FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Name Surname

Thesis title

Name of the department

Supervisor of the bachelor thesis: Supername Supersurname
Study programme: study programme

Prague YEAR

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication.

Title: Thesis title

Author: Name Surname

Department: Name of the department

Supervisor: Supername Supersurname, department

Abstract: Use the most precise, shortest sentences that state what problem the thesis addresses, how it is approached, pinpoint the exact result achieved, and describe the applications and significance of the results. Highlight anything novel that was discovered or improved by the thesis. Maximum length is 200 words, but try to fit into 120. Abstracts are often used for deciding if a reviewer will be suitable for the thesis; a well-written abstract thus increases the probability of getting a reviewer who will like the thesis.

Keywords: keyword, key phrase

Název práce: Název práce česky

Autor: Name Surname

Katedra: Název katedry česky

Vedoucí bakalářské práce: Supername Supersurname, department

Abstrakt: Abstrakt práce přeložte také do češtiny.

Klíčová slova: klíčová slova, klíčové fráze

Contents

1 Introduction **8**
- 1.1 Game Genres . . . 8
  - 1.1.1 Strategy . . . 8
  - 1.1.2 Tower Defense . . . 8
  - 1.1.3 Rogue-like . . . 9
  - 1.1.4 Combining the Two . . . 10
- 1.2 Original Vision . . . 10
- 1.3 Current Scope and Goals . . . 11

2 Game Design **12**
- 2.1 Design goals . . . 12
  - 2.1.1 Slay the Spire . . . 12
  - 2.1.2 Plants vs. Zombies . . . 12
  - 2.1.3 Strategic Depth in Each Battle . . . 12
  - 2.1.4 Strategic Depth in Each Run . . . 13
  - 2.1.5 Make Various Strategies Viable . . . 13
  - 2.1.6 Force Exploration . . . 13
  - 2.1.7 Provide a Challenge . . . 14
- 2.2 Battle . . . 14
- 2.3 Attacker Movement . . . 14
- 2.4 World . . . 16
- 2.5 Resources . . . 16
  - 2.5.1 Materials . . . 16
  - 2.5.2 Energy . . . 16
  - 2.5.3 Hull . . . 16
- 2.6 Graphical User Interface . . . 16
  - 2.6.1 Waves Left and Fuel . . . 17
  - 2.6.2 Hull . . . 17
  - 2.6.3 Wave Preview . . . 17
  - 2.6.4 Materials and Energy . . . 17
  - 2.6.5 Blueprints . . . 17
  - 2.6.6 Info Panel and Selection . . . 17
  - 2.6.7 Highlights and Range Visualization . . . 17
- 2.7 Attackers . . . 17
- 2.8 Buildings . . . 18
  - 2.8.1 Economic buildings . . . 18
  - 2.8.2 Towers . . . 18
- 2.9 Abilities . . . 18
- 2.10 Camera controls . . . 18
- 2.11 Future Features . . . 19

3 Analysis **21**
- 3.1 Game Engine . . . 21
- 3.2 Procedural Level Generation . . . 21
- 3.3 Random Number Generation . . . 21

3.4 Path Generation . . . 21
3.4.1 Initial Paths . . . 21
3.4.2 Final Paths . . . 21
3.4.3 Path Visualization . . . 22
3.5 Terrain Generation . . . 22
3.6 Resources and Obstacles . . . 23
3.7 Terrain Types . . . 24
3.8 World Builder . . . 24
3.9 Attacker Wave Generation . . . 24
3.10 Simulation . . . 24
3.11 Visuals and Interpolation . . . 24
3.12 Attacker Targeting . . . 25
3.13 Range Visualization . . . 25
3.14 Game Commnds . . . 25
3.15 Blueprints . . . 26
3.15.1 Attacker Stats . . . 26
3.15.2 Dynamic Descriptions . . . 26

4 Developer Documentation 27
4.1 Unity . . . 27
4.2 Scenes . . . 27
4.2.1 Battle . . . 27
4.2.2 Loading . . . 27
4.2.3 Main Menu . . . 27
4.3 ??? . . . 27

