University of Tartu

Institute of Technology

Computer Engineering Curriculum

Johannes Voldemar Langsepp

Pikseon – Procedural Dungeon Crawler

Bachelor's Thesis (12 ECTS)

Supervisor: Daniel Nael, MSc

Tartu 2024

Pikseon – Procedural Dungeon Crawler

Abstract: This thesis describes the design and development process of a dungeon crawler roguelike game with procedurally generated levels called *Pikseon.* The generation process uses a custom implementation of the Wave Function Collapse algorithm. The thesis also gives an overview of how the game was designed and how it was tested for usability. The feedback was analysed and plans for improvement are laid out.

Keywords: Computer game, game development, game design, Unity, procedural generation, usability testing, Wave Function Collapse

CERCS: P170 Computer science, numerical analysis, systems, control

Pikseon – Protseduuriline Kooparoomaja

Lühikokkuvõte: Lõputöö kirjeldab protseduuriliselt genereeritud "*roguelike"* kooparoomaja arvutimängu "*Pikseon"* arenduse ja disaini protsessi. Genereering põhineb kohandatud lainefunktsiooni kokkulangemise algoritmil. Lõputöö annab ka ülevaate mängu disaini valikutest ning sellest, kuidas seda testiti kasutatavuse osas. Tagasisidet on analüüsitud ning parandus- ning parendusvõimalused on välja toodud.

Võtmesõnad: Arvutimäng, mänguarendus, mängudisain, Unity, protseduuriline genereerimine, kasutatavuse testimine, Lainefunktsiooni kokkulangemise algoritm

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of contents

1 Introduction

The gaming industry is highly valued, with revenues in 2023 expected to reach 249.58 billion USD and projected to grow over 300 billion USD by 2025 [1]. Although most of this revenue comes from the mobile games and in-game advertising markets [1], the computer games market is growing as well, based on the fact that the number of games released on Steam has now passed 200 thousand. Last year 14457 games, the highest per year ever, were released on Steam in 2023[1].

On Steam[2], the number of games with the tag 'Rogue-like' released per year has grown each year since 2017[3]. Roguelike games are commonly focused on runs ending with death and the progression between runs [2]. This thesis describes the design and development process of a roguelike game called *Pikseon*. The game has a heavy emphasis on procedural dungeon generation, making use of a custom-built Wave Function Collapse algorithm. The game was tested for its usability and the necessary technical and usability improvements were mapped out.

Chapter 2 focuses on how genres of video games are handled in this thesis and defining the video game genres used. The chapter also contains an analysis of games similar to *Pikseon*. Chapter 3 discusses the ideas and processes behind the design of a video game, going into detail about the unique concept within *Pikseon* and the game's level design and initial goals of the project. Chapter 4 explores how the design was implemented and how the Unity engine was used for developing the game. The chapter gives a thorough explanation of how the Wave Function Collapse algorithm implemented into *Pikseon* operates and what the capabilities of the procedural generation system developed are. The chapter also gives an overview of all third-party assets used for the project. Chapter 5 outlines how the testing for *Pikseon* was conducted and contains an analysis of the testing results and how the game could be improved further.

[1] https://steamdb.info/stats/releases/
[2] https://store.steampowered.com/
[3] https://steamdb.info/stats/releases/?tagid=1716

The Appendixes' content is as follows:

- Appendix I - Overview of the terms used within the thesis,
- Appendix II - List and description of the files accompanying the thesis,
- Appendix III - Instructions on how to launch the game,
- Appendix IV - Information on the game's source code and how to access it,
- Appendix V - The licence.

The use of artificial intelligence in this thesis is limited to suggestions made by Google Docs[4] and use of GitHub Copilot[5] within Microsoft Visual Studio 2022[6]. All of the more significant uses of GitHub Copilot are documented within the source code available in Appendix IV.

[4] https://www.google.com/docs/about/
[5] https://github.com/features/copilot
[6] https://visualstudio.microsoft.com/

2 Background

A 2015 study [3] showed that 74% of users consider the genre of a video game when seeking new games to play. This chapter describes relevant video game genres and the games which were analysed during the creation of *Pikseon*.

