies as a flexible development approach, through the use of an iterative tasks. To manage and maintain this approach, a kanban board will be used (Rehkopf, n.d.), which will allow for large tasks to be segmented and micromanaged in a visual way. These segments will have deadlines applied; this is to manage scope of these tasks while also making long term workload deliverable. To allow for this approach to be iterative, weekly meetings with a supervisor will be organised, this will allow for constant feedback and higher management of the tasks. This will ensure that the project will progress smoothly and will remain within the aim and objectives scope.

3. Background

Games developers often focus on which mechanics will be implemented to create the core gameplay loop. This can lead to a game that boosts impressive features which can be enjoyed by consumers of the media. While this approach may work, there are key aspects of games that are often overlooked. This pitfall can make games feel unfulfilling and cause players to drop the game before finishing it, if it makes it past testing. This can cause developers to wonder why the core mechanics within their game feel empty. This is where the importance of this project comes into play, as gaining an understanding of why game mechanics engage players, and which offer the largest impact in the consumer's enjoyment.

This project will use previous developments in a PCG (procedural content generation) tool to create a game that will feature within the roguelite genre. This tool was created to explore how PCG can be used to create a dungeon-like environment. It explored the concepts of cellular automata, graph rewriting, pre-authored rooms, entity spawning. All of this together was able to create a compelling environment inspired by Unexplored (2017).

4. Project Plan

This paper’s objective is to create a game which feature mechanics that will engage the player. This game will be in the roguelike genre and will use a previously developed tool to create the game’s environment. While the game will be planned to be a full game, to correctly scope for the paper, the game will be a vertical prototype. This will allow for the core gameplay to be implemented thus the game mechanics can be interacted with. To achieve this a game design document (GDD) will be created to ensure consistency within the game’s features. To obtain which mechanics will be used, research into industry opinion will be essential, this will then be backed with a questionnaire. This questionnaire will rate how engaging key mechanics are and get participants to state why. These together will allow for the GDD to make informed decisions for the development process of the game.

5. Literature Review

5.1 Defining Player Engagement

Player engagement can be defined in a multitude of ways thus it is important to define what player engagement in terms of this study. This term will refer to the level of interaction, enjoyment, and immersion that a player will experience during gameplay. This definition avoids any confusion from motivation, as motivation can be defined as a driving force to lead players to play the game or as "hooks" (Eng, 2023). "Hooks" often come under "evil" practices, especially within the mobile games industry with psychological tricks such as daily rewards (Game Maker's Toolkit, 2018).

Having defined player engagement, an exploration into how developers get players to interact with their games. This can be achieved through the use of game mechanics, Dhule (2022) states how mechanics can have generic applications such as platforming shared in the "Super Mario" series and "Sonic the Hedgehog" or have game specific applications. But no matter how they are applied they are the pillar stone to how players are lead throughout the game experience. Brazie (2022) agrees with this statement with his definition, as he adds how game mechanics can be impactful and fluff depending on the game and the consequence the mechanic creates.

5.2 Evolution of Roguelikes

To allow for deeper analysis of the how game mechanics relate to player engagement, narrowing down the topic to the roguelike genre will enable clear criticism to be created while avoid conflicting fluff that swapping genres can cause as mentioned by Brazie (2022). The roguelike genre came to fame through its namesake, a game called "Rogue" made by Epynx Inc (1985). Overtime the genre gained popularity, but with that popularity came exploration of the genre. This led to the genre becoming very wide due to lack of any definition. In 2008, a definition was created during the International Roguelike Development Conference (IRDC) (2008). This conference set out key factors or in our case game mechanics that were essential for a game to be classified as a roguelike. In addition to essential mechanics, some non-essential mechanics were set out such as ASCII art. Due to the strict definitions of roguelikes, games that were pushing the boundaries of the genre were coined as roguelike-like or roguelites. An example of this is "Spelunky" made by Mossmouth (2008), which took the key mechanics of the genre and added 2D platforming. Stegner (2021) points out that this trend continued throughout the early 2000s, particularly in the indie scene causing the roguelite genre to stay.

Nowadays both genres are commonly referenced as roguelikes, with titles like "Hades" by Supergiant Games (2018) being referred as roguelikes despite missing key elements stated by IRDC. Despite this roguelites are an important part of the rise of popularity of the genres, this is due to the ability to draw in a large audience with mechanics that bleed into other genres such as "Slay the Spire" by Mega Crit (2019) with deck building. Stenger mentions how this genre bleeding often causes higher engagement for newcomers by giving them familiarity in a notoriously punishing genre. Jong (2024) mentions how adapting these gameplay elements into the typical roguelike genre cycle of permadeath, the games often resonate with audiences and create a diverse and replayable experience, this strongly aligns with Stegner's view of how the genre became popular. Additionally, Brazie's concept of game mechanics becoming fluff depending on the game is integral to both Stegner's and Jong's exploration of the genres, with developers taking a large challenge to correctly create a game with consequences that are tailored to the meshed mechanics, thus creating an engaging experience.

5.2.1 Key Mechanics for Player Engagement

As mentioned, there are a wide variety of game mechanics that can be used within genres, with multiple bleeding throughout genres. An example of this is progression mechanics, what is important is to gain an understanding of key mechanics within our chosen genre. This will allow for the framework of the game to appear which can then be polished further by introducing extra mechanics to make the game stand out on its own. This can be used to achieve mechanics to optimally flow together for a cohesive experience. An example of this is Csikszentmihalyi's exploration of "Flow" (1990). Csikszentmihalyi (1990) explores the concept of optimal experience, this is often referenced as a "flow state". Csikszentmihalyi refers to the flow state occurring when players are focused on realistic goals which allows for action opportunity for players to use their matched skills. These opportunities allow for tasks to be solely focused on while other components are forgotten temporarily. This logic can be used on the application of mechanics together, as well as how the player will experience them. Additionally, the state of flow can be used to describe how the player will be engaged with a game.

5.2.1.1 Defining Replayability

Roguelikes have a key central mechanic/motif which is replayability. This mechanic is essential due to the nature of the gameplay and can be seen throughout the genre's lifetime. Despite this, there are multiple methods of introducing replayability to games. Bycer (2018) explores the concept through the idea of varied consistency. He explains this via biomes, which he defines as a collection of predefined elements within a game space. These biomes act as a way to provide a consistent experience for the player, as it acts as a world building. This is due to items or enemies can be linked with biomes.

Bycer continues this thought onto "The Binding of Isaac", he breaks down how it implements replayability by shuffling content between runs while keeping content consistent to keep the game feel the same. Some consistent content is controls, biome layout, room types, and end bosses, meanwhile shuffled content is the rooms used, enemies, items, biome variants, and regular boss variants. Bycer continues this by stating how all these elements stop the player from building a routine while expanding their knowledge of the gameplay. When implemented well, replayability ties in with Csikszentmihalyi's concept of flow states as content swapping can keep the player on their toes by providing new gameplay to challenge the player.

5.2.1.2 Introducing Randomness

Randomness is a mechanic that has its uses in variety of games, this is thanks to the concept of information horizons. Burgun (2014) explains this term as how long it is till the player gains unknown information, this can be visualised as fog of war – which hides information until the player explores it. This information horizon leans into player engagement as it causes players to think about possibilities such as in chess which can be whittled down to a game of who can look ahead the furthest based off current information. This technique can be seen throughout games but where does randomness relate to this? Burgun explains some keyways of implementing a horizon, hidden information, execution uncertainty, exponential complexity, and finally randomness.

Randomness can be implemented in a wide variety of ways, an example of this is enemy loot tables, or hit chances in RPGs. In relation to roguelikes randomness is implemented throughout the game, from the PCG application of levels such as Bycer's concept of biomes, or through the types of enemies that the player will face in a given run. As mentioned, Randomness goes hand in hand with replayability, with randomness giving variety to bolster the iterations.

In terms of mechanics this randomness can be split into two key types which Johnson (2014) called pre-luck and post-luck in a GDC about game transparency. Game Maker's Toolkit (2020) explains this is more commonly called input and output randomness. Input randomness is when a random event occurs before the player makes a decision, while output is randomness occurring after player actions. This definition is from Engelstein on the podcast Ludology (2018), he continues his description by stating how output randomness often undercuts any strategy the player may try to create. This idea is supported by Johnson, with both agreeing that it can seem unfair. This can draw player engagement out if implemented incorrectly as it can build frustration if it affects the player negatively based on luck.

This has caused a recent swing towards input randomness becoming the preferred implementation of randomness. This can be seen with Subset Games (2024), their game "FTL: Faster than Light" (2012) was released in 2012 and featured output randomness heavily meanwhile their newer published game "Into the Breach" (2019) featured purely input randomness. This can come down to the nature of the "Into the Breach" being a strategy game but allowing for long term strategies to form and play out for the player will cause increase engagement. This is due to the player is being rewarded for this play style.

Controlling Randomness

Adding input randomness to a game and calling it a day will not cause player engagement to increase. These mechanics have to be implemented correctly, this also applied to output randomness, which despite often being labelled as frustrating, can actually engage the player in its own way. Game Maker's Toolkit (2020) explains this concept with "Spelunky" (Mossmouth, LLC, 2008), with chests scattered across the map having a small chance of dropping high tier loot such as jetpacks. He furthers this idea by stating this can cause a blur between the run being good due to good luck and making runs without this good luck being perceived as pointless. Another issue input randomness has is a too short of an information horizon, which can deny any strategies from forming. To reduce this, it is important to consider a game's information flow. Hoeppner (2017), relates this to adding high impact information is added to the game at set regular intervals as seen in Figure 1. This can allow the player's plan to playout and be disrupted by a curve ball which can make them adapt, thus giving the satisfaction from planning and overcoming an unforeseen circumstance.

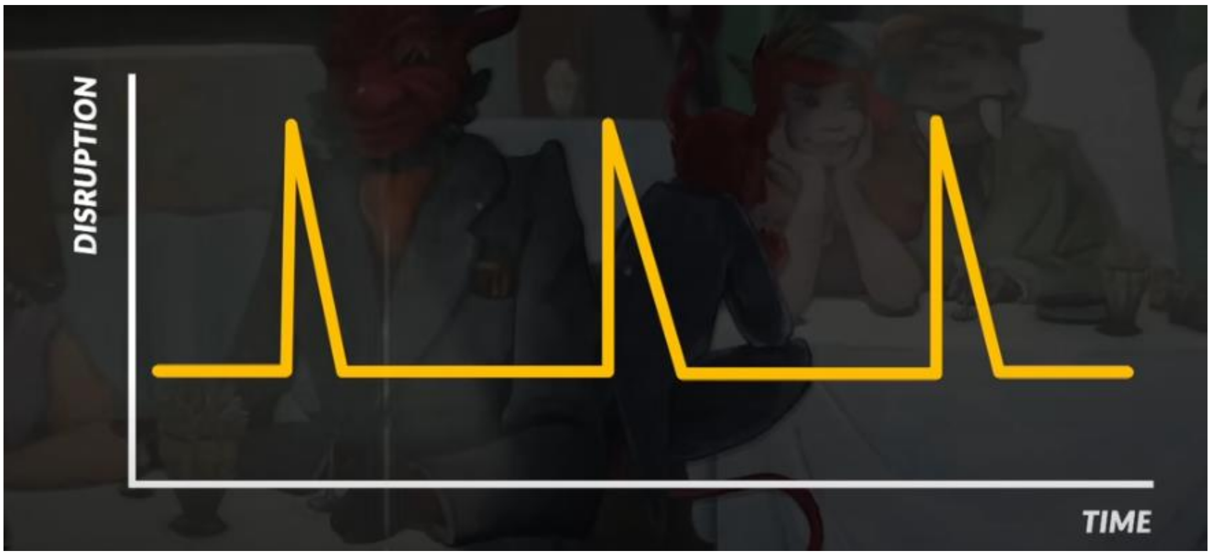


Figure 1: A spikey information flow (Game Maker's Toolkit, 2020)

Game Maker’s Toolkit (2020) goes onto the reasons to use output randomness, with the core reason is to implement an abstract combat system, where the player tells units what to do such as the “XCOM” (Firaxis Games, 2024) series or “Darkest Dungeon” (Red Hook Studios, 2016). Additionally, to soften the negative impact that output randomness can have, telegraphing the success rates of risks they will be taking. This additional information can allow the player to gain an understanding of the underlying systems and make informed decisions. On the other hand, output randomness can be used to benefit the player. Game Maker’s Toolkit (2020) points out the only output randomness system that “Into the Breach” has benefits the player. This occurs when an enemy hits a building within the game, this has a rare chance to trigger a resisted effect which means the building takes no damage. This system cannot be relied on due to its rareness but is a nice surprise when it triggers. Justin Ma a co-designer for “Into the Breach” states on Eggplant: The Secret Lives of Games podcast (2019) that this was implemented as no one complains when something good randomly happens.

5.2.1.3 Roguelike Progression

Roguelike progression is a hotly discussed topic, with there being two key approaches. As previously mentioned, roguelike and roguelites are quite similar genres. There is another difference in each genre which is how they handle progression. Roguelikes are well known for the permadeath approach, with no persistency between runs. Meanwhile roguelites, usually have some sort of persistency through permanent upgrades.

Game Maker’s Toolkit (2019) explores this concept and how it effects the game itself. He explains how roguelikes challenge the players skill and understanding of the game mechanics. This can create a scenario where players are unable to beat the game. To keep the player engaged, developers often implement systems to give players boosts. This has already been mentioned with “Spelunky” and input randomness in chests having a chance to give you high level loot early, additionally there is a questline that allows you to skip the first level once completed. This stops the player from seeing the “true” ending but allows for easier access to later levels. Game Maker’s Toolkit also bring up the implementation of unlockable variety or cosmetics as a way to implement a feel of progression to keep players engaged. This can be seen in “Enter the Gungeon” by Dodge Roll (2016), which allows to you buy more weapons with a consistent currency. These guns are not better than the ones the player starts with but can cause some glee when they appear in runs. On the other hand, roguelites do not have an issue with players not being able to complete the game due to progression being permanent, the game technically gets easier. This is where a major issue occurs, Game Maker’s Toolkit explains this as an uncertainty of skill, where the player is unsure if they are getting good at the game, or they have sunk enough time to unlock enough abilities. He continues this thought with how roguelites can often become a genre where players only have to spend enough time in the game to beat it. This does not create an engaging experience for the player, so methods must be implemented to counteract this feeling. One method implemented in various games is skill-based spending. In “Dead Cells” by Motion Twin (2018), players can on spend their upgrade points at key segments in the run. This means the player must have enough skills to reach these upgrade rooms since all points are lost on death. This assures the player that they are learning the game, while reducing the skill required to progress. This is furthered as the further the player progresses the more currency they earn, this rewards higher skilled players as they can save up their currency to spend it all at once at a later shop. This strategy adds depth to the experience, thus expanding on their engagement. This tangible progression allows for players to avoid the feeling of wasting time on failed runs which often happens during roguelikes.

5.2.1.4 Power of Difficulty Curves

All games have some sort of difficulty attached to them, whether this is the laid-back experience of "Stardew Valley" by ConcernedApe (2016), the fast pace challenging action of the "Doom" franchise from id Software (2016), or the procedural difficulty often found in roguelikes. Each of these provide the player with different experiences to keep the player engaged and to challenge their growing knowledge and skills. This is to stop the player from being bored from the gameplay becoming stale. Csikszentmihalyi (1990) discussed an optimum state to keep the player within to allow for full immersion and enjoyment. This term was coined as a flow state, which can be seen in Figure 2, where the player should be kept within the flow channel to maximise engagement.

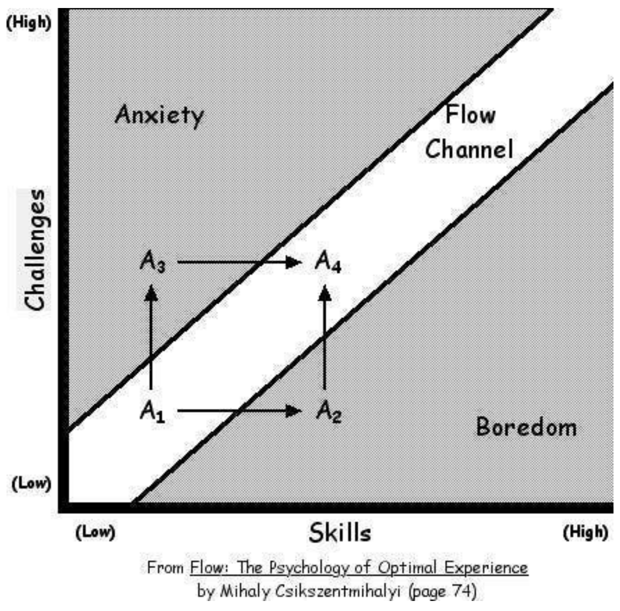


Figure 2: Diagram of Csikszentmihalyi flow channel (Icodewithben, 2023)

With that in mind, there is a key mechanic that can be applied to keep the player challenged. This is difficulty curves, which act as a way to format challenges for the player based on their progression through the game. Bycer (2021) agrees with this sentiment as he goes onto to state that difficulty curves provide an appropriate challenge throughout the game when done correctly which in turn helps reduce player frustration. These curves can be represented as literal curves, Larsen (2010) discusses some of these shapes. Each curve has their own purpose with some being straight linear curves which gives constant increase of difficulty, or fixed increase wave which increases the difficulty in spikes or waves, or interval difficulty that has peaks and troughs, or logarithmic which showcases a large increase initially then levels off as the game reaches max difficulty which can be seen in Figure 3. These techniques can be combined such as an interval logarithmic wave, which causes a wide variance in increasing difficulty that levels out as seen in Figure 4. This concept can be combined with Hoeppner's (2017) concept behind spiking information flows, in doing so the difficulty curve can have spikes in it. Strachen (N.A.) disagrees with the concept of plain difficulty curves, as he explains that a difficulty saw can present a challenge to the player while easing them into new mechanics as seen in Figure 5. He continues by explaining these new mechanics can be a new enemy type, puzzle, abilities, or the combination of mechanics. While he states that the introduction of new mechanics should lower the difficulty, it should lower it based on the current difficulty. This will stop jarring difficulty jumps to stop the player from getting uninterested in exploring the mechanic. This idea can ease players into new mechanics thus keeping them engaged and being turned away from interacting with it due to the difficulty punishing them from learning.

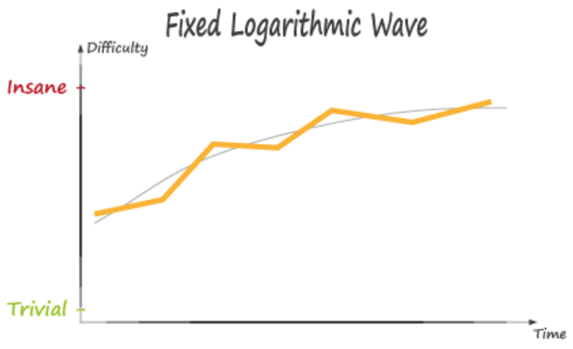


Figure 3: Fixed Logarithmic wave (Larsen, 2010).

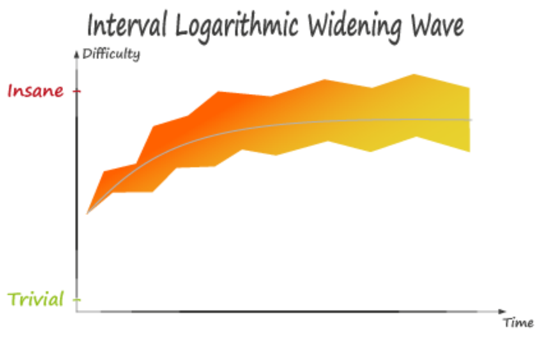


Figure 4: Interval Logarithmic Widening Wave (Larsen, 2010).