5 User Documentation — Designer 28
5.1 Terrain Types . . . 28
5.2 Blueprints . . . 28
5.2.1 Buildings . . . 28
5.2.2 Towers . . . 28
5.2.3 Abilities . . . 28
5.3 Attackers . . . 28
5.4 ??? . . . 28

6 User Documentation — Player 29
6.1 ?Introduction . . . 29
6.2 ?Controls . . . 29
6.3 ?Mechanics . . . 29

7 ?Playtesting 30

Conclusion 31

List of Figures 32

List of Tables 33

List of Abbreviations 34

A Attachments **35**
A.1 First Attachment 35

1 Introduction

Video games are a popular form of entertainment. There are a plethora of games to choose from, each offering a different experience. Still, it is always possible to create something new that players might enjoy. The author of this thesis enjoys both *tower defense* games and *rogue-like* games. There are not many games that combine these two genres. In this thesis, we will design and implement a video game, that uniquely blends these two genres, and discuss the decisions behind it.

1.1 Game Genres

One of the ways to categorize games is by their genre. Each game is unique, but labeling it with genres highlights how it's similar to other games. A genre can encompass many characteristics of a game, most often its mechanics, but also its theme, art style, and more. Genres have no exact definitions or strict boundaries and similarly, any individual game is usually a mix of different genres.

1.1.1 Strategy

One major genre is *strategy* games. **Strategy games** focus on tactics and long-term planning. They require a lot of thinking. There are various kinds of *strategy* games, but most often, players compete against each other to reach some goal. These players can be real humans or artificial intelligence agents. Strategy games often utilize hidden information or rely on the unpredictability of other players' actions to create an environment where there is no single best way to reach the goal. This means that players have to have a good understanding of the game and be able to adapt to the situation at hand.

There are many qualities players enjoy in *strategy* games. Of course, it feels great to outsmart your opponent or conquer a challenge. But *strategy* games also provide a sense of progression and accomplishment because of their depth. They are often very replayable because of the different situations that can arise from the game's mechanics.

One way to categorize a *strategy* game is whether is it *real-time* or *turn-based*. In a **turn-based strategy**, players take turns to make their moves. This allows for a slower pace and more time to think about the best move. In **real-time strategy** games, however, the environment evolves continuously and players have to react and make decisions quickly.

1.1.2 Tower Defense

Tower defense is a subgenre of *real-time strategy* where the player has to defend against waves of enemies. The player has to build towers, which attack the enemies as they approach, to defend their base. The attackers are very predictable and usually follow a set path. The player has to build their towers in a way that maximizes their effectiveness against the attackers. (do I want images and examples?)

The game usually consists of multiple levels, each presenting a different challenge. The player has to adapt to various attackers and level layouts. Sometimes the levels make up a campaign, where the player has to progress through harder and harder levels to reach the end. Other times, the levels are standalone and the player can choose which level to play, where the goal can be to survive as long as possible. This can take a few hours and may become very repetitive.

The attackers can have unique abilities or be resistant to certain types of towers. The composition of each wave can be predetermined or randomized to a varying degree. In this way, the game can force the player to adapt and use different strategies. To make their decisions interesting, the player has various towers and upgrades at their disposal, each with different abilities, often complimenting each other.

Players can get resources to build their defense passively, but some *tower defense* games also include an economy system. Here, the player has to build economic buildings to generate the resources. This adds another layer of strategy to the game, as the player has to balance their economy with their defense.

1.1.3 Rogue-like

Rogue-like is a subgenre of *role-playing games*. In **role-playing games**, the player takes on the role of a character and goes on an adventure. The character can grow stronger by acquiring new abilities, items, or experiences. The player has to make decisions about how to upgrade their characters to overcome the challenges they might face. This high level of customization and the sense of progression makes *role-playing games* very engaging.

Rogue-like is named after the game *Rogue*, released in 1980(how do I cite a game?). The game was known for its high difficulty and the fact that the player had to start from the beginning if they died. The *rogue-like* game is usually procedurally generated, meaning that the levels are created by an algorithm and are not set in stone. This is important because the player can't memorize the levels and has to rely on their skill and knowledge of the game's mechanics to progress. Since all in-game progress is lost when the player dies, the player has to learn from their mistakes and improve their skill to advance further. These are defining characteristics of *rogue-likes* that separate them from other *role-playing games*.