2.1 On Video Game Genres

Although most users [3] consider genres an important part of seeking games to play, the definitions of video game genres are often not clear. While genres of more mature forms of media, like literature and film, are based on observable and objective characteristics, video game genre classification characteristics vary widely over time. Classifying video games into genres has been attempted, for example Wolf [4] categorised video games into interactive genres based on the gameplay. Many different taxonomies, including the one by Wolf, have appeared over the decades, yet this formal academic work is often not followed by game developers or consumers [5]. In this thesis, video game genres are viewed as subjective labels, based more on how the game feels to players, rather than specific categories based on objective characteristics. The described genres for games mentioned in this thesis are based on the respective games' Steam Tags.[7] Steam Tags are custom labels applied by Steam users.[8]

[7] https://steamdb.info/tags/
[8] https://store.steampowered.com/tag/

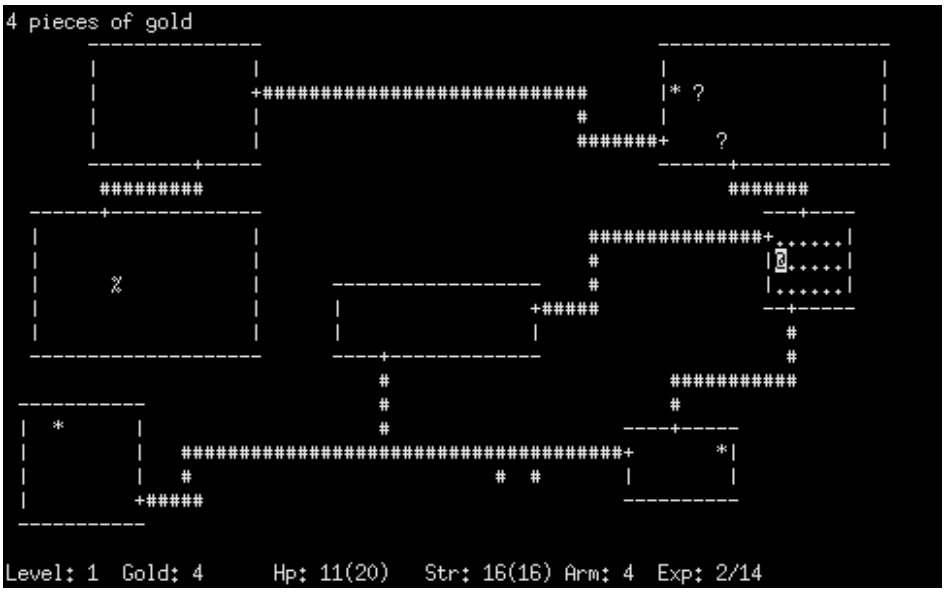

Figure 1. Screenshot of Rogue's 1980 version, showing a procedurally generated dungeon [9].

According to Andre Alves Garzia [6], roguelike games as a genre are widely agreed to have started from the game *Rogue*[10], developed in the early 1980s. The game world was drawn onto the screen using ASCII characters and offered infinite replayability thanks to its randomly generated dungeons (shown on Figure 1). In 2008, developers at the International Roguelike Development Conference defined the *Berlin Interpretation* as a way to objectively assess whether a game is a roguelike or not. They laid out high-value factors and low-value factors. A game does not need all of the factors to be a roguelike, but it gives a good overview of what the roguelike development community considered as important at that time. The high-value factors are:

- random generation – levels are generated randomly.
- permadeath – once a character dies, the game run is over.
- turn based – the user has time to plan their moves, the game waits for input.
- grid based – all of the entities in the game are placed on a grid.
- non-modal – all of the game functionality is possible on the same screen, no need to switch modes.
- complexity – the game has enough complexity to allow multiple solutions to problems.
- resource management – resources are limited and the player has to manage their use.

[9] https://upload.wikimedia.org/wikipedia/commons/0/0c/Rogue_Screenshot.png
[10] https://store.steampowered.com/app/1443430/Rogue/

- hack’n’slash – killing many enemies is integral to the game.
- exploration and discovery - exploring different environments and discovering objects is part of the game.