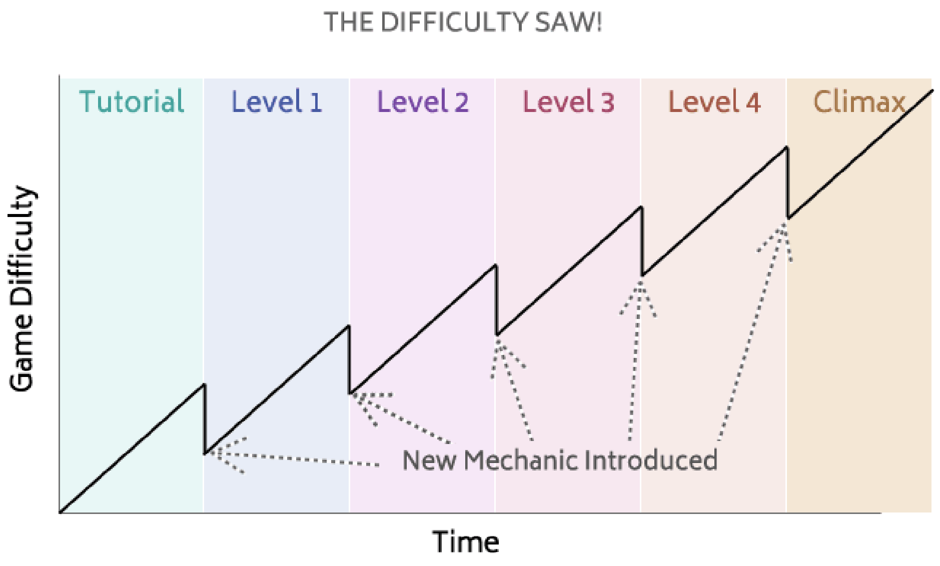


Figure 5: Difficulty saw image (Strachan, N.A.)

Now that difficulty curves have been explained, which ways can games get these diagrams into their games. One perfect example of this is “Sekrio Shadows Die Twice” by FromSoftware (2019), in this game the player has to face hard bosses with a series of attack patterns. After defeating the boss, the player unlocks new areas to explore with each area getting harder which challenges the player as they progress. This is a typical linear difficulty curve, but FromSoftware twists it with a difficulty saw. This is achieved by making the early enemies in the area weaker than ones just fought in the previous one.

Not only does this give the player a break from the intense fighting, but it gives a sense of progression due the player feeling stronger. This can keep the player engaged as it saves them from burning out on the intensity but gives them the satisfaction of growing their abilities. This is one-way developers can apply the curve to the game. Some others can be via resources, such as making healing potions appear less in areas to challenge the player or vice vera, or as Strachen states introduce new mechanics, or enemy abilities. Eaves (2021) continues this idea by stating that designers have a multitude of ways to influence difficulty with the key idea of keeping players on their toes to by introducing new or randomised mechanics. This can stop players from anticipating challenges while testing their knowledge, but this has to be done in moderation to avoid frustration and to break player engagement.

Since difficulty curves are tightly linked with progression through the game, it is important to consider how these curves often look for roguelikes and roguelites. Game Maker's Toolkit (2019) depicts what these difficulty curves could look like when linked to player progression seen in blue in Figure 6 and Figure 7. Figure 6, showcases the point discussed with a flat difficulty curve can cause some players to never finish the game if they do not pass the skill ceiling of the game. Meanwhile Figure 7, shows how permanent progression can cause the curve to unnaturally go down leading to victories which can be unsatisfying. Both of these showcase potential dangers of unmanaged difficulty curves which may drive players away due to poor engagement with the games systems.

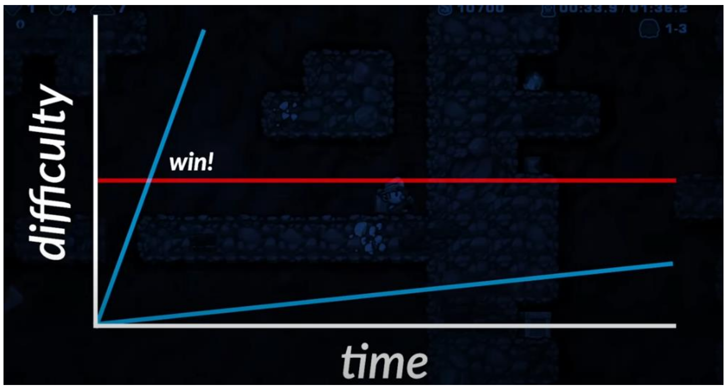


Figure 6: Typical roguelike difficulty curve (Game Maker's Toolkit, 2019)

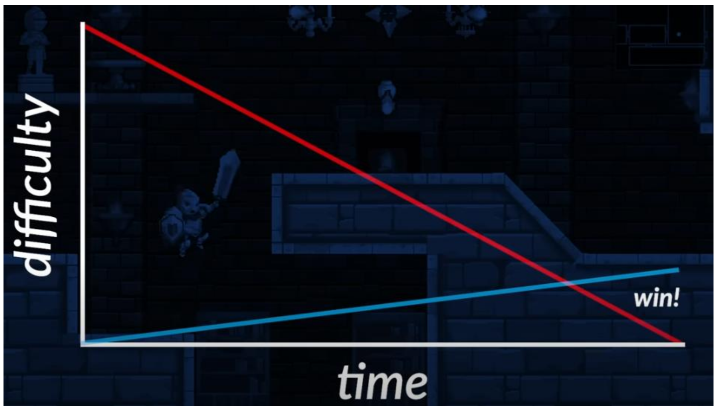


Figure 7: Typical roguelite difficulty curve (Game Maker's Toolkit, 2019)

5.3 Applying Mechanics to Target Player Types

5.3.1 Player theory

Now that some core mechanics have been fleshed out, being able to apply these mechanics to maximise their effectiveness is vital. This is where Bartle taxonomy of player types theorized by Richard Bartle (1996) aides. This theory was created to categorise the player types for his game Multi-User Dungeons (MUDs), and it does this via their preferred play styles and motivations. Bartle splits player types into four sections, Achievers who are players that are focused on in-game achievements and acquiring levels and rare items. Explorers who are driven to explore and learn about the environment, game world, and the mechanics that come with it. Socializers who enjoy player interaction, and mechanics linked with role-playing and immersion. And Killers who are motivated by competition, player vs player type combat, and imposing distress. While these categories general encompass players, Bartle states that players can move into different categories based on mood or the goal of the game but would spend most of their time in one category. As Bartle's theory gained in popularity it became overused as both Bartle and Kyatric (2013) state. Kyatric and Bartle discuss how the idea of categorising players into four sections draws people in and they try to implement the theory in non-MUD activities or apply it without fully grasping the concept. This discussion ends on the idea that Bartle taxonomy should be used to remind developers of the potential player types and how players will psychologically interact with the game environment. By using the theory, players may be more engaged as mechanics are being delivered in a way that encompasses their needs.

Another theory that can be used to categories player theory is The Player Experience of Need Satisfaction (PENS) by Ryan, Rigby, & Przybylski (2006), this theory is founded on the Self-Determination Theory (SDT). The SDT theory states that humans have three fundamental psychological needs; autonomy, competence, and relatedness. PENS takes this further by relating it to video games and how they fulfil psychological needs. This theory has scales to measure this such as intuitive controls, presence, and in-game relatedness. Intuitive controls are how the player interacts with the game itself through the control scheme and how easy a player can interact with the game's mechanics. Presence is the sense of player immersion within the environment, in reference to physical, emotional, and narrative elements. In-game readiness scales player connections to the game world and its inhabitants, with a focus of social interaction in the experience. By using PENS, a developer can evaluate how effective their game is at appeasing their players as it provides an understanding of motivational factors. This level of understanding can lead to more engaging experiences. While both theories function under the similar idea of satisfying players, PENS is able to evaluation its effectiveness. This allows for multistage checks throughout development. On the other hand, both theories can be used together to further the developers understanding of their player base and how to achieve an engaging experience. This is due to Bartle taxonomy can be used as a blanketing way to categories players, and then PENS can be used to evaluate the effectiveness of the strategies used.

6. Importance of Game Mechanic strategies

This test was using a questionnaire to gather data around key mechanics explored in the literature. Each participant will be asked for a rating on how engaging each mechanic is and give a reason behind the rating. This will allow for an exploration into the participants experience with the mechanics and how impactful these are onto their gameplay experience.

6.1 – How important is a large amount of content in keeping you engaged?

Overall, the consensus is that players enjoy a large amount of content to get into. Based on the responses given when asked why, usually the idea of more content allows them to be entertained for longer. That being said, they usually link this concept to the content being higher quality. The lower ratings coming from the sense of being overwhelmed. With over 60% of the participants giving a rating of 4, this showcases tat large amount of content is beneficial for player engagement, but this comes with a condition that said content is high quality and is presented is a manageable way. If applied in such a way to support the player's progression, the player may be lured to play more with the promise of more content if they continue to play and progress. This goes against Game Maker's toolkit concept of roguelikes suffering from a feeling of being unbeatable, as this promise can make the game feel beatable with the potential to gain access to beneficial content. This comparison does not undermine Game Maker's Toolkit but explores how complex the concept of promising the player content and enabling their progression is.

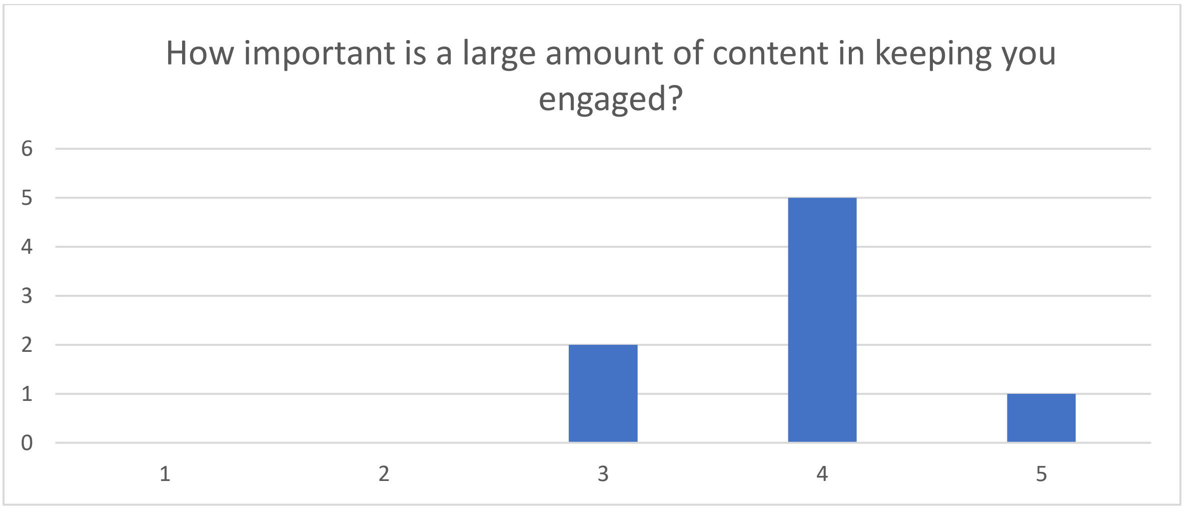


Figure 8: Bar graph showing data relating to the question How important is a large amount of content in keeping you engaged?

6.2 - Is variety of content more important than quantity of content?

Compared to quantity, players seem to value variety more. This is represented via the number of 5 ratings increasing by 4. Some qualitative responses indicate that games often throw repetitive quests at users such as fetch quests. These are used to give the player something to do rather than entertain them. One participant stated that variety is good, but it has to be given in a manageable way, this is to avoid overwhelming the player with too many options. This could be an issue due to each quest being different, which in turn makes them all seem important especially when compared to lots of similar quests. The concept of giving the player the varied content in a manageable way aligns itself with Bycer's (2021) explanation of biomes. These biomes will act as an isolated way of introducing players to new content in a way that is repeatable. This can take pressure off designers, which can then be focused on other aspects of the design of the game such as assuring the quality of the content. Another statement that was intriguing, was mentioning ADHD, and how providing variety of content allows for refreshing gameplay thus keeping said players engaged.

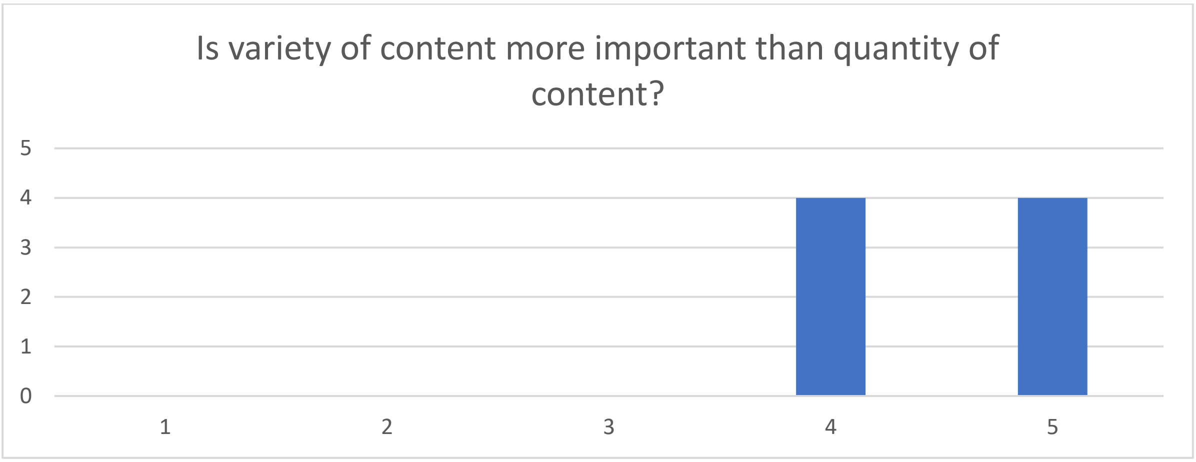


Figure 9: Bar graph showing data relating to the question "Is variety of content more important than quanity of content"

6.3 - How important is hidden information in keeping you engaged?

Hidden information was met with a wide range of ratings, with most participants giving a rating of 3. This was due to the uncertainty of how it would be applied, some participants stated that a key balance between known and hidden information is important. While other participants stated that withheld information can cause frustration. Most participants mentioned how it was not a defining "feature" of the game. While hidden features can cause frustration which definitely draws players away from the game by breaking that engagement, Bycer's concept of using biomes to provide consistency can be helped to reduce this frustration. This is because giving players a game design rule to learn about the environment they are in can enable them to pick up on and use on biomes later in the game. While may help reduce frustration, a large majority of participants stated how they do not interact or knowingly interact with hidden information. This can be used to inform that players can often overlook some mechanics and misunderstand how it is impacting their gameplay experience.

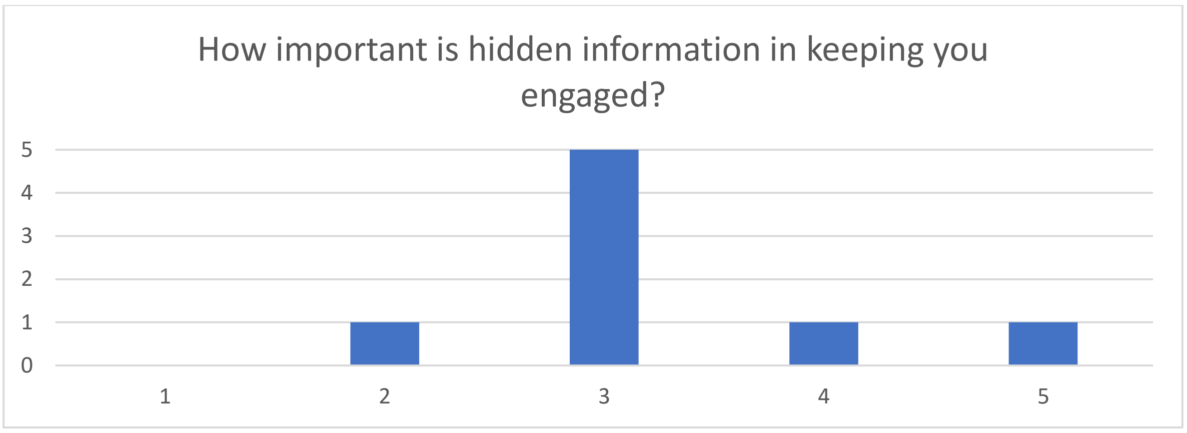


Figure 10: Bar graph showing data relating to the question "How important is hidden information in keeping you engaged?"

6.4 - How important is execution uncertainty in keeping you engaged?

Once again, the majority of respondents gave a rating of 3. Most respondents gave this rating as they acknowledged the benefits it can provide through breaking plans and challenge the player's ability to react to new information. They often countered this idea with how much uncertainty they player will face. As if the player's plans keep getting disrupted then annoyance can build. One respondent stated how they were unsure on how this mechanic would apply to the games that they play. While not entirely useful, it is important to acknowledge that some mechanics can fly under the radar to players or not even be used. This data makes Burgun's (2014) concept of information horizons being able to keep players engaged uncertain, as the majority of participants are unsure on the mechanic. As stated, this uncertainty mainly came from how it would be implemented, which can be solved with play tests of how the game would be implementing it.

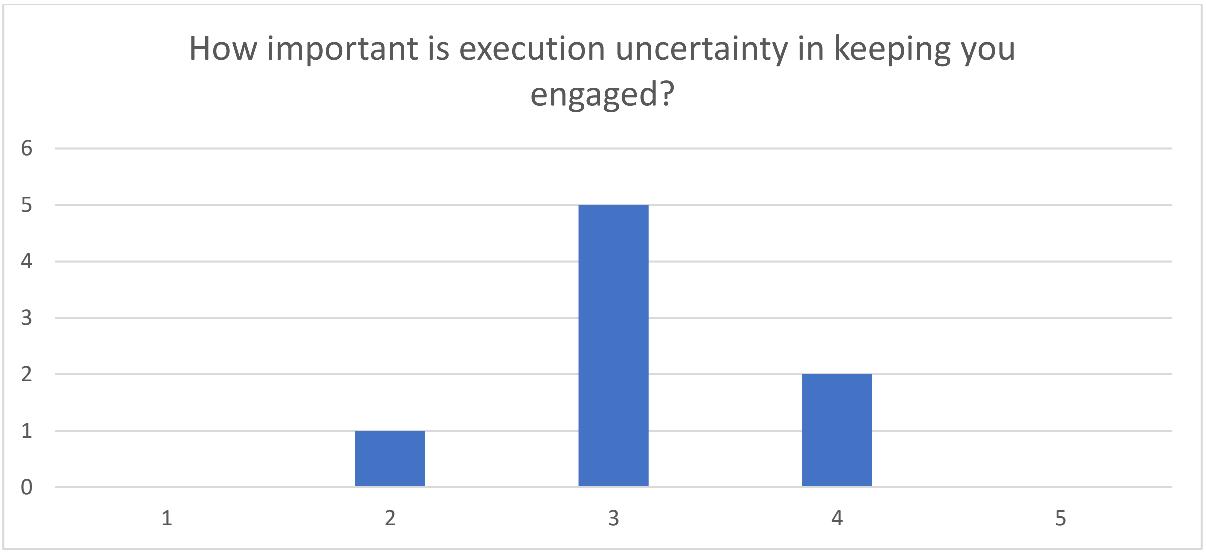


Figure 11: Bar graph showing data relating to the question "How important is execution uncertainty in keeping you engaged?"

6.5 - How important is Procedural Content Generation (PCG) in keeping player engagement?

PCG had a varied responses from participants, with the majority being 3 and above. 25% of participants gave a rating of 1, with their reasoning being a preference for hand crafted content, and solely seeing it as a developer sided tool. While preferring hand crafted content is a valid reason, some more understanding into how PCG works in games may provide improved ratings. Most participants that gave a rating of 3 stated how PCG's impact depends on the genre, or how both non-PCG and PCG levels provide enjoyment. Higher ratings mention how it aides in replayability, with the downside of it not aiding in single playthrough as much in terms of engagement. This supports Bycer's (2018) explanation of how "The Binding of Isaac" implemented replayability into its generation using consistent areas. Since "The Binding of Isaac" is a roguelike, the implementation McMillen used was founded in PCG to enable high amount of variety between runs.