In most *rogue-like* games, you explore a dungeon, where you have to fight enemies and avoid traps. You are then rewarded with items and other resources to improve your character. The game is usually turn-based, making it more strategic.

Many games break the traditional *rogue-like* formula but still share many of its characteristics. They might have a different theme, where you don't even fight monsters but instead compete in another task. They might include a *meta-progression* system, where you unlock new items or upgrades for your future runs. Real-time gameplay is also common, where more skills are tested than just the player's ability to think strategically. These games are often called *rogue-lites*, however, we will not make such a distinction since all genres can be bent and blended in many ways. (do I include more examples here?)

1.1.4 Combining the Two

Tower defense games and *rogue-like* games have some similarities. They both create complex systems for the player to master. This makes them both very strategic, and they appeal to more hardcore players. However, these genres also differ in some ways. We think that taking some mechanics from each will create a game that is fun to play for the right audience.

Of course, there is more than one way to combine these two genres, so we have to decide which mechanics might work well together. The main characteristics of a *rogue-like* define more the overall structure of the game, rather than the core gameplay loop. That core gameplay loop will be *tower defense*. *Tower defense* can get a bit stale if you find a strategy that can be used every time. This is how some *rogue-like* elements can help. As they progress, the player will be rewarded with new towers and abilities, selected randomly. This means that the player can't rely on the same equipment every time and will likely use a different strategy in each run.

A run could consist of one very long level and the player would receive rewards throughout it. We think this would discourage the player from using the newly acquired items because their strategy already counts on their old equipment. This is why every run should be composed of many short levels. This way, they get to start each level with a new strategy based on the arsenal at their disposal. Additionally, each level will be procedurally generated featuring different terrain and attackers.

1.2 Original Vision

The game will be for the PC. This is because it is going to be pretty complex and PCs usually have screens large enough to display all the information the player needs. The mouse and keyboard allow for precise control and a lot of buttons to be used. PC players are also more likely to enjoy challenging *strategy* games, which is what this game aims to be.

It will be a single-player game. As stated, the small-scale gameplay will be a *tower defense*, but on a larger scale, the game will be *rogue-like*. This means that it will consist of individual runs, where the player will start from scratch every time. Their goal is to get as far as possible, trying to reach the final level and beat the game.

The game should have some story to motivate the player on their journey. The details of this story are not yet decided, but the game will likely be set in space. The player will travel forward through an unexplored galaxy trying to reach a destination specified by the story. The levels will take place on different planets, each with its own theme and unique terrain. This sci-fi theme allows for a lot of freedom in the designs of the levels, buildings, and attackers. It also works well with the mechanics of the individual levels or, as we will call them "battles".

The goal of a single battle is to gather enough fuel to continue to the next planet. The faster the player gathers the fuel, the sooner they win the battle. Fuel is gathered passively, but additional buildings can be built to speed up the process. In the meantime, the player has to defend against waves of attackers by building towers and using abilities. All of this costs materials and energy —

resources, which are generated by economic buildings. Thus, the player will have to balance their use of resources among defense, economy, and gathering fuel.

On their way, the player will encounter various events, shops, and other anomalies. They can present additional choices and provide the player with opportunities to gain various rewards or punishments. Some information about the upcoming battles will be revealed to the player — for example, the attacker types and general difficulty of the battle. Harder battles will provide better rewards. The path the player takes will not be linear, allowing them to decide which battles to fight and what to interact with. This way they can choose to take a bigger risk for a potentially bigger reward or to play it safe.

I probably forgot to mention something

1.3 Current Scope and Goals

The scale of the full game is quite large, too much for the scope of a bachelor's thesis. Furthermore, we will not consider the story, art, or sound design as that is not our area of expertise. Instead, the goal will be to create a functional prototype, which can be used to playtest the core gameplay. It will require some base content — several attackers and blueprints — so the game can be properly tested. The prototype will be prepared for future development so that more content can be added later.