And the low-value factors are:

- single player character – the user controls a single character throughout the game.
- enemies and players are similar – the same principles apply to both players and enemies.
- tactical challenge – it is important to approach tactically to solve situations.
- ASCII display – the interface is built fully with ASCII characters.
- dungeons – the game is dungeon based, with multiple levels and rooms.
- numbers – the character statistics are deliberately shown as numeric values.

Out of these factors, *Pikseon* incorporates many of them, including random generation, permadeath, non-modality, complexity, hack’n’slash, exploration, single character, similarity of players and enemies and dungeons.

The Berlin Interpretation is criticised for being dated and not representing the current state of the genre [6]. For this thesis, it serves merely as an example of how roguelikes have been defined in the past. Based on this interpretation, *Pikseon* could be classified as a roguelike, since it fills many of the criteria. At the same time, based on this interpretation *Pikseon* could be classified as not a roguelike, since it does not comply with quite a few of the factors.

In order to classify such games into a genre of their own, many people use the term roguelite instead. Roguelites are usually interpreted as having some major characteristics of roguelikes but also deviating in some aspects. A common example is the lack of permadeath or the ability to progress between runs of the game, often classified as a sort of meta-progression [2].

According to an article by Keith Stuart [7], Dungeon Crawlers are usually fast-paced games with focus on exploration and combat. Narrative is often left in the background or left out in its entirety. The environment is usually labyrinth-like, with older games using grid-based systems. Dungeon crawlers are often built to achieve compelling gameplay.

2.2 Analysis of Similar Games

For inspiration before and during the development of *Pikseon*, *Curse of the Dead Gods*, *BPM: BULLETS PER MINUTE*, and *Vampire Survivors* were looked into. These games contain similar mechanics and ideas to what was envisioned for *Pikseon*.

Curse of the Dead Gods[11] (shown on Figure 2), released in 2021, is an action roguelite dungeon crawler developed by Passtech Games[12] and published by Focus Entertainment[13]. Single dungeon runs contain familiar elements of roguelikes, but this game has advanced systems of progression between the runs. When launching the game, the player will first be able to roam around a central hub, where they can purchase upgrades and make choices for the next dungeon run. This element was an inspiration for developing a similar hub to *Pikseon*, although an in-game location is replaced with a menu-based system. The feature did not make it into the testing build of *Pikseon.*