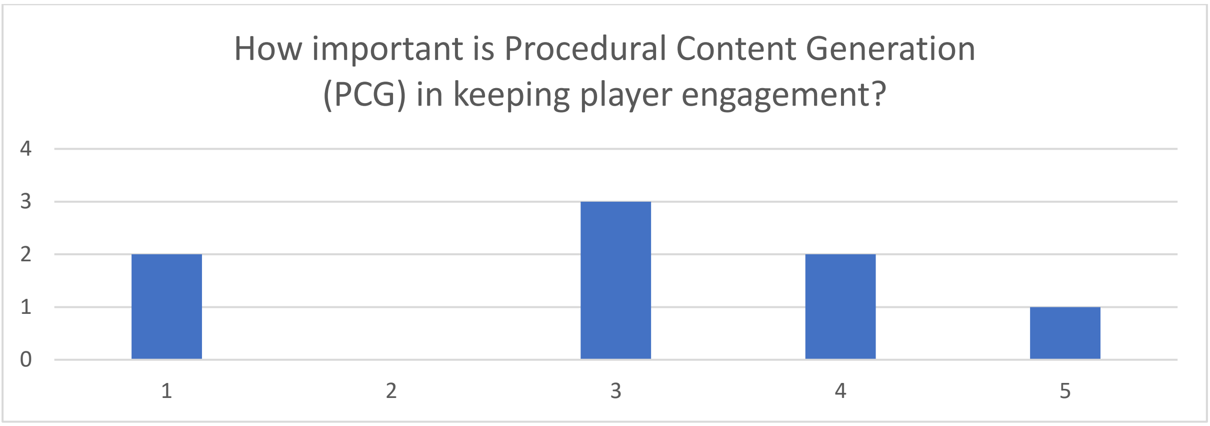


Figure 12: Bar graph showing data relating to the question "How important is Procedural content generation in keeping player engagement"

6.6 - How important is input randomness in keeping player engagement?

Overall, the majority of votes was for 4, with 4 votes for it. Most participants gave higher ratings due to the interesting gameplay that can come out of input randomness and its effects on replayability. This supports the concepts that Johnson (2014) discussed in his GDC talk, with the general finding being that players prefer due to it benefiting them and they can plan long term. As stated in the literature review, Subset games (2024) created "Into the Breach" (2019) with mainly input randomness effecting the gameplay. Subset Games discussed this decision to due long-term strategy is enabled which the participants stated is as a reason why input randomness keeps then engaged. With that combined with replayability, it gives players power over their choices throughout their gameplay allowing for different approaches in different playthroughs. Meanwhile, lower ratings came from the fear of frustrations that comes with randomness or with "metas" being formed from favourable scenarios or builds appearing. The effect of this can be reduced with correct designs of mechanics, to reduce the chances of frustration and to offer enough viable variety that players are not forced to use certain outcomes to progress.

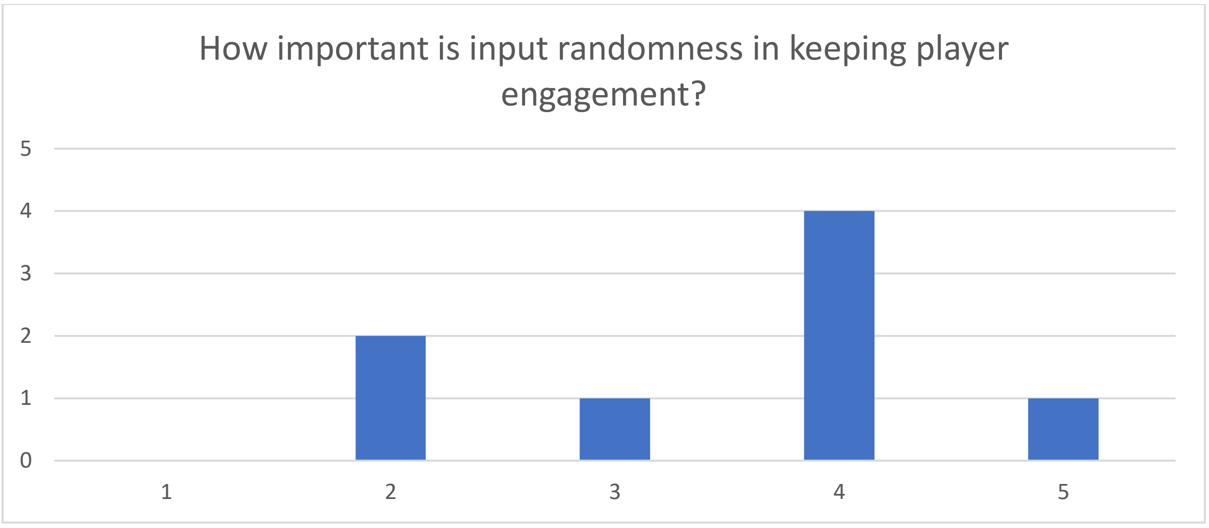


Figure 13: Bar graph showing data relating to the question "How important is input randomness in keeping player engagement?"

6.7 - How important is output randomness in keeping player engagement?

Output randomness received a lower overall rating than input randomness, with 75% of participants giving a rating of 3 and below. These ratings come from the idea of players enjoyment being disrupted, such as plans being ruined by random events, which can cause an unfair feeling. Some participants stated that this feeling was reduced by introducing mechanics that guide players expectations such as hit chances. These opinions were mirrored throughout the literature review with Johnson (2014), Ludology (2018), and Game Maker's Toolkit (2020) all stating that players often feel cheated when an event out of their control negatively effects the outcome of their gameplay. Which led to the implementation of hit chances in "XCOM" (2024) or using output randomness effecting the player in a positive way as seen in "Into the Breach" (2019) as Justin Ma (2019) stated. The participants that gave higher ratings highlighted the engaging process of being forced to adapt to different scenarios. This was stated with the condition of applying this mechanic in moderation, to avoid deterring players.

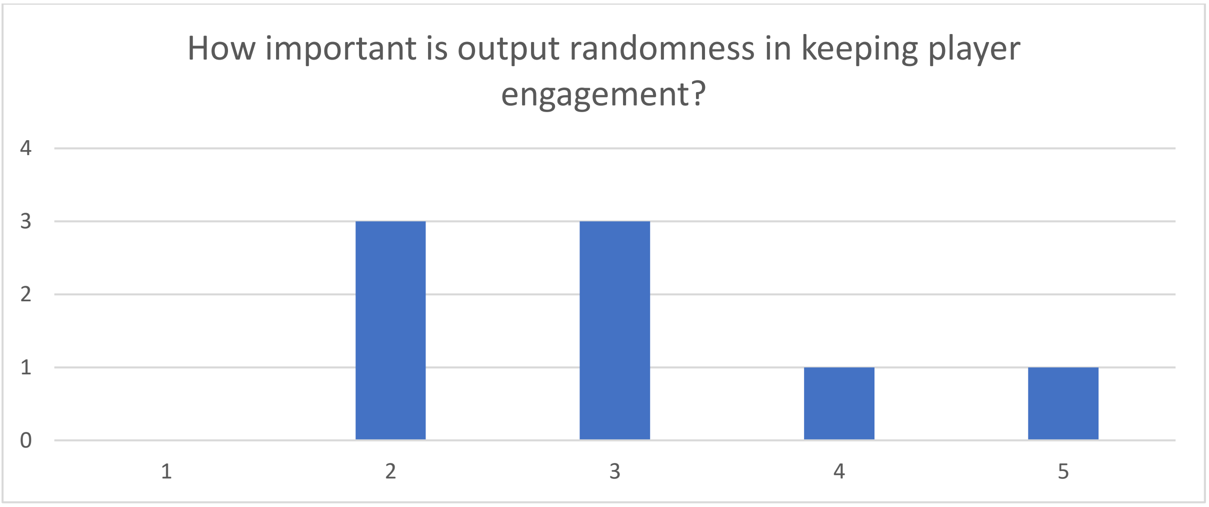


Figure 14: Bar graph showing data relating to the question "How important is output randomness in keeping player engagement?"

6. 8 - How important is the sense of progression in keeping player engagement?

Only one participant did not give a rating of 5 for the importance of the sense of progression. This unanimous high rating shows how important this mechanic is, with participants stating that is gives players a sense of purpose to play the game. This is usually followed by how tightly this mechanic is linked to the players enjoyment. This may be due to how the player can easily reflect and see how much progress they have made. Most participants state how different games achieve this feeling which shows how important this mechanic is that it is featured across multiple genres from first person shooters like COD with their prestige system or Minecraft's open world progression through getting better gear and achievements systems. Some participants also state how without this system games often feel tiresome and sluggish which leads to playing the game for short bursts.

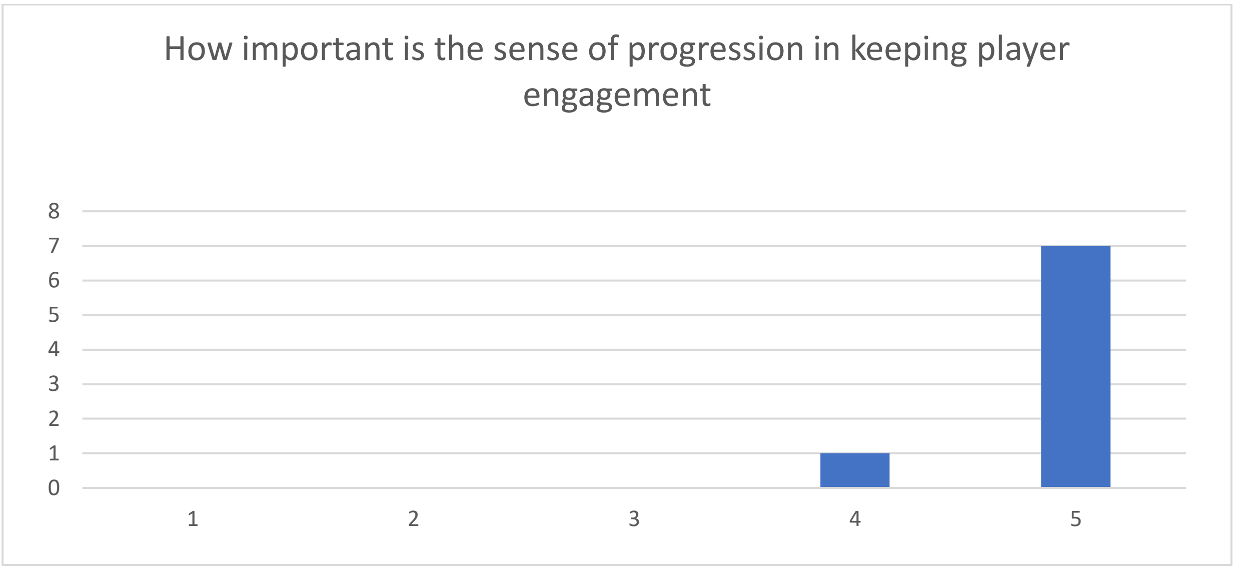


Figure 15: How important is the sense of player progression in keeping player engagement"

6.9 - How often do you often notice difficulty curves in games?

Half of the participants gave a rating of 3, with an equal balance either side. The reasons for giving 3 were conditional, with participants stating they notice it with large spikes or drops, which one stated they noticed it when coming up against harder bosses. This trend of players noticing the difficulty when the game progresses, is also shared with reasonings in higher ratings.

The participant that gave a rating of 5, stated that they often notice difficulty curves because they are actively paying attention to them, this may be to the fact that they are interested in the gameplay. Meanwhile, participants that gave lower ratings stated that they often notice the difficulty curves depending on the genre of game they play. Additionally, how the game progresses can make the curve more noticeable. This is in reference to the player gaining knowledge of the game's mechanics and being more proficient in playing the game. This can cause the player not to notice if they are not able to reflect on how their skillset has changed since the beginning of the game. This could be solved by using a difficulty saw as Strachen (N.A.) states it allows players to feel more powerful. These design choices can allow the players to feel the difficulty curves without noticing them, which in turn feeds back into the sense of progression and keeping themselves engaged.

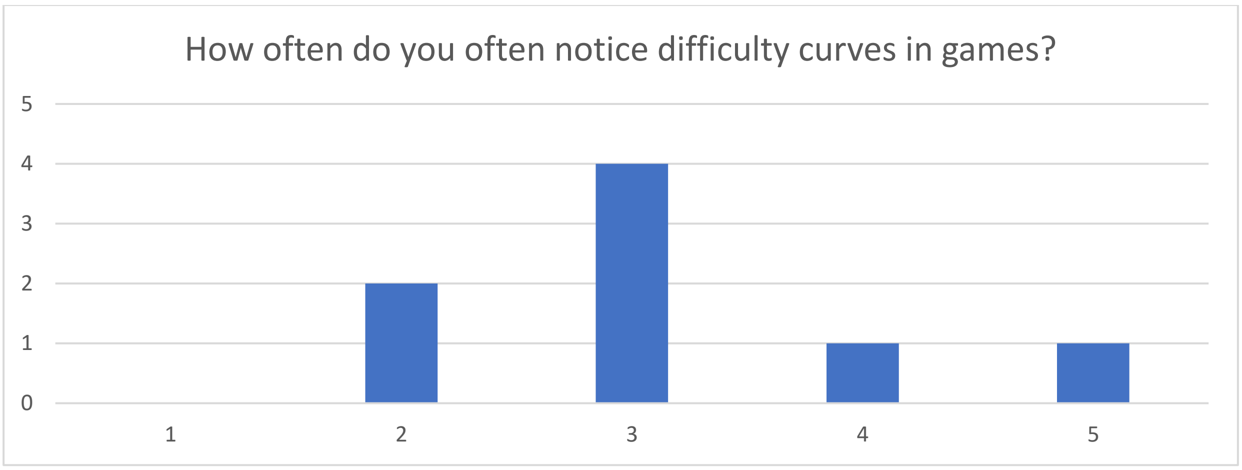


Figure 16: Bar graph showing data relating to the question "How often do you notice difficulty curves in games?"

6.10 - When playing Roguelikes do you often feel like the game is unbeatable? - When playing Roguelites do you often feel like the game lets you win through unlocking upgrades?

Both question 10 and 11 focus on the rogue genres, the graphs show the spread Figure 17 and Figure 18. These two graphs show very different trends, with it focusing on how players feel about the main pitfalls discussed by Game Makers Toolkit (2019). Figure 17, shows two extremes for the participants with it 50% of participants giving a rating of 4 and above while the other half gave ratings 2 and below. This graph represents how often roguelikes feel unbeatable, this could be explained by the participants skill in the genre. As often the genre features transferable skills. But this clearly supports Game Makers Toolkit's opinion of Roguelikes being unbeatable. This opinion is also supported by the graph shown in Figure 18. This is due an even spread of 2 votes from 2 to 5, this shows how roguelites can often fall into the trap of enabling the player to succeed eventually. This is supported by the vast majority of votes being above 3. Although, this could be supported by player skills in the genre.

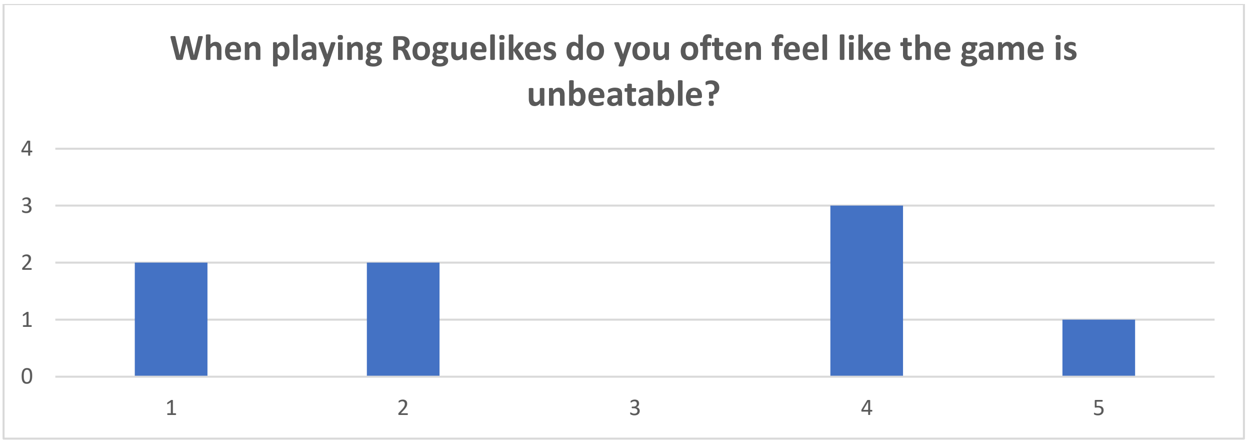


Figure 17: Bar graph showing data relating to the question "When playing roguelikes do you often feel like the game in unbeatable?"

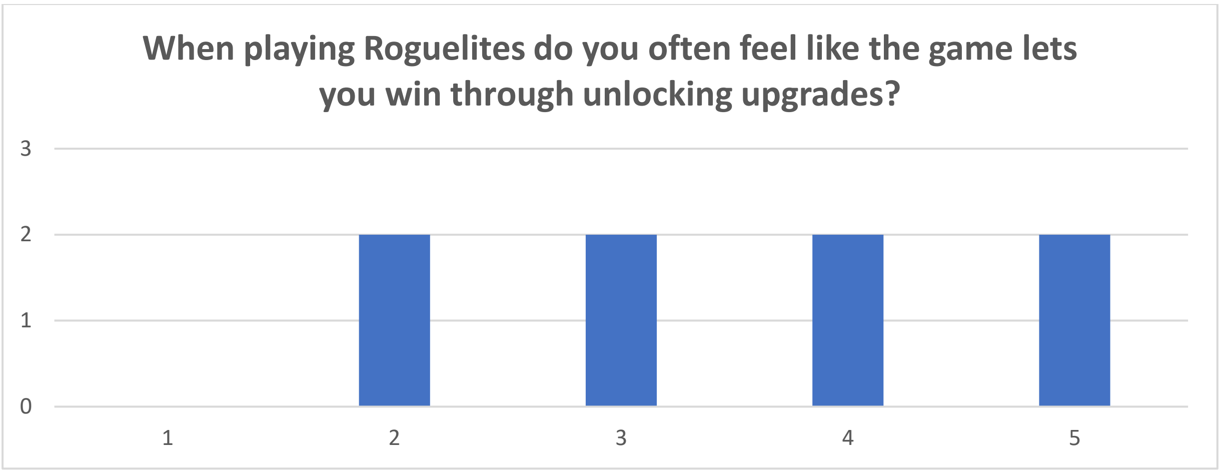


Figure 18: Bar graph showing data relating to the question "When playing roguelites do you often feel like the game lets you win through unlocking upgrades?"

6.11 Overall

Participants gave varied answers to the questions, but the majority agreed with each other. This proved that the majority of the mechanics presented to them would lead to players being more engaged if it was included. That being said, some participants were unsure on the importance of some mechanics such as PCG, this could come down to lack of knowledge of the mechanic and how its used within games, or not playing games that actively use this mechanic. Furthermore, participants also disliked mechanics such as output randomness. With many giving scenarios where the mechanic would cause frustration and break them out of the flow state (Csikszentmihalyi, 1990). That being said, the majority of participants agreed with the literature used to create the questionnaire. With this in mind the game's mechanics can be created based on the researchers' opinions, while also using the primary data to tweak any mechanics to suit the consensus of the data to appeal to the genres purpose and engage the player base.

7. Design

7.1 Game Design Document

Game design documents are a detailed guide that describes what your game is and how it will work (French, 2024), French continues this idea with how controversial GGDs can be due to them quickly becoming outdated as the project progresses. Despite this, one will be used to create a high-level plan for the project. This will act as a way to track the progress of the development and inform decisions when creating a vertical prototype of the game.

The GDD can be seen in the appendix, in the section named 11.1 Appendix A - Game Design Document. While the game design document was created in reference to this study, the scope of the project described within it larger than the scope of this project. This allows for the future of the game to be planned so project has the potential to be continued in the future. Despite this, the majority of the core gameplay loop will be created for this project. This is to create an accurate vertical prototype.

7.2 Requirements

Based off the GDD, the game will have a range of requirements based off the engagement questionnaire. Below are core requirements in relation to the gameplay loop.