Playtesting is very important because it can provide valuable feedback about the game's mechanics. This prototype will allow for testing the game to better adjust its design, tweak the mechanics, or make things more clear to the player. It is important to start with playtesting as soon as possible to avoid wasting time on content that relies on mechanics that will change.

The prototype will consist of all systems and mechanics necessary to play through a battle. This includes attackers, towers, abilities, and the economy. It will also include procedural generation of the battles including the terrain, attacker paths, and the makeup of the attacker waves. The player will be able to progress through battles and collect blueprints. However, there will be no map view to choose their path because for now, the progression will be linear. There will also be no events or shops, only battles. All the art and sound assets will be placeholders, but care will be taken to make everything as clear as possible to the player.

(this might be false) Additionally, the prototype will include a very rudimentary tutorial to explain the game's mechanics to the player. This is important because the game will be complex, and the player needs to understand how the game works to be able to play it properly.

The main goal of this thesis is to design and implement this prototype of the game, so it can be playtested. It will be implemented in the *Unity* game engine, using *C#* as the programming language.

2 Game Design

don't just document the design, but also the decisions behind it

- We want the game to be good, so we should think about the things that will make it good. It is for a specific audience.

2.1 Design goals

- these are the goals the game should achieve (each will have explanation — what and why)

1. strategic depth in each battle
2. strategic depth in each run
3. make various strategies viable
4. force exploration
5. provide a challenge

- some of our goals are achieved very well in other games of the relevant genres, so we can look at them for inspiration

2.1.1 Slay the Spire

- explain the game, use the following images

Figure 2.1 A fight in Slay the Spire.

2.1.2 Plants vs. Zombies

- explain the game, use the following images

2.1.3 Strategic Depth in Each Battle

- StS — you have to balance defense and offense — some enemies get stronger throughout the fight - refer to the fight image

Figure 2.2 Slay the Spire — map screen.

- PvZ — you have to balance defense and economy — in each level, you want plants that help you in the short term while you build your economy and plants which are more expensive, but effective once you can afford them

- in our game you have fuel mining as offense and defensive towers as defense, economic buildings and abilities, so you have to balance short-term and long-term power

2.1.4 Strategic Depth in Each Run

- StS — As you get better, you realize there is this interesting trade-off between short-term and long-term power - you want cards to survive the next few fights, but they will be duds later; you also want cards which will have greater synergy in the future, but don't help you right now.

- examples - iron wave, entrench

2.1.5 Make Various Strategies Viable

- StS — some cards interact in interesting ways that make them stronger ; there are cards and relics which fundamentally change the way your deck works (ex. barricade and entrench)

- PvZ — the interactions are not so strong, but you have to combine the plants such that you have no weak spots — cheap and expensive, for specific zombie types

- make blueprints that have unique effects that change the way you play

2.1.6 Force Exploration

- StS — you have to explore different strategies because you get different cards every time, some enemies ensure you are prepared for everything and intentionally break some strategies

- characters

Figure 2.3 A plants vs. Zombies level.

- PvZ — you have to explore different strategies because you have to adapt to different zombies and level environments, not too deep; after the campaign, three seed slots are selected for you.

- choose from random blueprints, various attackers with abilities and terrain types

2.1.7 Provide a Challenge

- StS — not easy to beat; after you beat the game, you can play on a higher ascension

- we will have something similar to StS

2.2 Battle

- overview viz section 1.2

- build stuff, send waves, use abilities

2.3 Attacker Movement

- free movement with individual pathfinding X

- too unpredictable for the player — The player needs to plan in advance in order to maximize the possibility to take calculated risks - solution: visualize enemy paths — Too many different paths - too much clutter; Still planning at most one wave in advance

- linear lanes (like in Plants vs. Zombies) X - not enough space for interesting building placement

Figure 2.4 Sunflowers and Potato Mines in Plants vs. Zombies. Sunflowers are necessary to fuel your economy, while Potato Mines are a cheap way to deal with the first few zombies.

Figure 2.5 *Iron Wave* and *Entrench* — cards in Slay the Spire.

Figure 2.6 A small portion of the many interesting cards in Slay the Spire.