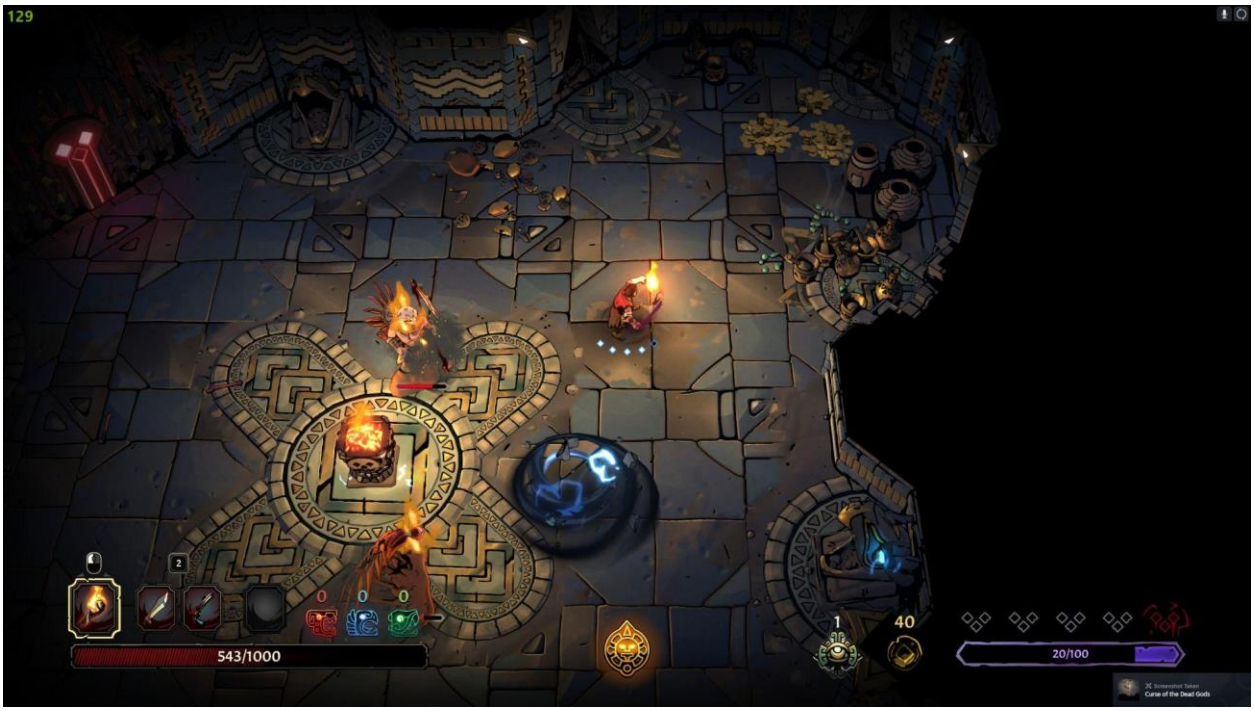

Figure 2. Screenshot of Curse of the Dead Gods.

[11] https://store.steampowered.com/app/1123770/Curse_of_the_Dead_Gods/
[12] https://www.passtechgames.com/
[13] https://www.focus-entmt.com/en

BPM: BULLETS PER MINUTE[14] (shown on Figure 3), released in 2020, is a rhythm-based FPS roguelike developed by Awe Interactive[15]. Although the gameplay is coherent with the perma-death principle of roguelikes, there are unlockable features that carry on between individual runs, so elements of a roguelite are also present. When the player reaches certain milestones, new abilities and characters to play as will be unlocked in the main menu when starting a new run. This feature was an inspiration for developing multiple characters for *Pikseon*.

Figure 3. Screenshot of BPM: BULLETS PER MINUTE.

[14] https://store.steampowered.com/app/1286350/BPM_BULLETS_PER_MINUTE/
[15] https://www.aweinteractive.com/

Vampire Survivors[16] (shown in Figure 4), released in October 2022, is a two-dimensional top-down survival game, featuring minimalistic gameplay and some elements of roguelikes developed and published by poncle[17]. *Vampire Survivors* has been received very well by players, having 98.6% positive reviews[18] on Steam as of February 2024. The objective of the game is to survive in an infinite world as long as possible while the difficulty constantly increases. This system is not used in *Pikseon*, which instead implements finite dungeons. When the player character dies, the player will need to start a new game. This was an inspiration to develop a similar roguelike-like approach in *Pikseon*.

Figure 4. Screenshot of Vampire Survivors.

These games were used as an inspiration for the design of *Pikseon*, with the next chapter covering the design decisions of *Pikseon*.

[16] https://store.steampowered.com/app/1794680/Vampire_Survivors/
[17] https://poncle.games/
[18] https://vginsights.com/game/1794680

3 Game Design

In his Book of Lenses, Jesse Schell [8] said: "*Game design is the act of deciding what a game should be*." In this chapter the decisions around various design elements of *Pikseon* are laid out.

3.1 The Unique Gameplay Element

The mechanics of a game are the rules and procedures that define how the players can interact with your game [8]. One way to make a game stand out is to have it contain something unique, something new. This chapter describes the design of a unique gameplay element for *Pikseon* and how it was designed.

The main inspiration behind *Pikseon* was the idea of using pixelation on separate objects within the game. The idea for pixelating separate objects stemmed from using the Beautify 3[19] asset while developing a pixelated racing game (*ProceduRACE*[20]) for a game jam. In *ProceduRACE*, the whole screen is pixelated using post-processing options in Beautify 3, but it lacks support for per-object pixelation. This meant that a new solution was needed to allow per-object pixelation effects for *Pikseon*.