Requirement	Purpose of requirement
Ability to generate dungeon.	The game is based off this dungeon, the game will have to generate correctly to allow the player to play and interact with the core game loop.
Working inventory system that allows for the player to gain and equip armour, which applies its stats.	This allows for the player to store gear and equip it to prepare for the boss fight.
Working deck system that has a limited card count and allows the player to gain and place cards.	This is the main way for the player to interact with the dungeon. These cards act as a way for the player to progress towards the boss fight and defend their dungeon.
Player is able to hover over items, tiles, or cards and see information about it.	This will allow the player to make informed decisions about the actions they make. This will enable plan making and reduce the initial learning curve of the game.
Adventurer has the ability to enter combat and win or lose.	As the enemy of this game, adventurers' main purpose is to fight the minions in the dungeon. If they win the combat they will progress further towards the player, if they lose the player will be rewarded.
Player is able to enter combat and win/lose	This is the loss condition, as if the player loses the dungeon cannot progress. If the player wins combat against the adventurer they will gain equipment, unless it is the boss which is the win condition.
Working vertical slice.	The game should have key core gameplay to allow for game mechanics to be implemented.

Figure 19: Table of requirements the vertical slice's gameplay loop should meet.

8. Implementation

The game will feature the mechanics researched; this implementation will allow for the exploration of these mechanics. Using these mechanics in the development process, should enable the game to be more engaging for players, especially considering the demographic that was sampled for the questionnaire.

The game itself was described in the games design document in Figure 47. To summarize, the game will feature a tower defence inspired auto combat, with deck building allow the player to customize the map. In the context of the game, the player is a dungeon master working under an overlord to conquer the realm. Opposing this is guilds spread across the realm, these guilds are filled with adventurers who want to take down the dungeons and stop the evil overlord. In terms of the gameplay, the player will have a deck of cards to build, which can be used for the defence of the dungeon. The dungeon is made up of three parts in a loop, this can be seen in Figure 20. Upon placing a card, the tile will change permanently, and will increase the attention of the dungeon. If the dungeon gains too much attention the player will have to fend off a party of heroes which acts as a boss fight.

- Pathway (Grey)
    - The actual circuit of the dungeon where the player will spawn minions and the adventurer will follow to the player. These tiles will mainly spawn minions to aide in the defence of the dungeon.
- Path side (Green)
    - Next to the path and has special cards that can apply in combat effects, these tiles can also spawn certain minions.
- Terrain (Brown)
    - These act as buffs for the player-sided entities. These buff stats and indirectly affect combat.

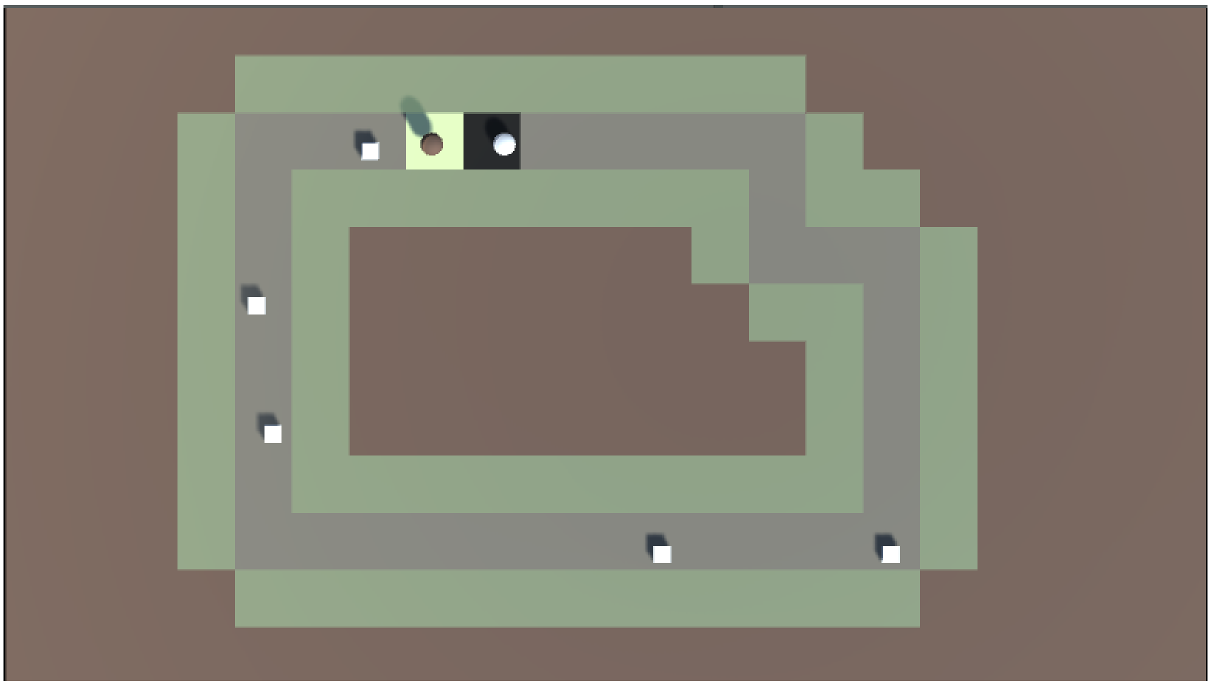


Figure 20: Showcasing of the dungeon map

The adventurers will walk into the dungeon and fight the minions that spawn from the dungeon. This walk in area is the black square in Figure 20, the adventurer will then move along the path to where the player is which is the yellow square.

8.1 Difficulty curves

The game's core gameplay features a struggle for the player as adventurers endlessly attacks their dungeon. To counter act this the player must place down defences which will increase the attention rating of the dungeon which will cause the boss to spawn. This creates a balancing act for the player to farm enough adventurers to get good armour and tiles, while not waiting too long as the more adventurers they defeat, the harder the boss fight will be. This struggle between preparedness and difficulty engages the player to create a plan for the long term while managing short term goals.

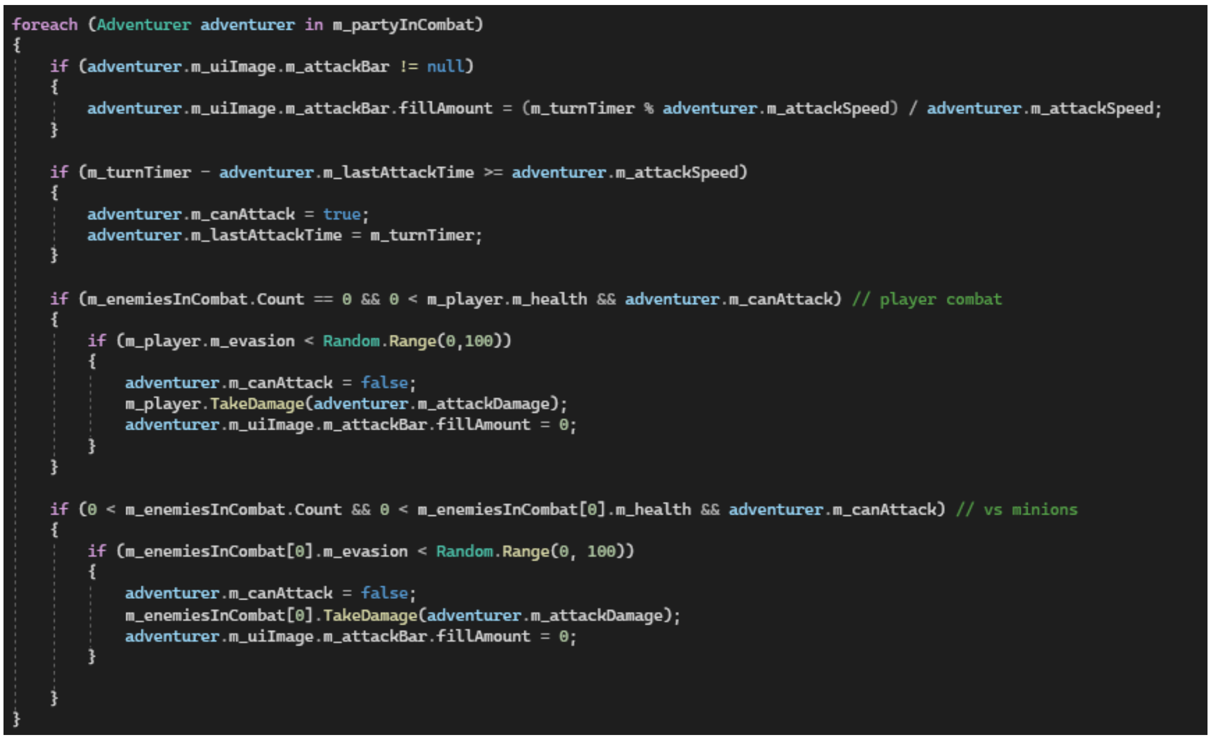
This execution uncertainty creates an opportunity for the player to have wide scale plans which they can narrow the later they get into the playthrough. An example of this would be which armour build they will aim for. Since adventurers will drop equipment randomly, the player will have to gamble and manage their equipment correctly to build up enhancements that will work cohesively together. This allows Csikszentmihalyi's (1990) flow state to slowly build as the challenge of piecing together an equipment build that works will become harder the further the player gets into a run, but the player will become more competent in this as time goes on. To counter act this competence to keep the player engaged and actively thinking, they will have to manage their dungeon layout and so it can defend against increasingly difficult adventurers. As mentioned, this difficulty will scale with the number of adventurers defeated, with the adventurers' base stats increasing. While this would create a linear curve for difficulty, to create a spike curve an array of adventurers will be attacking the dungeon. This will cause slight variation between the base stats with some having higher health while others having higher attack speed. This will keep the player on their toes and give them relief when a weaker adventurer is spawned.

8.2 Output randomness

The majority of participants agreed that output randomness if implemented incorrectly will cause increased annoyance at the game, this matched Johnson (2014) and Engelstein (2018) opinions with them stating it can undermine the players plans and build frustration due to "bad luck". To reduce this, output randomness is not a major component of the game. Despite this it still features within the combat system in one stat, evasion. This can be seen in Figure 21 to Figure 23, with a random number is generated and if the entities evasion stat is larger than that number they will dodge the attack. While this does not directly benefit the player as adventurers can also make use of this stat, the player has a way to universally increase this stat through terrain tiles and to upgrade their own character through equipment dropped by adventurers. While combat happens often, this control over evasion allows the player to take advantage of the output randomness which should reduce the frustration caused.

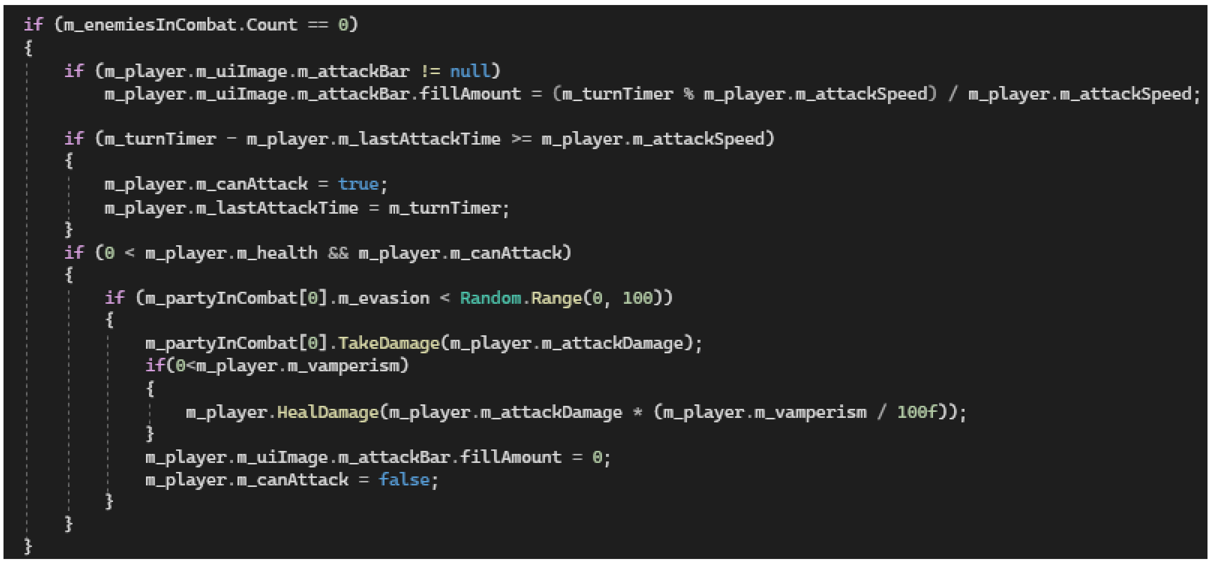
8.2.1 Combat

Combat is implemented in a simply way of entities attacking based off turn timers. An example of this is seen in Figure 21, where every adventurer in combat is looped through and checked if they are able to attack, via the turn timer and their attack speed. If they are able to attack a check is performed to see if they are in combat with the player or minions. The entity being attacked will then try to evade the attack based on their evasion, if this fails, they will take damage based off the adventurer's damage. This will then reset the adventurer's UI attack bar. This is similar to the players attack phase seen in Figure 22, and the minion's attack phase seen in Figure 23. Except one key difference of a vampirism stat. Due to the player and minions being monstrous evil beings, they have access to a life stealing stat, which heals the entity by a percentage of the damage they have done. With more healing being done with a higher vampirism stat. Figure 24, showcases an additional stat that affects combat, which is AC (armour class). This reduces the amount of damage the entity will take based off its AC stat.



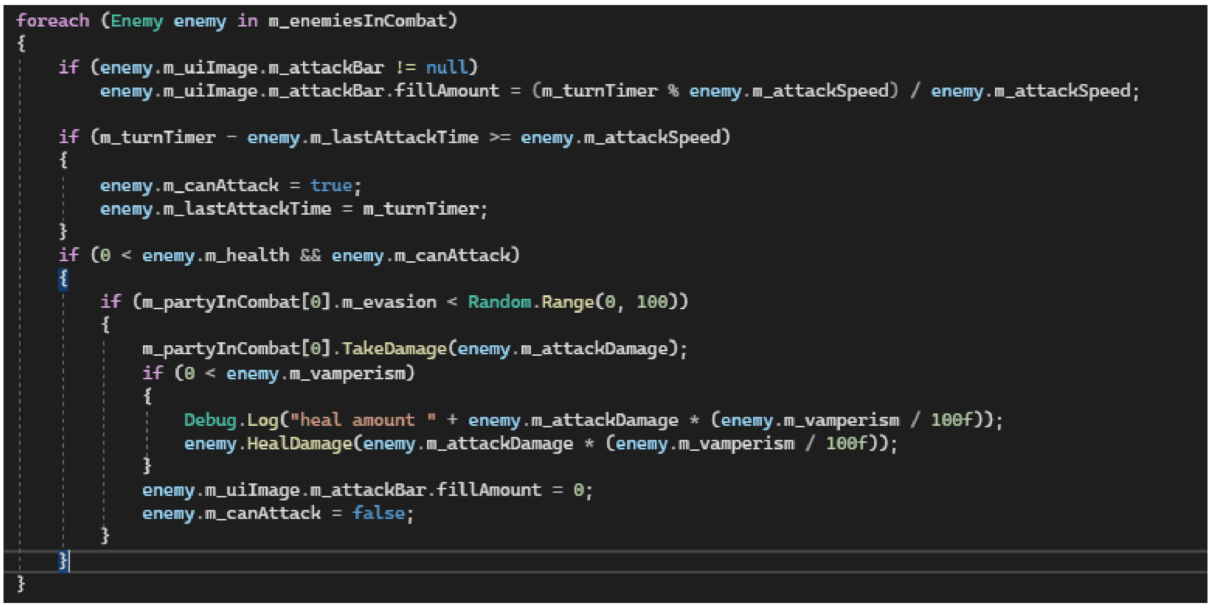
```
foreach (Adventurer adventurer in m_partyInCombat)
{
    if (adventurer.m_uiImage.m_attackBar != null)
    {
        adventurer.m_uiImage.m_attackBar.fillAmount = (m_turnTimer % adventurer.m_attackSpeed) / adventurer.m_attackSpeed;
    }

    if (m_turnTimer - adventurer.m_lastAttackTime >= adventurer.m_attackSpeed)
    {
        adventurer.m_canAttack = true;
        adventurer.m_lastAttackTime = m_turnTimer;
    }

    if (m_enemiesInCombat.Count == 0 && 0 < m_player.m_health && adventurer.m_canAttack) // player combat
    {
        if (m_player.m_evasion < Random.Range(0,100))
        {
            adventurer.m_canAttack = false;
            m_player.TakeDamage(adventurer.m_attackDamage);
            adventurer.m_uiImage.m_attackBar.fillAmount = 0;
        }
    }

    if (0 < m_enemiesInCombat.Count && 0 < m_enemiesInCombat[0].m_health && adventurer.m_canAttack) // vs minions
    {
        if (m_enemiesInCombat[0].m_evasion < Random.Range(0, 100))
        {
            adventurer.m_canAttack = false;
            m_enemiesInCombat[0].TakeDamage(adventurer.m_attackDamage);
            adventurer.m_uiImage.m_attackBar.fillAmount = 0;
        }

    }
}
```

Figure 21: Adventurer combat phase within AttackPhase function in the CombatManager.



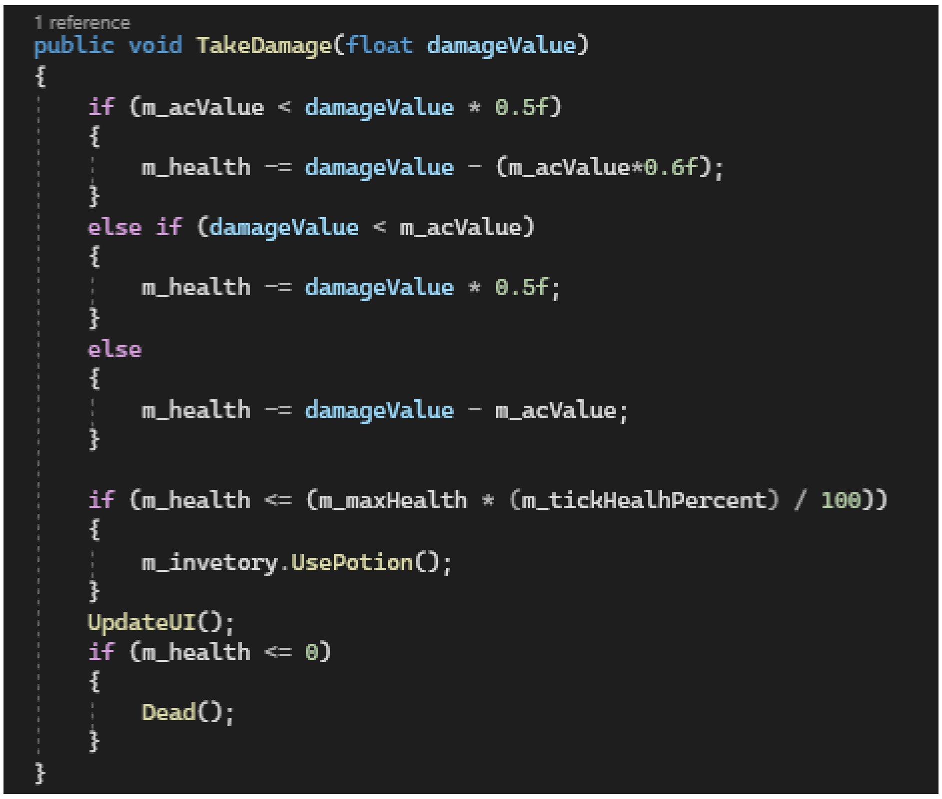
```
if (m_enemiesInCombat.Count == 0)
{
    if (m_player.m_uiImage.m_attackBar != null)
        m_player.m_uiImage.m_attackBar.fillAmount = (m_turnTimer % m_player.m_attackSpeed) / m_player.m_attackSpeed;

    if (m_turnTimer - m_player.m_lastAttackTime >= m_player.m_attackSpeed)
    {
        m_player.m_canAttack = true;
        m_player.m_lastAttackTime = m_turnTimer;
    }
    if (0 < m_player.m_health && m_player.m_canAttack)
    {
        if (m_partyInCombat[0].m_evasion < Random.Range(0, 100))
        {
            m_partyInCombat[0].TakeDamage(m_player.m_attackDamage);
            if(0<m_player.m_vamperism)
            {
                m_player.HealDamage(m_player.m_attackDamage * (m_player.m_vamperism / 100f));
            }
            m_player.m_uiImage.m_attackBar.fillAmount = 0;
            m_player.m_canAttack = false;
        }
    }
}
```