- paths based on your buildings X - what if they block off the path?
- predefined paths ✓
- branching
- other qualities

2.4 World

- free placement X
- grid of tiles ✓- more granularity, easier to develop intuition, same for attacker paths
- hexagons X
- squares ✓
- 3D ✓- Simple and intuitive way to make the level itself more interesting Some towers won't be able to shoot uphill or downhill - more interesting decisions for the player
- Terrain types (for now, just one)
- Obstacles

2.5 Resources

for each subsection: what is it, why is it, how to acquire it, what are the constraints

2.5.1 Materials

2.5.2 Energy

2.5.3 Hull

2.6 Graphical User Interface

specify controls

Figure 2.7 All the plants of Plants vs. Zombies in the in-game almanac.

2.6.1 Waves Left and Fuel

2.6.2 Hull

2.6.3 Wave Preview

2.6.4 Materials and Energy

2.6.5 Blueprints

- what they represent
- cost and cooldowns
- rarities
- make them unique
- lenticular design

2.6.6 Info Panel and Selection

- what it looks like and what's on it
- select blueprints
- select buildings
- select attackers

2.6.7 Highlights and Range Visualization

2.7 Attackers

- move, have health
- sizes

Figure 2.8 Card reward screen in Slay the Spire. Here the player can choose on of three randomly selected cards to add to their deck.

- special abilities - passive, repeating, reactive

2.8 Buildings

- how and when to build, one per tile
- special building have special abilities
- main types:

2.8.1 Economic buildings

-provide resources

2.8.2 Towers

- attack attackers
- range
- targeting
- projectile types
- damage types

2.9 Abilities

- used mid-wave
- usually instant effects
- free placement, global placement, tile placement, use on a building

2.10 Camera controls

- zoom to look closely, rotate so the terrain doesn't hide stuff

Figure 2.9 Seed select screen in a rooftop level. Here, plants must be planted in flower pots, and most plants cannot shoot uphill.

2.11 Future Features

- run structure
- map
- events and shops
- difficulty levels
- unlocks

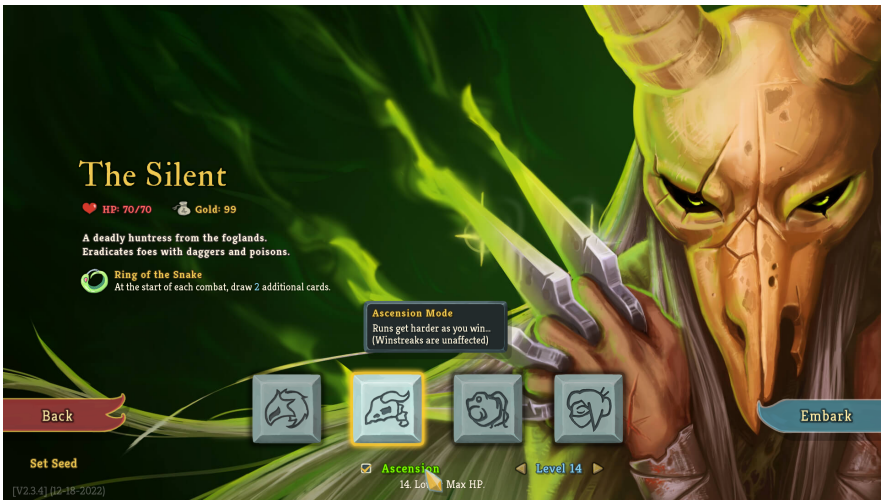

Figure 2.10 Character select screen in Slay the Spire. Here the player can also choose their ascension level, making the game harder.

3 Analysis

3.1 Game Engine

- unity because of familiarity
- mention the features we will use

3.2 Procedural Level Generation

- make each level unique
- one random seed dictates whole run
- in background thread
- debug visualizations (images)

3.3 Random Number Generation

- unity random X
- system random X
- custom random ✓
- why LCG

3.4 Path Generation

- how to make paths with the required qualities?

