# 4 Investigation of Project Idea

## 4.1 Literature Review

### 4.1.1 Procedural Generation

Procedural Generation, in simple terms, is the creation of data by computers. This data can be theoretically anything, but where procedural generation is most used is in the creation of content for video games and various media. Procedural generation is most commonly used for the creation of content and assets which possess random elements or would be tedious to create manually. In the modern-day, procedural generation is mostly associated with the generation of maps and terrain in games such as “Minecraft” and “Terraria”. It is also heavily associated with the “roguelike” genre of games, having become a core aspect of the genre over the past four decades, with some of the first games of the genre making use of procedural generation for level generation, the most notable of which being “Rogue”, the early 1980s dungeon crawler known for inspiring the genre.

There are many benefits to generating the levels/stages of a game procedurally, with some of the most prominent being added replay value, not needing to spend development time on level design, and reduced memory usage. The arguably greatest benefit of the above-listed is the added replay value. Having a game's stage be randomly generated means the player is all but guaranteed to never see the same stage twice. This does wonders for keeping games fresh and, when combined with an equally enthralling gameplay loop, can lead to creating a near-endless experience for the player.

On the opposite side of this spectrum, however, lie the downsides of procedural generation, with the most notable of them being increased randomness, difficulty adding scripted events, and it being more taxing on hardware. Although adding randomness to a game can increase its replay value, it can also act as a great detriment towards the balance of the game. The generation could cause scenarios where the player is unable to progress or even miss important events. A game that demonstrates the rights and wrongs of procedural generation would be “Risk of Rain”.

“Risk of Rain” is a roguelike where the stages the player must progress are procedurally generated, with treasure and the exit being randomly placed across the stage. The game is praised for its addictive gameplay and difficulty, but it has also seen its fair share of detraction due to how the generation of its treasure and, most importantly, stage exits can at times be inaccessible. This makes the game somewhat of a balancing nightmare due to its heavy utilization of procedural generation, sometimes causing the player to receive an overwhelming number of items by the end of the first stage or to get close to none on subsequent playthroughs. This proves that too much variance can be a detriment to games that wish to incorporate procedural generation. The developers of “Risk of Rain” did correct this issue in the game's sequel “Risk of Rain 2” by reducing what areas of stages could be procedurally generated, guaranteeing a minimum number of treasure chests that can spawn on each stage, and by limiting the positions the stage exit could spawn on to only several positions on the stage that the player is assured to reach.

This style of procedural generation with limits works wonders by still allowing the game to be varied while not causing situations in which the game is unbeatable. HullBreaker will use this limited procedural generation to create the navigation system for the stages of the game, as it will help to create the addictive gameplay loop desired for the game and assist in retaining control of game balance.

### 4.1.2 Dynamic Difficulty

Dynamic Difficulty is the process of adjusting the various behaviours and stats in a game based on the performance of the player. These adjustments can be as simple as increasing enemy health and speed to the addition of completely new enemies and scenarios depending on how well the player is performing. The player's performance can be judged off a number of different statistics, and many games have taken various different approaches to this. Some examples of statistics that can be indicative of how challenged the player can be by the game include:

- How much health the player has lost.
- The player's average damage dealt to enemies.
- The time it takes the player to overcome an encounter.
- The number of upgrades or power-ups a player possesses.
- Etc.

One or more of these above statistics can be allocated a score, and then various enemy or world properties can be increased, or even potentially decreased, proportionally to that score. This is an excellent way of adding additional challenge to a game and can even potentially work to let new players ease into the game if enemies become slightly weaker if they do not perform well.

This way of measuring difficulty, however, can cause issues if not enough data is taken in to determine how well a player is doing. An example of this would be the dynamic difficulty in the racing game “Mario Kart”. The further a player is from first place in “Mario Kart”, the lower quality of items they will receive, with first place only being able to obtain two types of items. This form of single-faceted difficulty scaling encourages a style of play where the player may choose to intentionally slow down to increase the quality of the items they receive. This can be seen in a positive and negative light as it is a form of strategic play, but it is also abusing the game's dynamic difficulty to give the player an advantage they otherwise would not have.

There is also the issue of the player experiencing a “high moment”, that being if the player gets lucky and, as an example, hits a critical attack on an enemy dealing high damage, this will then tell the dynamic difficulty system that the difficulty must be increased to account for the increase in damage. This effectively punishes the player for getting lucky and this is not desirable.

HullBreaker will improve on this system by using a lot of different and varied player data when determining the score that will dictate the increase in difficulty. It will also have a minimum difficulty decrease value, a certain range that once the difficulty increases, it sets a brand-new minimum at which the difficulty can then decrease. This means that difficulty increases are partly permanent but have a bit of give in case the player experiences a “high moment”.

## 4.1.3 Card-Based Combat

In the past decade, games, and especially roguelikes, have been utilizing new and different combat systems. Some very notable ones being “The Binding of Isaac's” top-down shooter style of combat and the game “Peglin” with its combat system being very reminiscent of the puzzle game known as “Peggle”. The most popular and interesting form of the combat system in these new age roguelikes would have to be the card game-based system.

The most popular example of which, and the main inspiration for HullBreaker, being “Slay the Spire”. In this game, the player is tasked with progressing through a tower to ultimately defeat the boss that lies at the top. The combat in “Slay the Spire” consists of the player entering a fight and drawing cards from their deck. These cards have a multitude of different effects varying from dealing damage, healing, gaining shield, adding temporary buffs, debuffing enemies, etc. “Slay the Spire” was the first game of its genre to create a combat system like this and is now regarded as a founder of the “Card-Based Roguelike” genre.

The combat system itself has many nuances and quirks. It creates many interesting decisions in gameplay and presents many interesting choices. A player can add many strong cards to their deck and hope to see them during combat, or they may choose to reduce their deck size to make seeing certain cards more consistent. It gives great replay incentive and encourages players to keep playing to see if they can create a powerful card combination or find interesting new cards.

HullBreaker will be making its combat system also a card-based one but with a twist when it comes to how the player will build their deck. From the start of the game, the player will select a “Commander” who will start with a certain ship. These ships are made of modular parts that, when editing the ship, can be detached or re-attached. Depending on what ship parts are currently attached, cards will be added to the player's combat deck, and these are the cards with which the player will fight enemies. When the player defeats an enemy, they will be prompted to salvage a part of the enemy's ship to attach to their own, adding more cards to their deck. This will be a fresh new spin on the recently popular genre and will also present its own nuances in design and execution.