Figure 22: Player combat phase within AttackPhase function in the CombatManager.



```
foreach (Enemy enemy in m_enemiesInCombat)
{
    if (enemy.m_uiImage.m_attackBar != null)
        enemy.m_uiImage.m_attackBar.fillAmount = (m_turnTimer % enemy.m_attackSpeed) / enemy.m_attackSpeed;

    if (m_turnTimer - enemy.m_lastAttackTime >= enemy.m_attackSpeed)
    {
        enemy.m_canAttack = true;
        enemy.m_lastAttackTime = m_turnTimer;
    }
    if (0 < enemy.m_health && enemy.m_canAttack)
    {
        if (m_partyInCombat[0].m_evasion < Random.Range(0, 100))
        {
            m_partyInCombat[0].TakeDamage(enemy.m_attackDamage);
            if (0 < enemy.m_vamperism)
            {
                Debug.Log("heal amount " + enemy.m_attackDamage * (enemy.m_vamperism / 100f));
                enemy.HealDamage(enemy.m_attackDamage * (enemy.m_vamperism / 100f));
            }
            enemy.m_uiImage.m_attackBar.fillAmount = 0;
            enemy.m_canAttack = false;
        }
    }
}
```

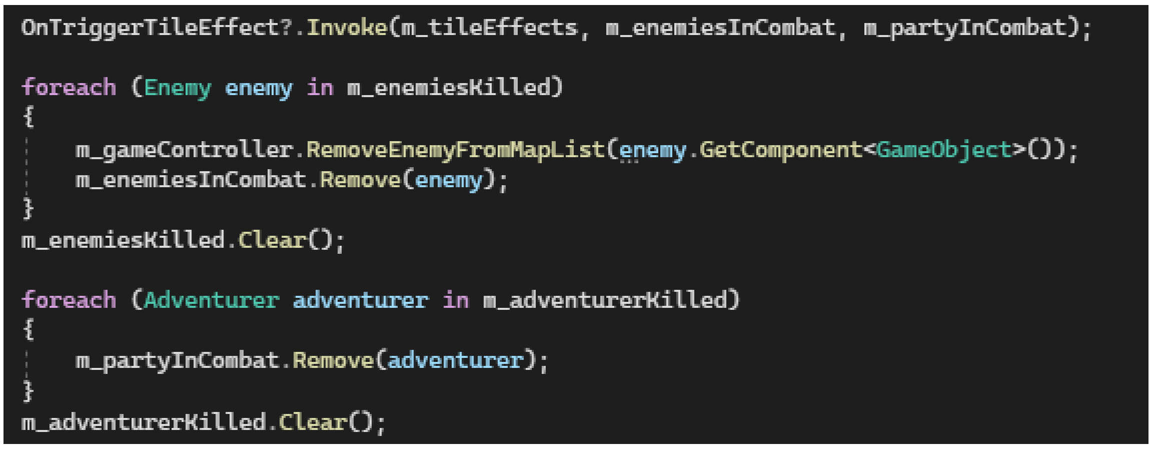
Figure 23: Minion combat phase within AttackPhase function in the CombatManager.



```
1 reference
public void TakeDamage(float damageValue)
{
    if (m_acValue < damageValue * 0.5f)
    {
        m_health -= damageValue - (m_acValue*0.6f);
    }
    else if (damageValue < m_acValue)
    {
        m_health -= damageValue * 0.5f;
    }
    else
    {
        m_health -= damageValue - m_acValue;
    }

    if (m_health <= (m_maxHealth * (m_tickHealhPercent) / 100))
    {
        m_invetory.UsePotion();
    }
    UpdateUI();
    if (m_health <= 0)
    {
        Dead();
    }
}
```

Figure 24: TakeDamage function within the Player script.

After all entities in combat have taken their turn, tile effects occur which is explained in 8.6.1 Tile effects. After this any dead entities are removed from the active combat list to speed up the process of combat.



```
OnTriggerTileEffect?.Invoke(m_tileEffects, m_enemiesInCombat, m_partyInCombat);

foreach (Enemy enemy in m_enemiesKilled)
{
    m_gameController.RemoveEnemyFromMapList(enemy.GetComponent<GameObject>());
    m_enemiesInCombat.Remove(enemy);
}
m_enemiesKilled.Clear();

foreach (Adventurer adventurer in m_adventurerKilled)
{
    m_partyInCombat.Remove(adventurer);
}
m_adventurerKilled.Clear();
```

Figure 25: End of each "round" of combat within AttackPhase function in the CombatManager.

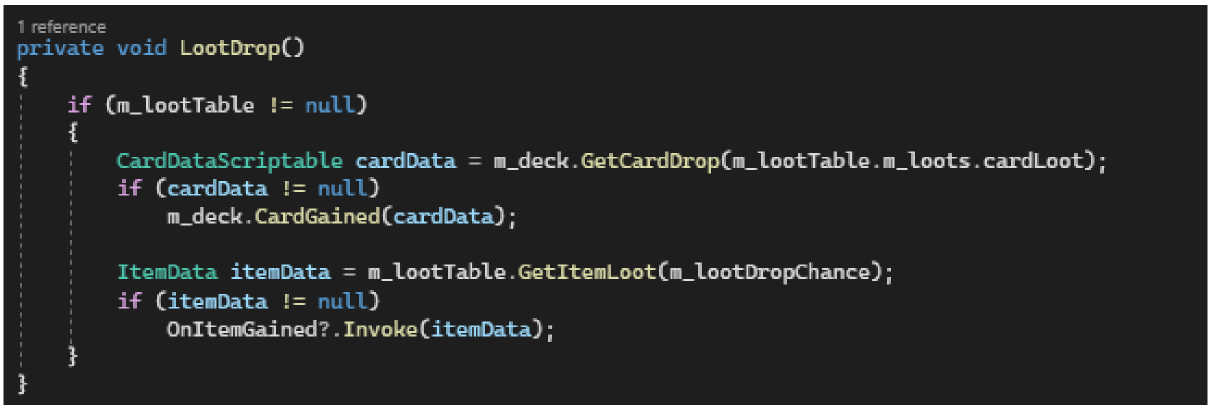
8.3 Input randomness

Johnson (2014) defined output as randomness occurring after player actions, he followed this up with stating that players often prefer this randomness as they are able to form long term plans. This was backed up above with the primary data gathered, this made it a great choice to engage the players to form long term plans within this game. While typically input randomness is found within turn based games such as “Enter the Breach” (2019), due to the combat of this game, the player gets to constantly make decisions and act out this plan. To aide in this, when the player hovers cards or items the game will pause, this is to reduce frustration when acting out said plans.

While equipping items and placing tiles do not seem to require skill, the player will have to be careful as these actions cannot be undone. This allows the player to be punished if they do not correctly plan out a layout of the base or equips an item that does not suit the build they are going for. On the other hand, the punishment aspect allows for the player’s long-term plans to feel more rewarding, which in turn encourages the player to engage with the game in the long term.

8.3.1 Loot drops

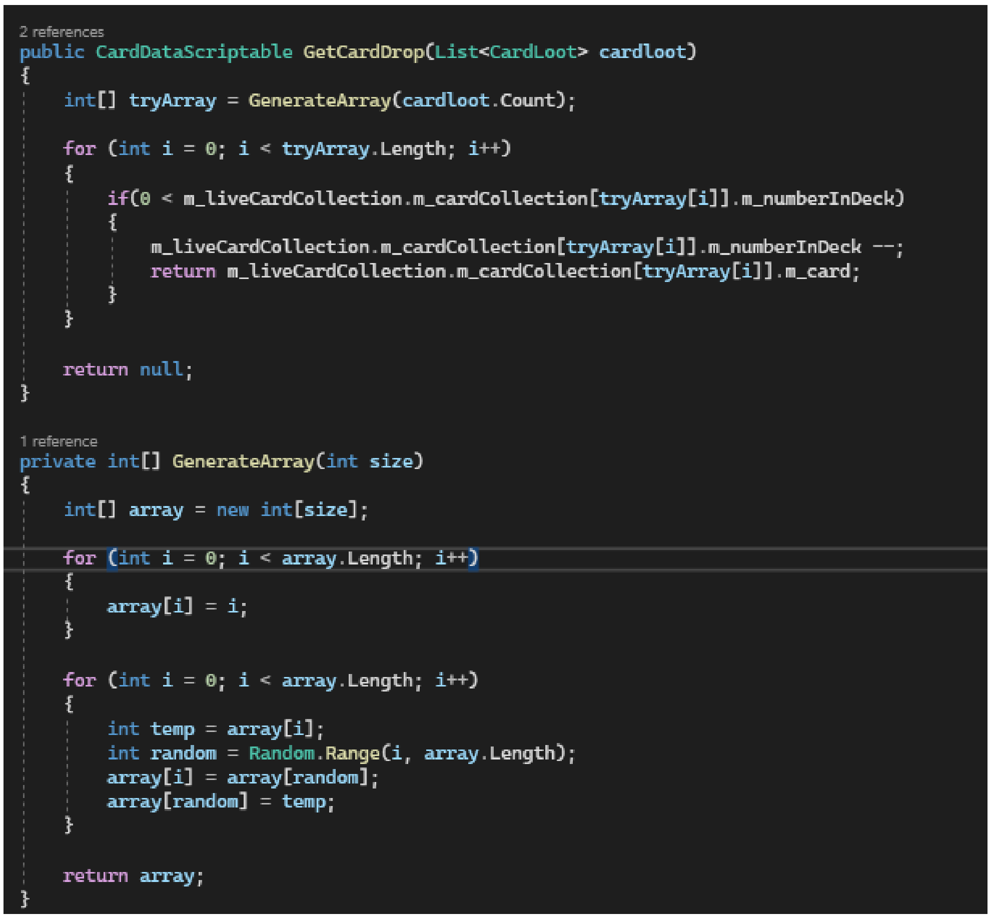
When adventurers and enemies die, they will drop loot from their linked loot table, Figure 29 shows an example of a basic adventurer. When the entity dies the function LoopDrop seen in Figure 26, gets called. This manages two separate loot drops: cards for the player to place and equipment. Adventurers will drop the majority of equipment the player will find; this will be discussed in 8.3.2 Inventory.



```
1 reference
private void LootDrop()
{
    if (m_lootTable != null)
    {
        CardDataScriptable cardData = m_deck.GetCardDrop(m_lootTable.m_loots.cardLoot);
        if (cardData != null)
            m_deck.CardGained(cardData);

        ItemData itemData = m_lootTable.GetItemLoot(m_lootDropChance);
        if (itemData != null)
            OnItemGained?.Invoke(itemData);
    }
}
```

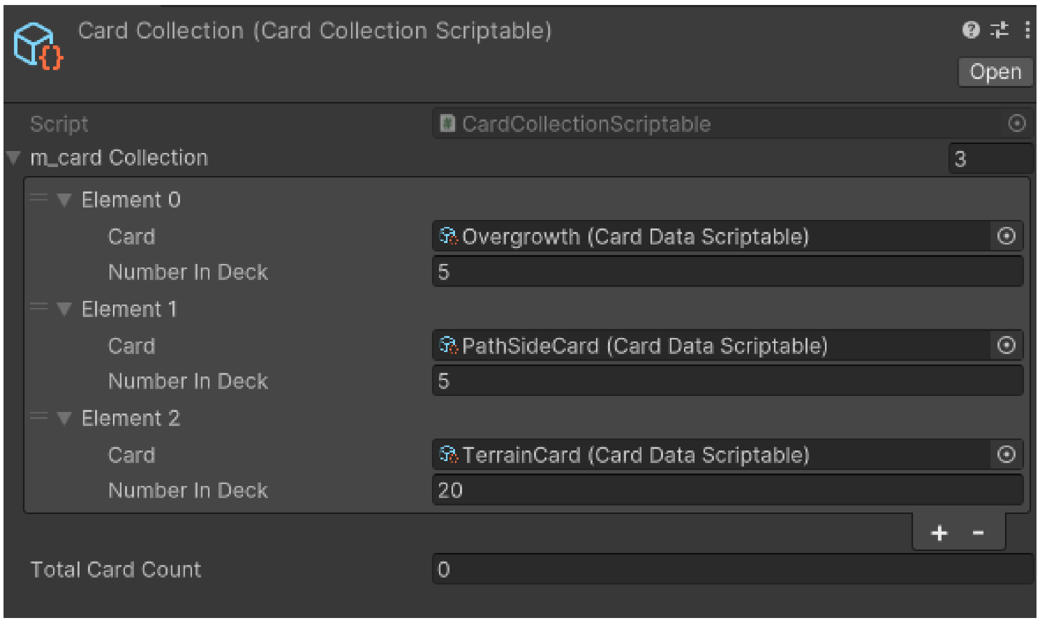
Figure 26: Function LootDrop in Adventurer script, showing two functions being called based on loot table linked

As seen in Figure 26, to get the card drop a list of cards is sent to the deck script, this list of cards is stored within the loot table linked to the entity. The GetCardDrop function can be seen in Figure 27, this function generates an array of random numbers depending on how many cards are within the loot table. This array will then be used as an index within the deck cards, this will check to see if there is more than one card at the random index, this is seen in Figure 28. If there is more than one card, then that card's count will reduce. This card will then get based back to the entity script and then passed to the CardGained function.



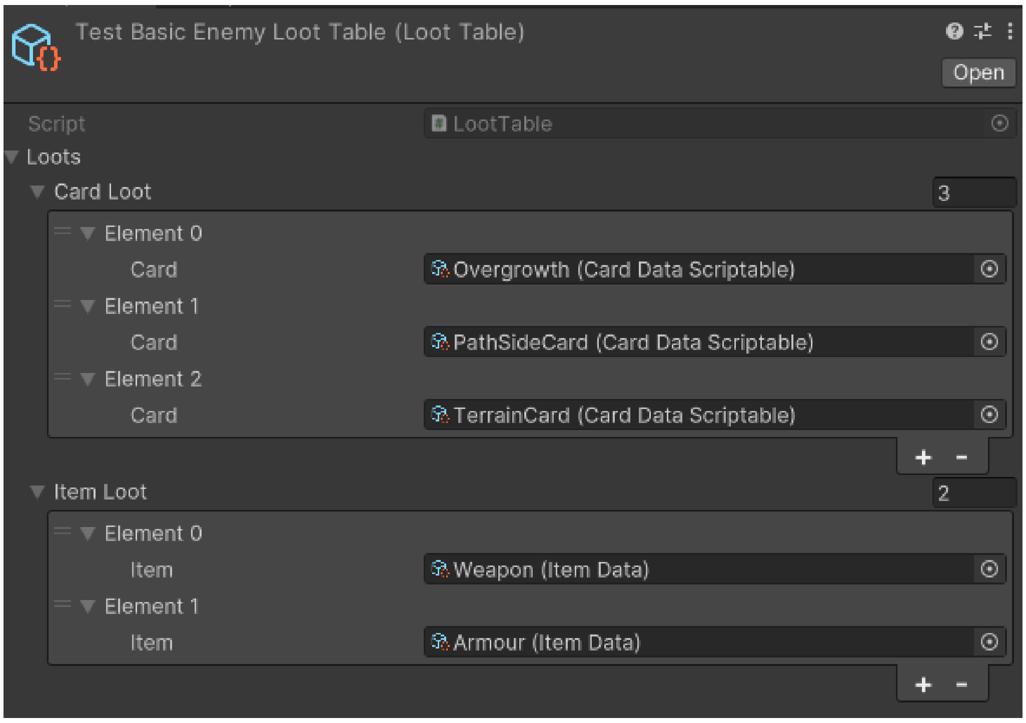
```
2 references
public CardDataScriptable GetCardDrop(List<CardLoot> cardloot)
{
    int[] tryArray = GenerateArray(cardloot.Count);

    for (int i = 0; i < tryArray.Length; i++)
    {
        if(0 < m_liveCardCollection.m_cardCollection[tryArray[i]].m_numberInDeck)
        {
            m_liveCardCollection.m_cardCollection[tryArray[i]].m_numberInDeck --;
            return m_liveCardCollection.m_cardCollection[tryArray[i]].m_card;
        }
    }

    return null;
}

1 reference
private int[] GenerateArray(int size)
{
    int[] array = new int[size];

    for (int i = 0; i < array.Length; i++)
    {
        array[i] = i;
    }

    for (int i = 0; i < array.Length; i++)
    {
        int temp = array[i];
        int random = Random.Range(i, array.Length);
        array[i] = array[random];
        array[random] = temp;
    }

    return array;
}
```

Figure 27: Screenshot of the function GetCardDrop() seen in deck script


Card Collection (Card Collection Scriptable)

Script	CardCollectionScriptable
m_card Collection	3
Element 0	
Card	Overgrowth (Card Data Scriptable)
Number In Deck	5
Element 1	
Card	PathSideCard (Card Data Scriptable)
Number In Deck	5
Element 2	
Card	TerrainCard (Card Data Scriptable)
Number In Deck	20
Total Card Count	0

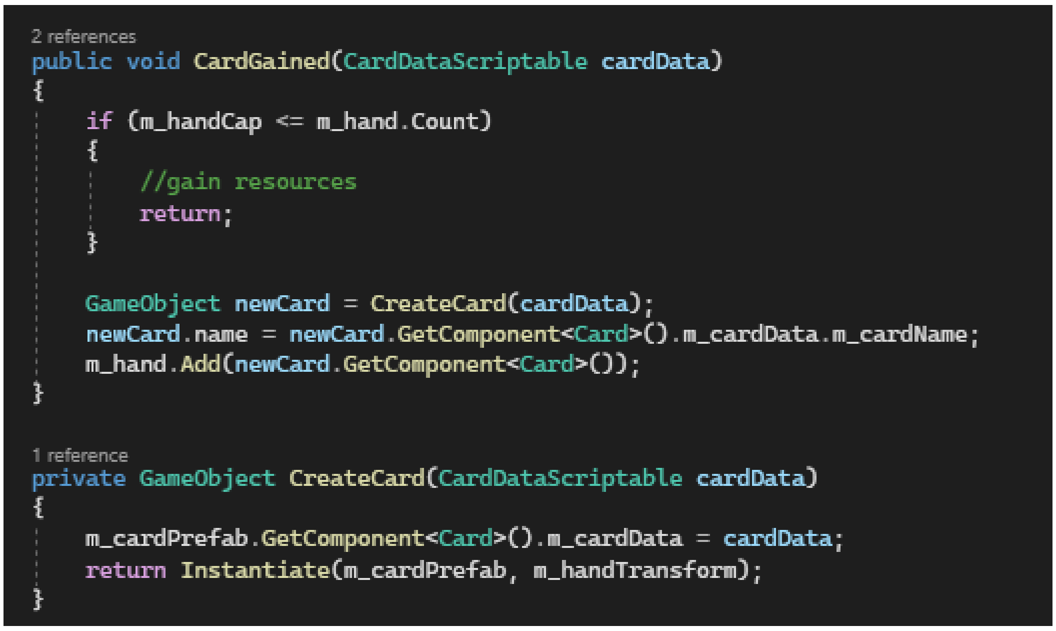
Figure 28: Scriptable object of the deck of cards being used and the count of each in the deck.


Test Basic Enemy Loot Table (Loot Table)

Script	LootTable
Loots	
Card Loot	3
Element 0	
Card	Overgrowth (Card Data Scriptable)
Element 1	
Card	PathSideCard (Card Data Scriptable)
Element 2	
Card	TerrainCard (Card Data Scriptable)
Item Loot	2
Element 0	
Item	Weapon (Item Data)
Element 1	
Item	Armour (Item Data)

Figure 29:An example of a scriptable object of a loot table linked to adventurers.

When CardGained gets called, it checks uses the card prefab to instantiate the card as a child of the hand transform in the canvas UI (user interface). This is spaced via Unity’s vertical layout group, which is aided by the card prefabs using layout elements to suit the size of the cards.