3.4.1 Initial Paths

- generate fixed number of paths with given lengths
- these are just plans to ensure paths of required length exist
- first pick *start points* from along the edges of the level (all paths end in the center of the level)
- choose randomly from positions with the correct parity (even / odd path length)
- spread them out by removing all positions near already chosen one
- then generate paths
- first trace it randomly, with bias away from the start and from path tiles
- simulated annealing
- there are just plans to make sure the paths exist and the terrain in a way that ensures that the paths are not blocked

3.4.2 Final Paths

- after terrain generation, the paths are traced for real
- guaranteed shortest length
- DFS, find all branches, but continue from bottom of the stack when a paths is found

3.4.3 Path Visualization

- line renderer ✓
- why does it look this way

3.5 Terrain Generation

- fractal noise X
- WFC ✓

this is gonna be multiple subsections

illustrate with images (from the videos I made)

- *slots* offset compared to *tiles*
- less distinct variants
- more control over transitions
- prepare *modules*
- generate grid of *slots*, mark them as *uncollapsed*
- each *slot* can become one of many *modules*
- the *modules* that can be placed in a given slot are its *domain*
- at the start each *slot* has all *modules* in its *domain*
- from the *domain*, compute all possible *boundary conditions*
- for example - there is a *module* in the *domain* with a cliff on its east boundary, so mark cliff to the east as possible
- mark all *slots* as *dirty*
- then repeat:
- propagate constraints
- for each *dirty slot*, until there are none:
- mark as *not dirty*
- find out which *modules* from its *domain* can be placed here and remove the rest from its *domain*
- decide only by neighbors' (orthogonal and diagonal) *boundary conditions*
- update *boundary conditions*
- if *boundary conditions* changed, mark all *uncollapsed* neighbors as *dirty*
- *collapse* a slot
- save the current *state* of all *slots* on a stack
- pick a *slot*
- pick one *module* from its *domain* at random
- weighted - provides control
- remove each other *module* from its *domain*
- update *boundary conditions*
- mark *uncollapsed* neighbors as *dirty*
- if a *slot* ends up with 0 *modules* in its *domain*, backtrack
- pop a prevoiusly saved *state* from the stack and revert to it
- remove the previously chosen *module* from the *domain* of the previously *collapsed slot*
- Which slot to collapse?
- fail fast approach
- prioritize slot with least options

- changed to slot with least entropy, because that's more accurate

- slots near most constraining modules were prioritized, making them more common and leading to repetitive terrain features

- better to just collapse a random slot

- in the end still weighted by entropy

- at first I tried to prefer slots with more entropy

- define overall structure first by sparsely covering the world, then fill in details

- often led to deep dead-ends with a lot of backtracking

- in the end, tiles with less entropy are preferred

- Limited backtracking depth

- usually when more backtracking is required, the search would take too long and it's faster to restart the algorithm

3.6 Resources and Obstacles

- after terrain generation, place blockers on tiles

- materials for the player to mine

- just rocks for variety - the player cant build on these

- set up as a few stages

- each stage has:

- one type of blocker (e.g. ore, small rocks, big rocks)

- *min* and *max* amounts

- *base chance* to place

- whether they can be placed on slanted tiles

- which Terrain Types they can be on (currently there is only one)

- *forces* - effect on chance based on already placed blockers

- for example: negative force with magnitude *m* from stage *s* means the chance to place a blocker on a given tile is decreased by *m/d* for each blocker placed in stage *s*, where *d* is its distance from the considered tile

- for each stage:

- repeat until at least min blockers have been placed (in this stage)

- for each tile without a path or blocker (in random order):

- if random number between 0 and 1 < modified chance:

- place the blocker of the given type

- if there are max blockers (placed in this stage), end the stage

- scattering

- unity physics engine X

- parallel

For the blockers, I didn't want repetitive obstacle models, so they are generated proceduraly by scattering many simpler models (decorations) on each tile

- first compute weights based on various factors (images!!!)