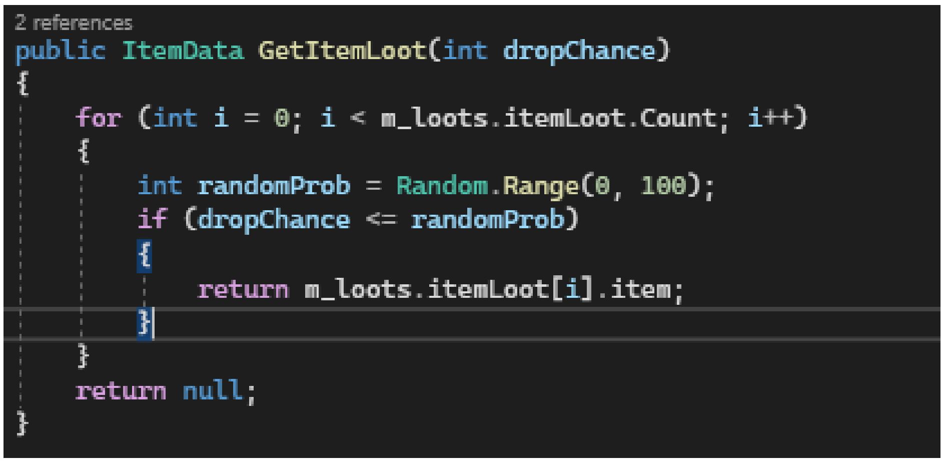


```
2 references
public void CardGained(CardDataScriptable cardData)
{
    if (m_handCap <= m_hand.Count)
    {
        //gain resources
        return;
    }

    GameObject newCard = CreateCard(cardData);
    newCard.name = newCard.GetComponent<Card>().m_cardData.m_cardName;
    m_hand.Add(newCard.GetComponent<Card>());
}

1 reference
private GameObject CreateCard(CardDataScriptable cardData)
{
    m_cardPrefab.GetComponent<Card>().m_cardData = cardData;
    return Instantiate(m_cardPrefab, m_handTransform);
}
```

Figure 30: Screenshot of the function to add a card to the players hand.

8.3.2 Inventory

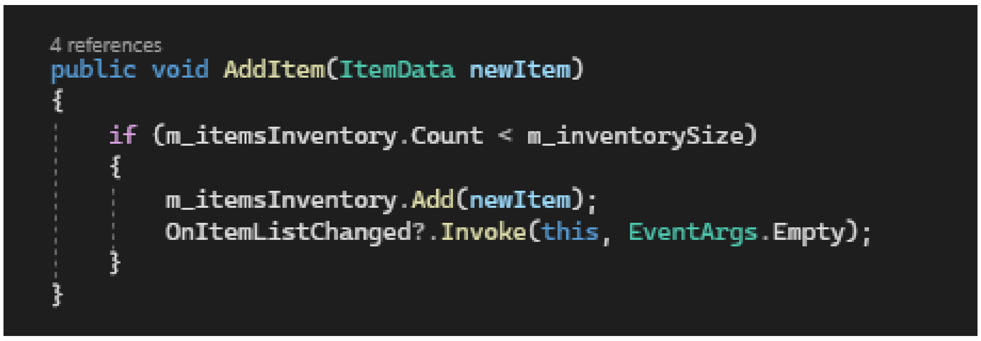
Items are handled slightly differently to cards, in Figure 26 the function GetItemLoot is called which can be seen in Figure 31. This function takes the drop chance from the adventurer's loot drop chance which is stored within the entity scriptable object which is described in 8.5 Entity Data. This drop chance is then used to check if the adventurer will gain the item, with each item rolling a random number with a range of 100. This is then compared to the adventurers drop chance, if less then the item is returned to the LootDrop function.



```
2 references
public ItemData GetItemLoot(int dropChance)
{
    for (int i = 0; i < m_loots.itemLoot.Count; i++)
    {
        int randomProb = Random.Range(0, 100);
        if (dropChance <= randomProb)
        {
            return m_loots.itemLoot[i].item;
        }
    }
    return null;
}
```

Figure 31: GetItemLoot function in LootTable script

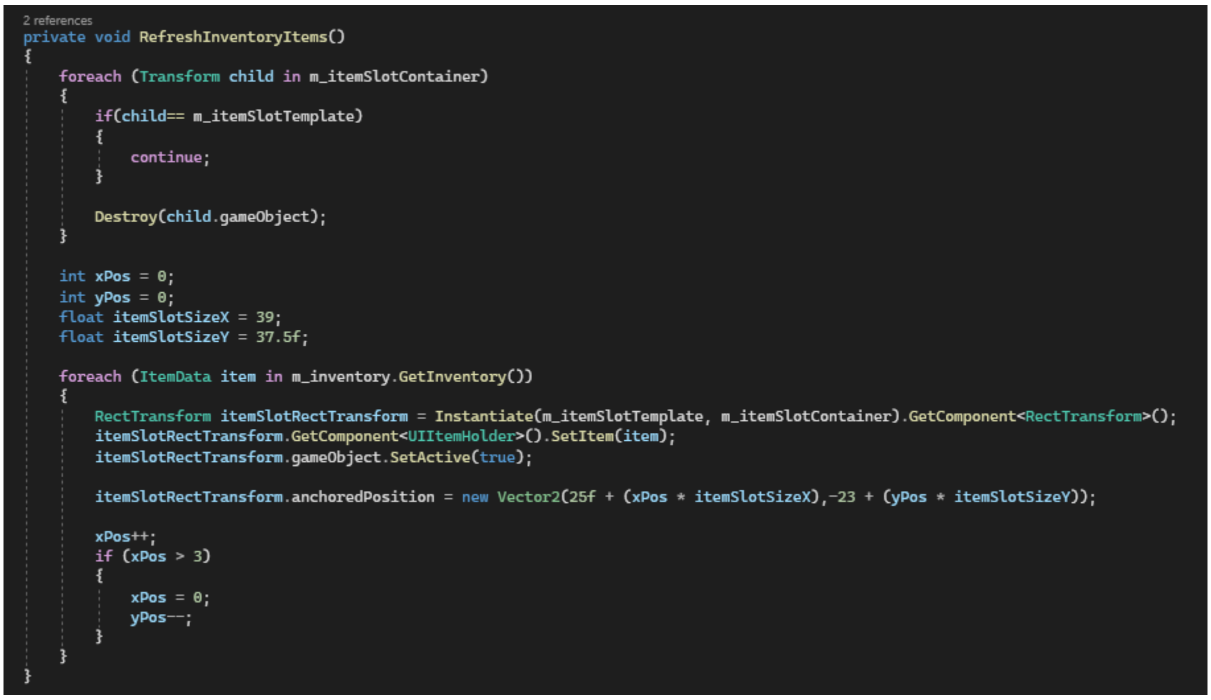
From there the item is sent to the player's inventory through an event as seen in Figure 26. In the inventory script this event triggers the AddItem function which checks to see if the player has enough inventory space. Once this is confirmed the item will be added to the list of items and another event is fired called OnItemListChanged, this updates the UI to allow the player to get visual feedback on gaining said item.



```
4 references
public void AddItem(ItemData newItem)
{
    if (m_itemsInventory.Count < m_inventorySize)
    {
        m_itemsInventory.Add(newItem);
        OnItemListChanged?.Invoke(this, EventArgs.Empty);
    }
}
```

Figure 32: AddItem function within Inventory script.

That event then tiggers a function within the UIInventory script which can be seen in Figure 33. Upon being called the function deletes all UI elements representing items in the inventory, this allows for correct formatting. Then the inventory of item is received form the inventory script in a foreach loop. This instantiates each item within the UI, while setting the data for said item on the UI element. After 4 items have been added a new row is created till the max inventory number is reached which is 12. The UI inventory can be seen in Figure 34, this also shows the displaying the item stats through a pop-up UI element. The top pop up showcases an equipped item that matches the hovered item while the bottom showcases the item that the player is hovering.



```
2 references
private void RefreshInventoryItems()
{
    foreach (Transform child in m_itemSlotContainer)
    {
        if(child== m_itemSlotTemplate)
        {
            continue;
        }

        Destroy(child.gameObject);
    }

    int xPos = 0;
    int yPos = 0;
    float itemSlotSizeX = 39;
    float itemSlotSizeY = 37.5f;

    foreach (ItemData item in m_inventory.GetInventory())
    {
        RectTransform itemSlotRectTransform = Instantiate(m_itemSlotTemplate, m_itemSlotContainer).GetComponent<RectTransform>();
        itemSlotRectTransform.GetComponent<UIItemHolder>().SetItem(item);
        itemSlotRectTransform.gameObject.SetActive(true);

        itemSlotRectTransform.anchoredPosition = new Vector2(25f + (xPos * itemSlotSizeX),-23 + (yPos * itemSlotSizeY));

        xPos++;
        if (xPos > 3)
        {
            xPos = 0;
            yPos--;
        }
    }
}
```

Figure 33: RefreshIventoryItems function within the UIInventory script.

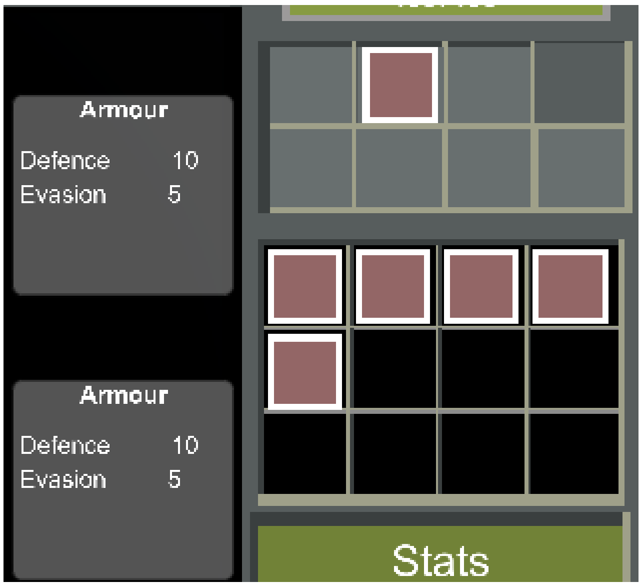


Figure 34: Example of UI inventory and UI elements showcasing item stats of equipped item and hovered inventory item.

8.4 Generation of the map

The generation of the map is an important factor in keeping players engaged long-term. Since this game is a roguelite, the player could be playing multiple runs. This has the potential to cause boredom is the map is too similar. To counteract this PCG will be used to change the map between runs. To further this engagement, the player will be able to interact with the map directly. This will allow them to shape the environment of the dungeon to their liking with the deck that they have created.

8.4.1 Generation of the dungeon

The generation of the dungeon is handled through the use of tool that was developed in previous projects (Bennett, 2024). The tool uses graph rewriting to create the layout of the dungeon, while the tool offers cellular automaton to generate an environment based off the data, it will not be used. Instead, the base tilemap will be used with a low scale so each tile is represented by a large tile to allow the player to easily interact with each tile. An example of this can be seen in Figure 20.

8.4.2 Placing cards to change tiles

Placing cards to change tiles within the dungeon is a core game mechanic so it is important that it works. As seen in Figure 35, when the player drops a dragged card the card’s parent will be checked to see if it has changed, if it has not changed from the origin the position will be changed back to its origin in the hand UI. The indicators showing valid placements for the cards then get reset. If the player enters combat while trying to place a card, then it will be reset back to the origin position to stop any errors from occurring. If not, then a coroutine will delay the placement of the card till the end of the frame. After this the card is set to placed, while the game controller verifies this. This calls the function Figure 37, this communicates with other scripts to set data of where that card was placed to change the tile data. The card then gets removed from the hand, while the spawner updates list of tiles with the enemy type from the card data which can be seen in Figure 38. After this is done the attention rating increases by one and the UI is updated. A bool is returned to the WaitForPlacement function and if the card is not placed the UI is made active again.

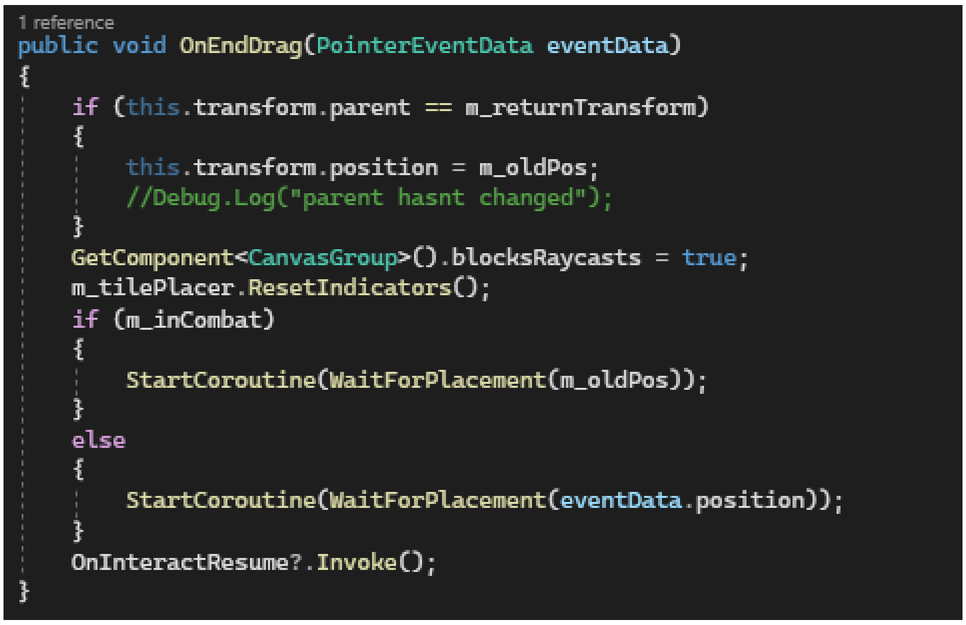
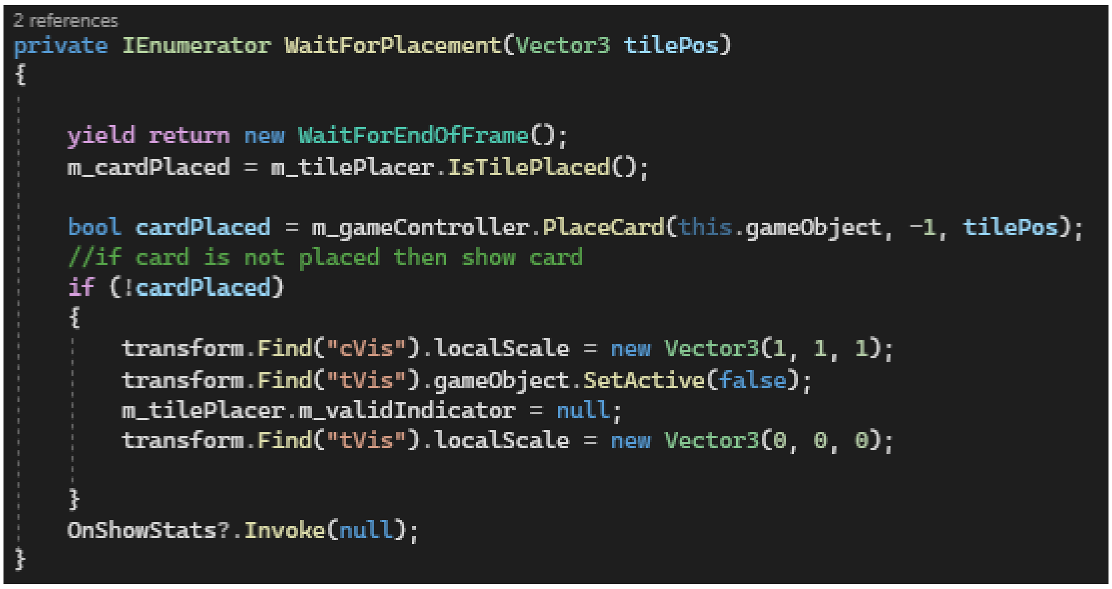
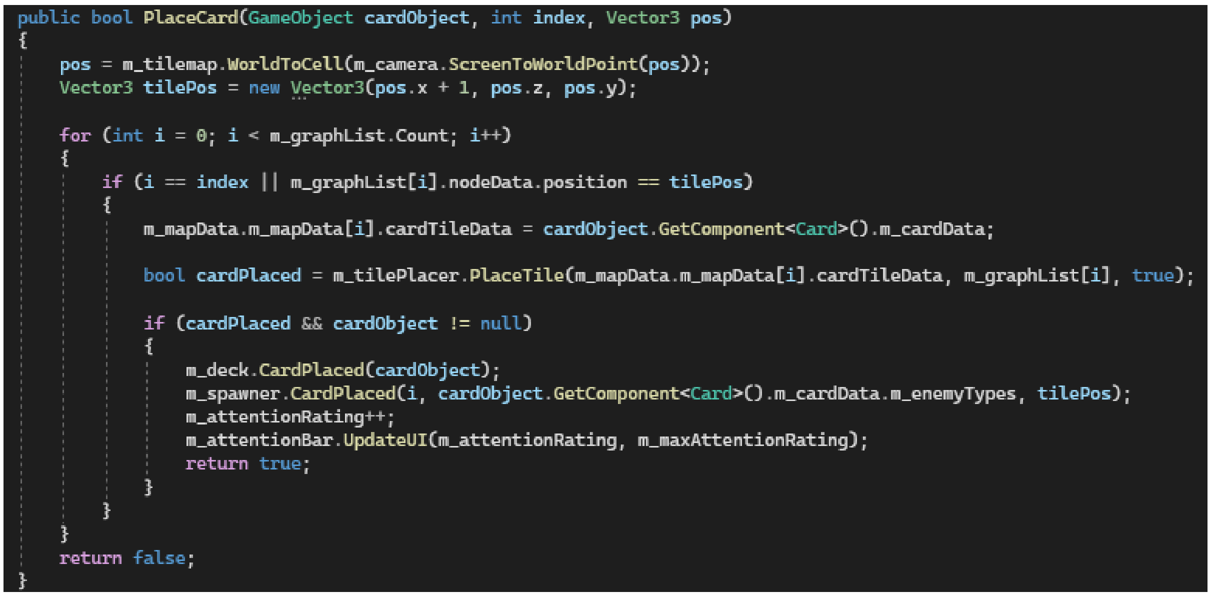

```
1 reference
public void OnEndDrag(PointerEventData eventData)
{
    if (this.transform.parent == m_returnTransform)
    {
        this.transform.position = m_oldPos;
        //Debug.Log("parent hasnt changed");
    }
    GetComponent<CanvasGroup>().blocksRaycasts = true;
    m_tilePlacer.ResetIndicators();
    if (m_inCombat)
    {
        StartCoroutine(WaitForPlacement(m_oldPos));
    }
    else
    {
        StartCoroutine(WaitForPlacement(eventData.position));
    }
    OnInteractResume?.Invoke();
}
```

Figure 35: OnEndDrag function within Draggable script.



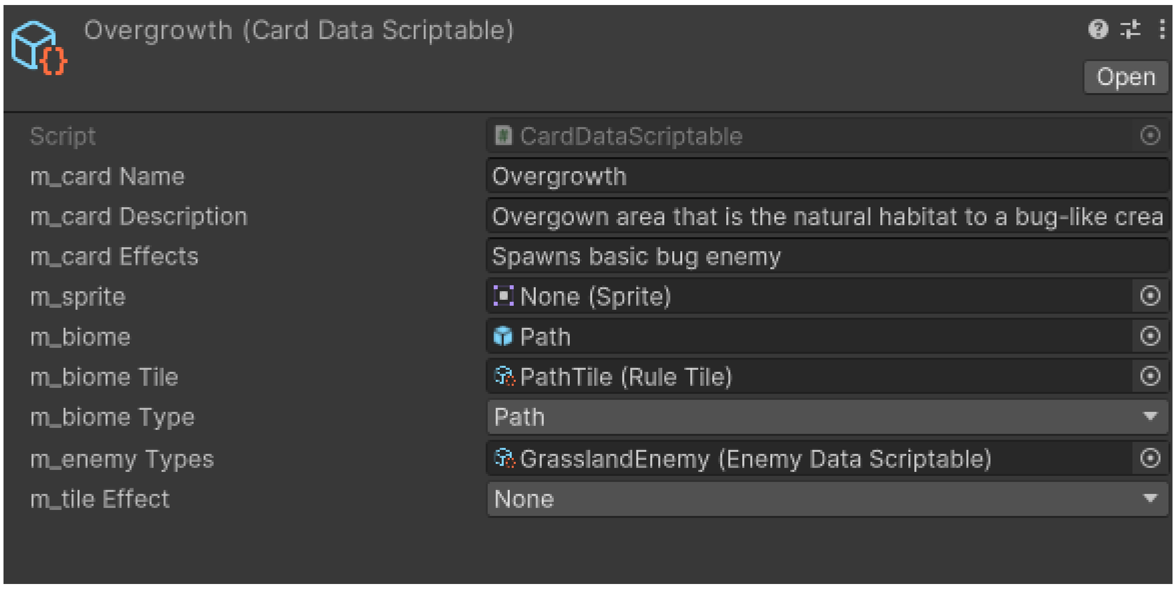
```
2 references
private IEnumerator WaitForPlacement(Vector3 tilePos)
{

    yield return new WaitForEndOfFrame();
    m_cardPlaced = m_tilePlacer.IsTilePlaced();

    bool cardPlaced = m_gameController.PlaceCard(this.gameObject, -1, tilePos);
    //if card is not placed then show card
    if (!cardPlaced)
    {
        transform.Find("cVis").localScale = new Vector3(1, 1, 1);
        transform.Find("tVis").gameObject.SetActive(false);
        m_tilePlacer.m_validIndicator = null;
        transform.Find("tVis").localScale = new Vector3(0, 0, 0);

    }
    OnShowStats?.Invoke(null);
}
```

Figure 36: WaitForPlacement coroutine within Draggable script.



```
public bool PlaceCard(GameObject cardObject, int index, Vector3 pos)
{
    pos = m_tilemap.WorldToCell(m_camera.ScreenToWorldPoint(pos));
    Vector3 tilePos = new Vector3(pos.x + 1, pos.z, pos.y);

    for (int i = 0; i < m_graphList.Count; i++)
    {
        if (i == index || m_graphList[i].nodeData.position == tilePos)
        {
            m_mapData.m_mapData[i].cardTileData = cardObject.GetComponent<Card>().m_cardData;

            bool cardPlaced = m_tilePlacer.PlaceTile(m_mapData.m_mapData[i].cardTileData, m_graphList[i], true);

            if (cardPlaced && cardObject != null)
            {
                m_deck.CardPlaced(cardObject);
                m_spawner.CardPlaced(i, cardObject.GetComponent<Card>().m_cardData.m_enemyTypes, tilePos);
                m_attentionRating++;
                m_attentionBar.UpdateUI(m_attentionRating, m_maxAttentionRating);
                return true;
            }
        }
    }
    return false;
}
```

Figure 37: PlaceCard function within GameController script.

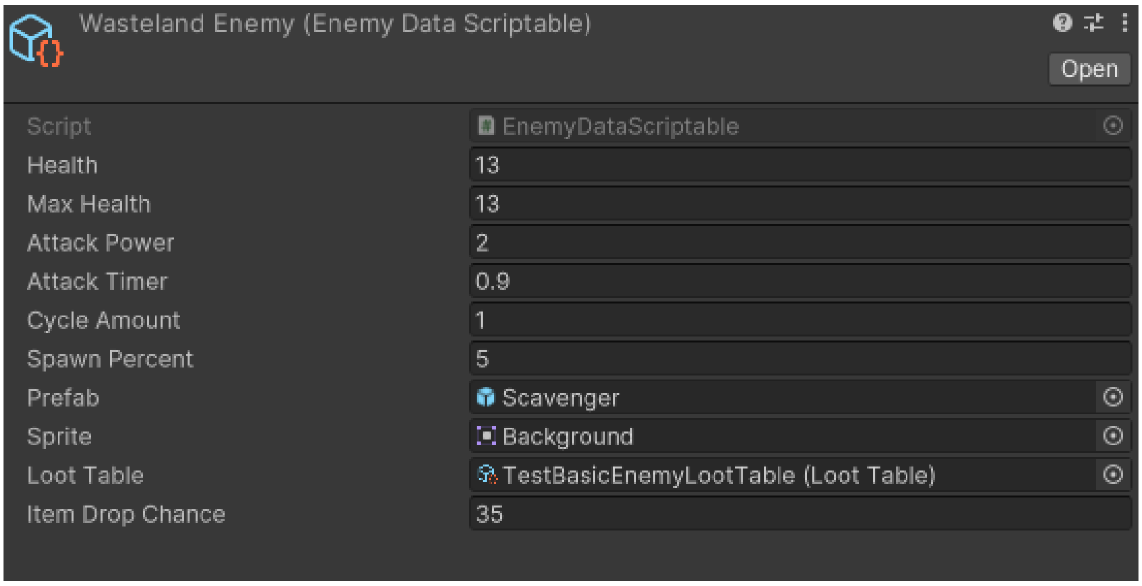


Overgrowth (Card Data Scriptable)	Open
Script	CardDataScriptable
m_card Name	Overgrowth
m_card Description	Overgown area that is the natural habitat to a bug-like crea
m_card Effects	Spawns basic bug enemy
m_sprite	None (Sprite)
m_biome	Path
m_biome Tile	PathTile (Rule Tile)
m_biome Type	Path
m_enemy Types	GrasslandEnemy (Enemy Data Scriptable)
m_tile Effect	None

Figure 38: Screenshot of an example tile/card data scriptable object

8.5 Entity Data

This game has a wide range of minions and adventurers, which means there is a lot of data stored. To manage this data in engine, scriptable objects were created to quickly generate entities. An example of this can be seen in Figure 39. This scriptable object stores the entities base stats for combat as well as extra data used across multiple scripts. This allows for quick adjustments of stats and easy management since the data is stored in one place.



Wasteland Enemy (Enemy Data Scriptable)	Open
Script	EnemyDataScriptable
Health	13
Max Health	13
Attack Power	2
Attack Timer	0.9
Cycle Amount	1
Spawn Percent	5
Prefab	Scavenger
Sprite	Background
Loot Table	TestBasicEnemyLootTable (Loot Table)
Item Drop Chance	35

Figure 39: Screenshot of a enemy data scriptable object.

8.6 Extra tile features

Within Figure 38, a key component of the game can be seen which was glossed over from the card placement information. This is tile effects and enemy types, which can be seen in Figure 40. Tile effects can be found throughout the dungeon's different tile types meanwhile enemy types are typically found on the path. Tile effects are the main cause of interesting combat, with each effect having impacts on each fight. Some of these effects can be combined together to give minions the advantage in combat. The swamp tile is an example of this as it causes all entities within the tile to take damage over time when in combat. But the minions that are attached (mosquito minions) have vampirism, which allows them to out heal this damage. If combined with a carnivorous tile, which kills any entity that reaches 20% of their health, the mosquito minions will be able to out heal this 20% marker, while adventurers are punished for longer combat in this tile. This ties into Burgun's (2014) theory of information horizons to engage the player. As the player cannot control when they receive these cards, they will have to plan the layout of their dungeon as to not make a mistake. This engagement is important, as mentioned above, when plans succeed the player feels rewarded and thus wants to continue playing.

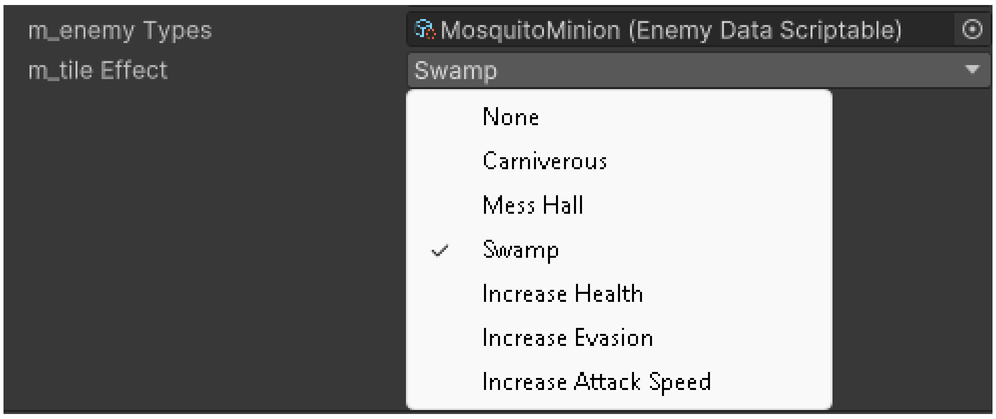
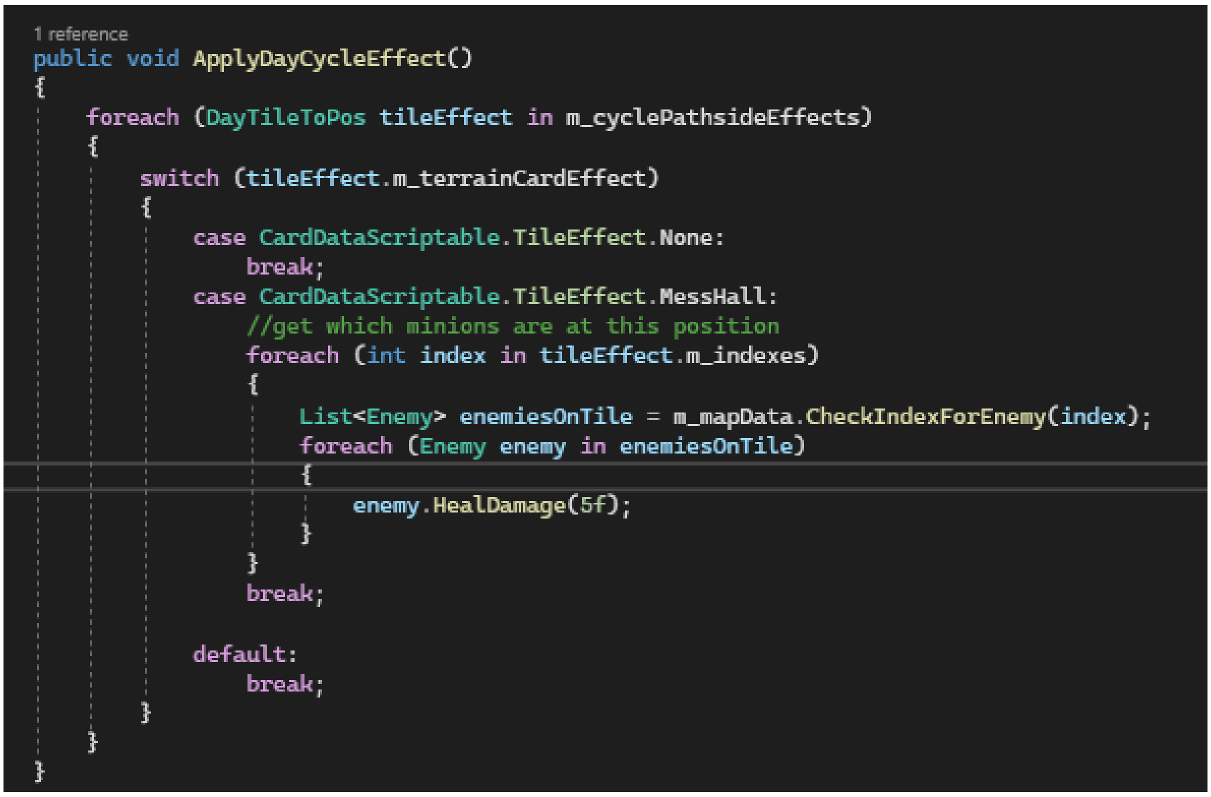

m_enemy Types: MosquitoMinion (Enemy Data Scriptable)
m_tile Effect: Swamp

Figure 40: Example of enemy types and tile effects on a CardDataScriptbable object.

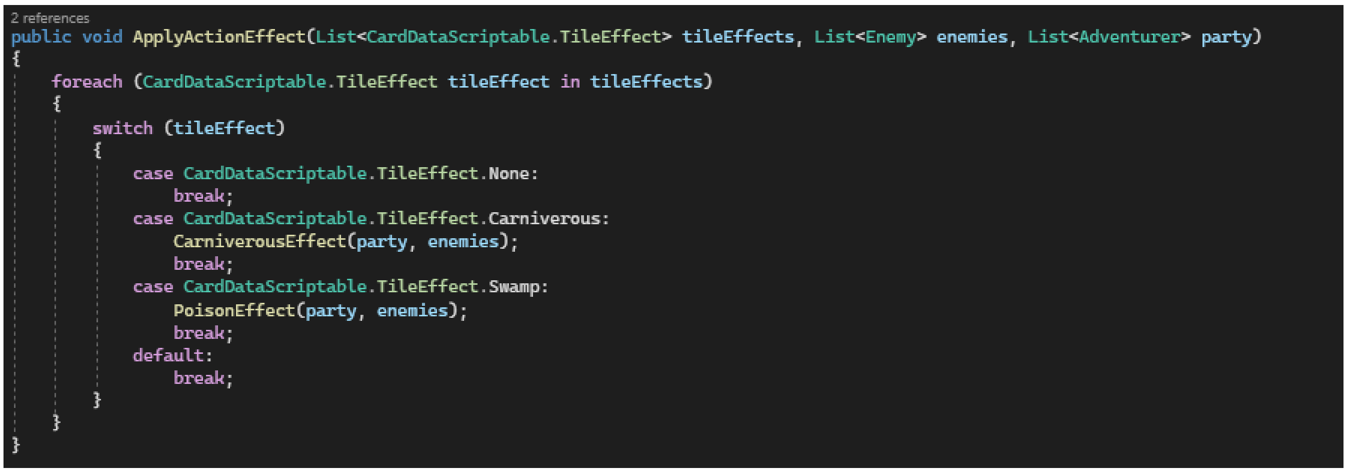
8.6.1 Tile effects

Tile effects are triggered in three different ways, a day cycle completion, in combat, or passively. This implementation can be seen in Figure 41,Figure 42, and Figure 43. The example seen in Figure 41, is triggered by the day completing and is called from the game controller script. This will then use a foreach loop to cycle all tile effects added to its representative list. As used for all implementations of tile effects, a switch case is used to check which tile effect will be implemented. This is similar to Figure 42, except this is called after every second of combat. Each tile effect function will relate to the tile placed. Passive tile effects are found in terrain tiles and is implemented slightly differently as seen in Figure 43. This function is called when the player places a new terrain tile to update all of the entities stats to represent the dungeon's landscape. There is an alternative function that applies the current tiles to new minions that are spawned, but that works in a similar fashion as seen in Figure 43 but is called by the Spawner script.



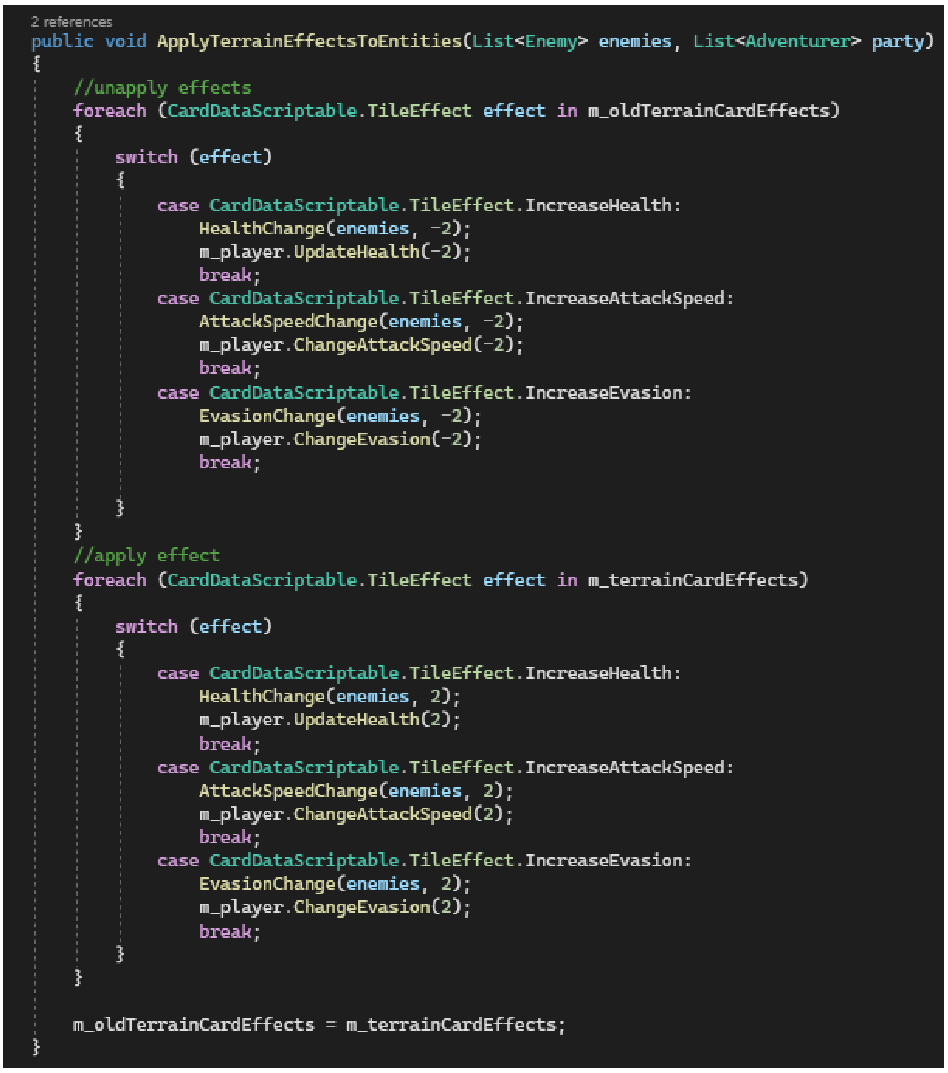
```
1 reference
public void ApplyDayCycleEffect()
{
    foreach (DayTileToPos tileEffect in m_cyclePathsideEffects)
    {
        switch (tileEffect.m_terrainCardEffect)
        {
            case CardDataScriptable.TileEffect.None:
                break;
            case CardDataScriptable.TileEffect.MessHall:
                //get which minions are at this position
                foreach (int index in tileEffect.m_indexes)
                {
                    List<Enemy> enemiesOnTile = m_mapData.CheckIndexForEnemy(index);
                    foreach (Enemy enemy in enemiesOnTile)
                    {
                        enemy.HealDamage(5f);
                    }
                }
                break;

            default:
                break;
        }
    }
}
```

Figure 41: ApplyDayCycleEffect function within TileEffect script



```
2 references
public void ApplyActionEffect(List<CardDataScriptable.TileEffect> tileEffects, List<Enemy> enemies, List<Adventurer> party)
{
    foreach (CardDataScriptable.TileEffect tileEffect in tileEffects)
    {
        switch (tileEffect)
        {
            case CardDataScriptable.TileEffect.None:
                break;
            case CardDataScriptable.TileEffect.Carniverous:
                CarniverousEffect(party, enemies);
                break;
            case CardDataScriptable.TileEffect.Swamp:
                PoisonEffect(party, enemies);
                break;
            default:
                break;
        }
    }
}
```

Figure 42: ApplyActionEffect function within TileEffects script.