- distance to path

- height

- distance to other blockers

- customizable thanks to modular approach

- then scatter decorations in stages, each stage again having one type of decoration and many parameters

- for each tile in random order repeat x (specified for this stage) times:

- pick a random position within it
- calcualte the weight at this position (based on settings)
- check that it is greater than some threshold (based on settings)
- calculate the minimum distance to other decorations (from weight, based on settings)
- check that the position is far enough from other decorations
- calculate the decoration size (from weight, based on settings)
- place the decoration on this position, with the given size

3.7 Terrain Types

- what information is tied to the type
- why txt (inspector was not as legible)

3.8 World Builder

- builds the world from the generated data, it needs to be done in the main thread

3.9 Attacker Wave Generation

- creates a randomized plan of waves
- two types of waves
- combine different attackers in sequence
- combine different attackers in parallel (only possible with multiple paths, rarer)
- each wave gets some throughput budget and buffer
- each attacker has a given cost
- when planning a wave, select attackers and spacing, such that the througput budget is exceeded
- for each attacker subtract the throughput overshoot from buffer
- fit such that as much of buffer gets used without going over

3.10 Simulation

- use fixed updates for game logic
- why?
- 20Hz = fixed time step 0.05s
- options to speed up or possibly pause - changing fixed update rate - not yet implemented

3.11 Visuals and Interpolation

- interpolate positions and visuals on Update
- many visuals are game-speed agnostic - TODO: use unscaledDeltaTime

- I thought about some custom mini-framework for this, but many of the simulated variables the visuals are based on should be handled on case-by-case basis

3.12 Attacker Targeting

- Towers use it to acquire targets
- handles which Attackers are in range and which one is chosen as the current target
- can require line of sight to the enemy
- different targeting types
- rotation
- heights
- possibly ensure a trajectory
- preferred target (configurable)

3.13 Range Visualization

- IMAGES!!
- Draw the range on the terrain mesh
- Draw on which parts of paths will Attackers be targeted
- green - all sizes
- yellow - only large
- Terrain shader uses compressed texture format instead of raw texture
- Options:
- quadrant compression format, 2bytes per node
- less CPU time, because the data is already in this format
- up to 48KiB per frame
- more GPU time
- 256x256 texture, 1byte per pixel
- more CPU time
- 64KiB per frame
- fast on GPU
- only 1 channel - cannot interpolate
- mipmaps -> one additional state
- less CPU time
- 33% more data
- more pixels per byte
- possible future optimization
- less data
- more difficult indexing and stuff both on CPU and GPU
- interpolation could work with more than one channel and without mipmaps

3.14 Game Commnds

- we want various components to modify how other components function

- examples
- also react to events as a bonus

3.15 Blueprints

- why are they implemented this way

3.15.1 Attacker Stats

- blueprints for attackers

3.15.2 Dynamic Descriptions

- explain what things do and their stats
- attackers and blueprints
- dynamically reflect the changes made by other components

4 Developer Documentation

4.1 Unity

4.2 Scenes

4.2.1 Battle

4.2.2 Loading

4.2.3 Main Menu

4.3 ???

5 User Documentation — Designer

5.1 Terrain Types

5.2 Blueprints

5.2.1 Buildings

5.2.2 Towers

5.2.3 Abilities

5.3 Attackers

5.4 ???

6 User Documentation — Player

6.1 ?Introduction

6.2 ?Controls

6.3 ?Mechanics

7 ?Playtesting

Conclusion

List of Figures

2.1 A fight in Slay the Spire. 12
2.2 Slay the Spire — map screen. 13
2.3 A plants vs. Zombies level. 14
2.4 Sunflowers and Potato Mines in Plants vs. Zombies. Sunflowers are necessary to fuel your economy, while Potato Mines are a cheap way to deal with the first few zombies. 15
2.5 *Iron Wave* and *Entrench* — cards in Slay the Spire. 15
2.6 A small portion of the many interesting cards in Slay the Spire. . 16
2.7 All the plants of Plants vs. Zombies in the in-game almanac. . . . 17
2.8 Card reward screen in Slay the Spire. Here the player can choose on of three randomly selected cards to add to their deck. 18
2.9 Seed select screen in a rooftop level. Here, plants must be planted in flower pots, and most plants cannot shoot uphill. 19
2.10 Character select screen in Slay the Spire. Here the player can also choose their ascension level, making the game harder. 20

List of Tables

List of Abbreviations

A Attachments

A.1 First Attachment