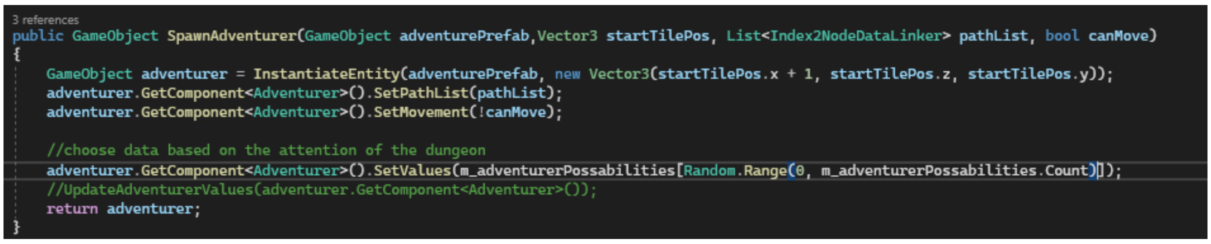


```
2 references
public void ApplyTerrainEffectsToEntities(List<Enemy> enemies, List<Adventurer> party)
{
    //unapply effects
    foreach (CardDataScriptable.TileEffect effect in m_oldTerrainCardEffects)
    {
        switch (effect)
        {
            case CardDataScriptable.TileEffect.IncreaseHealth:
                HealthChange(enemies, -2);
                m_player.UpdateHealth(-2);
                break;
            case CardDataScriptable.TileEffect.IncreaseAttackSpeed:
                AttackSpeedChange(enemies, -2);
                m_player.ChangeAttackSpeed(-2);
                break;
            case CardDataScriptable.TileEffect.IncreaseEvasion:
                EvasionChange(enemies, -2);
                m_player.ChangeEvasion(-2);
                break;

        }
    }
    //apply effect
    foreach (CardDataScriptable.TileEffect effect in m_terrainCardEffects)
    {
        switch (effect)
        {
            case CardDataScriptable.TileEffect.IncreaseHealth:
                HealthChange(enemies, 2);
                m_player.UpdateHealth(2);
                break;
            case CardDataScriptable.TileEffect.IncreaseAttackSpeed:
                AttackSpeedChange(enemies, 2);
                m_player.ChangeAttackSpeed(2);
                break;
            case CardDataScriptable.TileEffect.IncreaseEvasion:
                EvasionChange(enemies, 2);
                m_player.ChangeEvasion(2);
                break;
        }
    }

    m_oldTerrainCardEffects = m_terrainCardEffects;
}
```

Figure 43: ApplyTerrainEffectsToEntities function within TileEffects script.

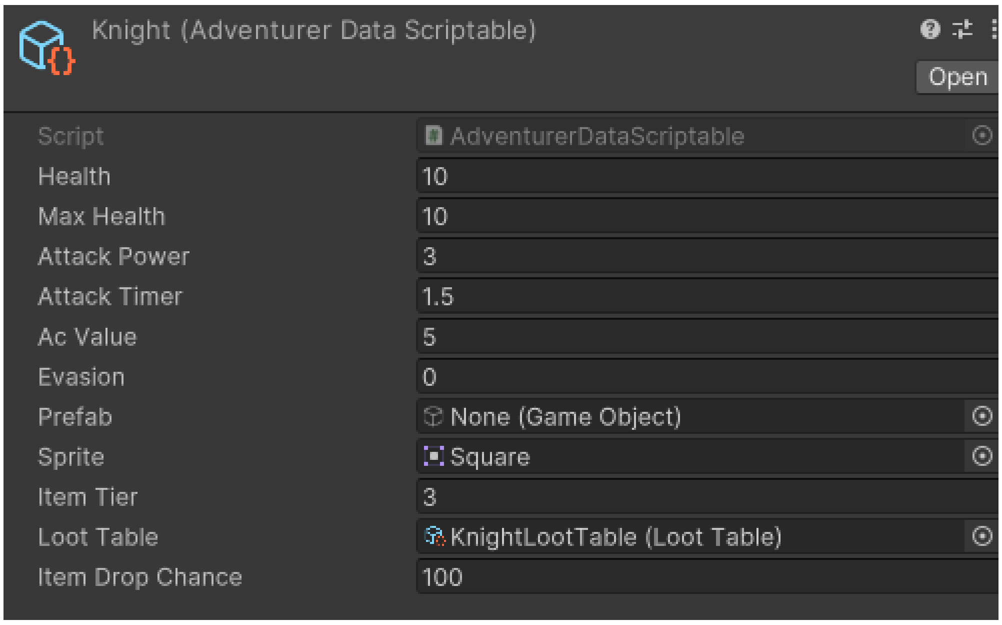
8.6.2 Entity spawning

Spawning adventurers is a key aspect for the game since it is the player's opposition, this is done through a spawner script. The game object that represents it is instantiated from the start tile location, which is found at position one on the path list from the dungeon generator. This path list is then passed to the adventurer script as well as the adventurer's stats from scriptable object that stores the different types of adventurers. An example of this can be seen in Figure 45.



```
3 references
public GameObject SpawnAdventurer(GameObject adventurePrefab,Vector3 startTilePos, List<Index2NodeDataLinker> pathList, bool canMove)
{
    GameObject adventurer = InstantiateEntity(adventurePrefab, new Vector3(startTilePos.x + 1, startTilePos.z, startTilePos.y));
    adventurer.GetComponent<Adventurer>().SetPathList(pathList);
    adventurer.GetComponent<Adventurer>().SetMovement(!canMove);

    //choose data based on the attention of the dungeon
    adventurer.GetComponent<Adventurer>().SetValues(m_adventurerPossabilities[Random.Range(0, m_adventurerPossabilities.Count)]);
    //UpdateAdventurerValues(adventurer.GetComponent<Adventurer>());
    return adventurer;
}
```

Figure 44: SpawnAdventurer function within Spawner script.


Knight (Adventurer Data Scriptable)

Script	AdventurerDataScriptable
Health	10
Max Health	10
Attack Power	3
Attack Timer	1.5
Ac Value	5
Evasion	0
Prefab	None (Game Object)
Sprite	Square
Item Tier	3
Loot Table	KnightLootTable (Loot Table)
Item Drop Chance	100

Figure 45: Screenshot of an example for adventurer data scriptable object

As seen in Figure 46, spawning minions(enemies) works in a slightly different way due to the spawn location changing. When a tile gets placed, the enemy data gets added to a list called m_enemyCollective. This allows for all minions to be accessed, which is done through a for loop, if enough day cycles have passed to the minion will be spawned with all relevant data being set.

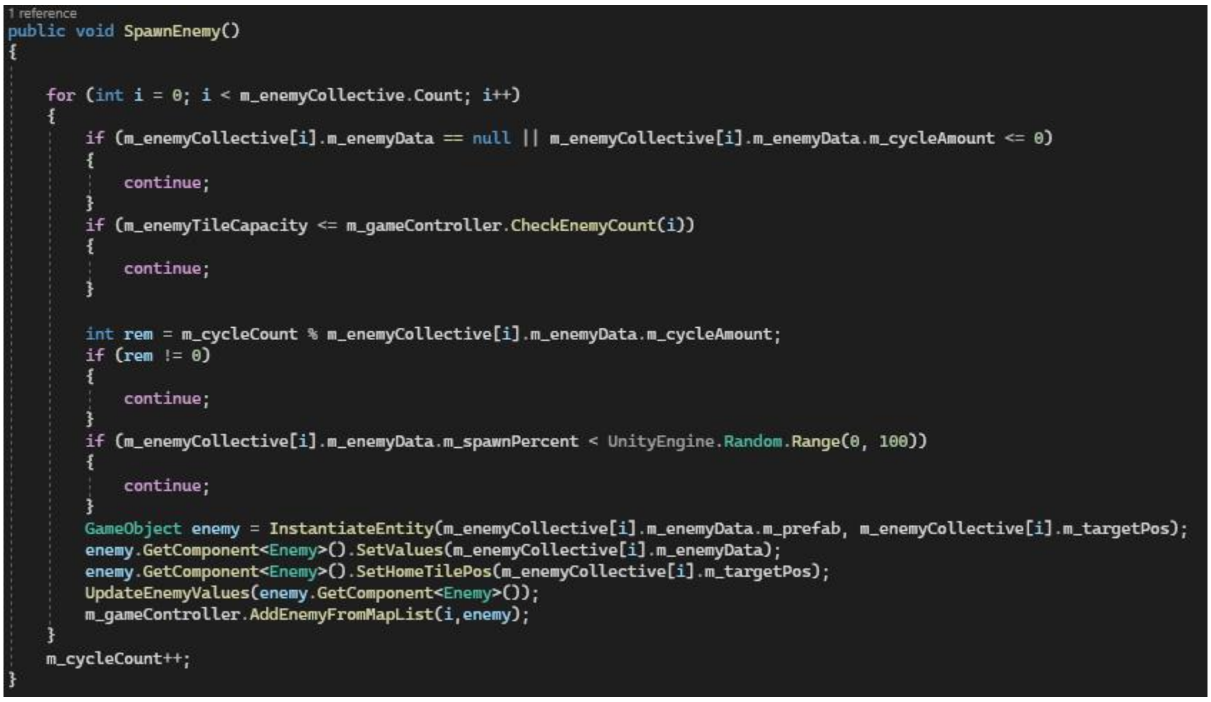

```
1 reference
public void SpawnEnemy()
{

    for (int i = 0; i < m_enemyCollective.Count; i++)
    {
        if (m_enemyCollective[i].m_enemyData == null || m_enemyCollective[i].m_enemyData.m_cycleAmount <= 0)
        {
            continue;
        }
        if (m_enemyTileCapacity <= m_gameController.CheckEnemyCount(i))
        {
            continue;
        }

        int rem = m_cycleCount % m_enemyCollective[i].m_enemyData.m_cycleAmount;
        if (rem != 0)
        {
            continue;
        }
        if (m_enemyCollective[i].m_enemyData.m_spawnPercent < UnityEngine.Random.Range(0, 100))
        {
            continue;
        }
        GameObject enemy = InstantiateEntity(m_enemyCollective[i].m_enemyData.m_prefab, m_enemyCollective[i].m_targetPos);
        enemy.GetComponent<Enemy>().SetValues(m_enemyCollective[i].m_enemyData);
        enemy.GetComponent<Enemy>().SetHomeTilePos(m_enemyCollective[i].m_targetPos);
        UpdateEnemyValues(enemy.GetComponent<Enemy>());
        m_gameController.AddEnemyFromMapList(i,enemy);
    }
    m_cycleCount++;
}
```

Figure 46: SpawnEnemy function within Spawner script

9. Critical Evaluation

This project is the conclusion to three other projects which focused on creating an environment through graph rewriting and cellular automaton, with this paper focusing on using the tool developed to create an engaging game. The theme of these projects was continued throughout each, with it being the exploration of the roguelike genre. This was chosen due to the genre's interesting mechanics and the opportunity it gives developers to explore and gain further understandings of gameplay mechanics to keep players engaged throughout multiple playthroughs. This was delved deep in this project, with it being the core focus of the game artifact that features these mechanics. Being able to get perspectives of inside the industry of mechanics that can aide in gaining user's attention and keeping it is very insightful and offers new approaches to the development process. Even more so when including the feedback on the questionnaire, as it is easy for the development process to be isolated from the targeted users. Being able to get insight into consumer's opinions and having them align with the industry's experience and research is reassuring and aided in the choice in mechanics that feature within the artifact.

One area that did not align with industry views, was PCG. Due to the previous projects heavily relied on the practice of using PCG this came to a surprise. Despite saying that, it is a good insight to gain. This is because developers often notice game mechanics frequently, but to consumers outside of that viewpoint, some mechanics may be unknown. With this idea in mind, it is important to consider how a user would interact with a mechanic if it were their first time seeing it. As even if you make a game for die hard fans, there will still be a proportion of the users who use the game as a gateway into that genre and its mechanics. Other issues came with developing the game itself, with some mechanics proving to be hard to implement into the game. An example of this is one of the requirements seen in Figure 19. This was placing cards, and it is interacting with the tilemap of the dungeon. This was a challenge due to it being built upon the tilemap system of the tool, which was simplistic. This meant it had to be changed to better suit the game. Additionally, being able to drag a UI element and place it onto a 3D gameobject was an obstacle. This was overcome by using Unity's premade tilemap function that allowed a tile to be changed via a vector3 position.

On a second approach of this project, allowing for the scope to include user testing considerations would benefit. This would allow for the research into the different methods employed by the industry and how the industry applies these changes. That being said, this might be too large of a scope to do in one project, so it could be used as a project to explore the later stages of the development process. This will allow for insight into developing a tool, creating a game with that tool, and then finishing the game and getting user feedback. This is important as most projects consider how they will start and tend to ignore how they will finish.

10. Bibliography

Bartle, R., 1996. *HEARTS, CLUBS, DIAMONDS, SPADES: PLAYERS WHO SUIT MUDS.* [Online]
Available at: https://mud.co.uk/richard/hcds.htm#Bartle,%201990a
[Accessed May 2024].

Bennett, W., 2024. *EXPANDING GENERATIVE ITERATIONS OF THE PROTOTYPE,* s.l.: s.n.

Brazie, A., 2022. *Video Game Mechanics: A Beginner's Guide (with Examples).* [Online]
Available at: https://gamedesignskills.com/game-design/video-game-mechanics/#what-are-video-game-mechanics
[Accessed June 2024].

Burgun, K., 2014. *Uncapped Look-Ahead and the Information Horizon.* [Online]
Available at: https://keithburgun.net/uncapped-look-ahead-and-the-information-horizon/
[Accessed June 2024].

Bycer, J., 2018. *Replayability in Game Design.* [Online]
Available at: https://medium.com/super-jump/replayability-in-game-design-798fbb91a726
[Accessed June 2024].

Bycer, J., 2021. *Designing Difficulty Systems in Videogames.* [Online]
Available at: https://game-wisdom.com/critical/debating-difficulty-game-design
[Accessed June 2024].

ConcernedApe, 2016. *Stardew Valley.* [Online]
Available at: https://store.steampowered.com/app/413150/Stardew_Valley/
[Accessed June 2024].

Csikszentmihalyi, M., 1990. *Flow: The Psychology of Optimal Experience.* s.l.:Harper & Row.

Dhule, M., 2022. *Exploring Game Mechanics: Principles and Techniques to Make Fun, Engaging Games.* s.l.:Berkeley, CA: Apress L. P.

Dodge Roll, 2016. *Enter the Gungeon.* [Online]
Available at: https://store.steampowered.com/app/311690/Enter_the_Gungeon/
[Accessed March 2023].

Eaves, T., 2021. *Game Design: Getting Difficulty Right.* [Online]
Available at: https://www.linkedin.com/pulse/game-design-getting-difficulty-right-thomas-eaves/
[Accessed June 2024].

Eggplant: The Secret Lives of Games, 2019. *12: Into the Breach with Justin Ma.* [Online]
Available at: https://thespelunkyshowlike.libsyn.com/12-into-the-breach-with-justin-ma
[Accessed June 2024].

Eng, D., 2023. *What is Player Engagement?.* [Online]
Available at: https://www.universityxp.com/blog/2023/10/17/what-is-player-engagement
[Accessed June 2024].

Epynx, Inc., 1985. *Rogue.* [Online]
Available at: https://store.steampowered.com/app/1443430/Rogue/
[Accessed Novemeber 2023].

Firaxis Games, 2024. *XCOM.* [Online]
Available at: https://store.steampowered.com/franchise/xcom
[Accessed June 2024].

French, J., 2024. *How to write a game design document..* [Online]
Available at: https://gamedevbeginner.com/how-to-write-a-game-design-document-with-examples/
[Accessed July 2024].

From Software, 2019. *Sekiro Shadows Die Twice.* [Online]
Available at: https://www.sekirothegame.com/uk/en/
[Accessed June 2024].

Game Maker's Toolkit, 2018. *How to Keep Players Engaged (Without Being Evil).* [Online]
Available at: https://www.youtube.com/watch?v=hbzGO_Qonu0
[Accessed June 2024].

Game Maker's Toolkit, 2019. *Roguelikes, Persistency, and Progression.* [Online]
Available at: https://www.youtube.com/watch?v=G9FB5R4wVno
[Accessed June 2024].

Game Maker's Toolkit, 2020. *The Two Types of Random in Game Design.* [Online]
Available at: https://www.youtube.com/watch?v=dwI5b-wRLic&t=929s
[Accessed June 2024].

GeeksForGeeks, 2024. *What is Agile Methodology?.* [Online]
Available at: https://www.geeksforgeeks.org/what-is-agile-methodology/
[Accessed July 2024].

Hoeppner, E., 2017. *Plan Distruption.* [Online]
Available at: https://web.archive.org/web/20200809233814/http://www.ellahoeppner.com/
[Accessed June 2024].

Icodewithben, 2023. *Mihaly Csikszentmihalyi's Flow theory — Game Design ideas.* [Online]
Available at: https://medium.com/@icodewithben/mihaly-csikszentmihalyis-flow-theory-game-design-ideas-9a06306b0fb8
[Accessed June 2024].

id Software, 2016. *Doom.* [Online]
Available at: https://store.steampowered.com/app/379720/DOOM/
[Accessed June 2024].

IRDC, 2008. *The International Roguelike Development Conference.* Berlin: s.n.

Johnson, S., 2014. *A Study in Transparency: How Board Games Matter.* s.l., GDC.

Jong, P., 2024. *A Beginner's Guide to Roguelike Games.* [Online]
Available at: https://gamespeakmag.com/reviews/what-is-a-roguelike/
[Accessed June 2024].

Kyatric, 2013. *Bartle's Taxonomy of Player Types (And Why It Doesn't Apply to Everything).* [Online]
Available at: https://code.tutsplus.com/bartles-taxonomy-of-player-types-and-why-it-doesnt-apply-to-everything--gamedev-4173a
[Accessed May 2024].

Larsen, J. M., 2010. *Difficulty Curves.* [Online]
Available at: https://www.gamedeveloper.com/design/difficulty-curves
[Accessed June 2024].

Larsen, J. M., 2010. *Difficulty Curves.* [Online]
Available at: https://www.gamedeveloper.com/design/difficulty-curves
[Accessed March 2024].

Ludology, 2018. *Input Output Randomness.* [Online]
Available at: https://podcasts.apple.com/gb/podcast/ludology/id419046224
[Accessed June 2024].

Ludomotion, 2017. *Unexplored.* [Online]
Available at: https://store.steampowered.com/app/506870/Unexplored/
[Accessed December 2022].

Mega Crit, 2019. *Slay the Spire.* [Online]
Available at: https://store.steampowered.com/app/646570/Slay_the_Spire/
[Accessed June 2024].

Mossmouth, LLC, 2008. *Spelunky.* [Online]
Available at: https://spelunkyworld.com/
[Accessed June 2024].

Motion Twin, 2018. *Dead Cells.* [Online]
Available at: https://store.steampowered.com/app/588650/Dead_Cells/
[Accessed June 2024].

Neves, P., 2018. *Looking at Player Motivation Models.* [Online]
Available at: https://medium.com/@pauladneves/looking-at-player-motivation-models-a80f18cd90f9
[Accessed May 2024].

Red Hook Studios, 2016. *Darkest Dungeon.* [Online]
Available at: https://www.darkestdungeon.com/darkest-dungeon/
[Accessed June 2024].

Rehkopf, M., n.d.. *What is a kanban board.* [Online]
Available at: https://www.atlassian.com/agile/kanban/boards
[Accessed December 2023].

Ryan, R. M., Rigby, C. S. & Pryzybylski, A. K., 2006. *The Motivational Pull of Video Games: A Self-Determination,* s.l.: Springer Science.

Schoenau-Fog, H., 2011. *The Player Engagement Process – An Exploration of Continuation ,* s.l.: Digra congerence.

Stegner, B., 2021. *What Are Roguelike and Roguelite Video Games?.* [Online]
Available at: https://www.makeuseof.com/what-are-roguelike-and-roguelite-video-games/
[Accessed June 2024].

Strachan, D., N.A.. *The idea of a difficulty curve is all wrong.* [Online]
Available at: http://www.davetech.co.uk/difficultycurves
[Accessed June 2024].

Subset Games, 2012. *FTL: Faster then Light.* [Online]
Available at: https://subsetgames.com/ftl.html
[Accessed June 2024].

Subset Games, 2019. *Into the Breach.* [Online]
Available at: https://subsetgames.com/itb.html
[Accessed June 2024].

Subset Games, 2024. *Subset Games.* [Online]
Available at: https://subsetgames.com/
[Accessed June 2024].

Supergiant Games, 2018. *Hades.* [Online]
Available at: https://store.steampowered.com/app/1145360/Hades/
[Accessed June 2024].

Wrike, N.D. *What Is Agile Methodology in Project Management.* [Online]
Available at: https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/
[Accessed December 2023].

11.Appendix

11.1 Appendix A - Game Design Document

[DATE]

WILL BENNETT
[COMPANY NAME]
[Company address]

Figure 47: Games design document for game