[19] https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/beautify-3-advanced-post-processing-233073

[20] https://yanddalf.itch.io/procedurace

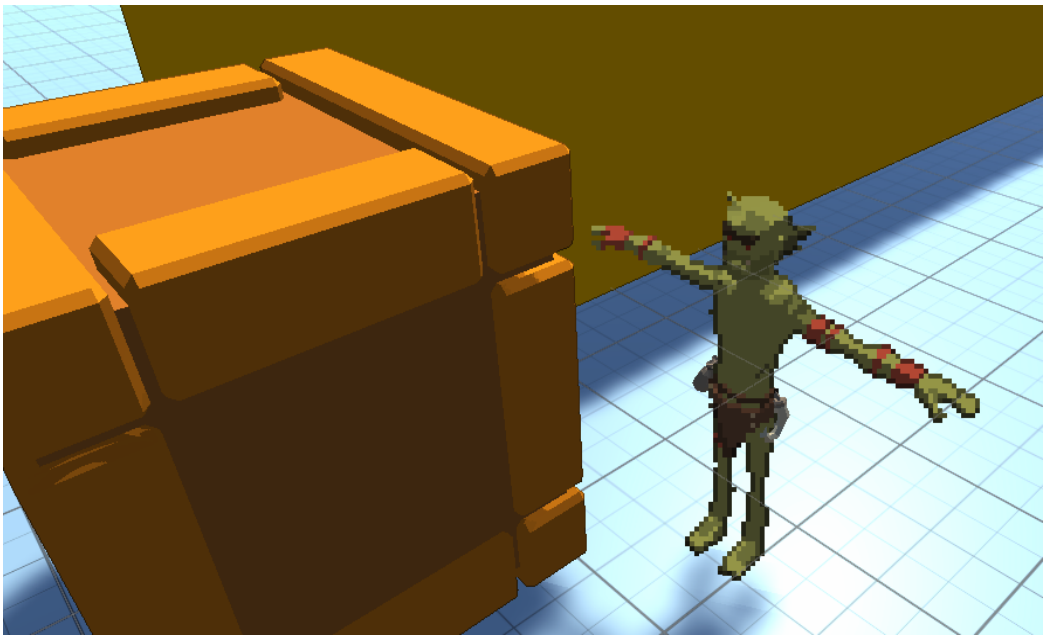

Figure 5. Pixelated enemy next to an un-pixelated crate.

When searching for existing assets from the Unity Asset Store[21], two assets were found that seemed to allow per-object pixelation capabilities: Pixelate[22] and ProPixelizer[23]. After a more thorough investigation, the former clearly was not suited for the task, as it is meant to develop pixelated 2D animations, rather than pixelate 3D objects in runtime. ProPixelizer however appeared to offer exactly what was needed to implement the per-object pixelation, example shown on figure 5, that supported lighting, animations and runtime modulation of the pixelation amount [9].

In *Pikseon*, the player is able to manipulate the pixelation of some of the objects and entities in the world around them. This is achieved with a visual cue of pixels flying away from objects or towards objects. The less pixelated objects are, the "healthier" or less broken they are. In order to un-pixelate (or in other words, restore to a lesser state of pixelation) objects, the player will need to spend *pixls* (an in-game currency, not to be confused with pixels). Pixls can be gained by defeating enemies and destroying obstacles. When obstacles or enemies are damaged, their pixelation levels will increase.

[21] https://assetstore.unity.com/
[22] https://assetstore.unity.com/packages/add-ons/pixelate-194727
[23] https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/propixelizer-177877

3.2 Level Design

"Levels are the space where a player explores the rules and mechanics of a game [10]."

Level design can be classified as a form of "invisible design", since players can see mechanics in games, but they often miss how these mechanics are enhanced by level design [11]. As Michael Salmond [11] said, "*Without good level design, there is no game*." For example, the jumping mechanic of a game might go unnoticed without levels that require or promote jumping. Level designers work on creating maps and worlds and then populate those with everything from obstacles to objectives [12].

The level design of *Pikseon* is loosely grid-based. The levels consist of rooms connected by corridors, which are always connected to each other in one of four directions. These directions are classified internally as north (positive Z axis in Unity world space), south (negative Z axis), east (positive X axis) and west (negative X axis). Although the placement is confined to a grid, the size of the rooms and corridors does not need to be consistent, as shown on figure 6. The inner workings of this mechanism are explored further in chapter 4.3.

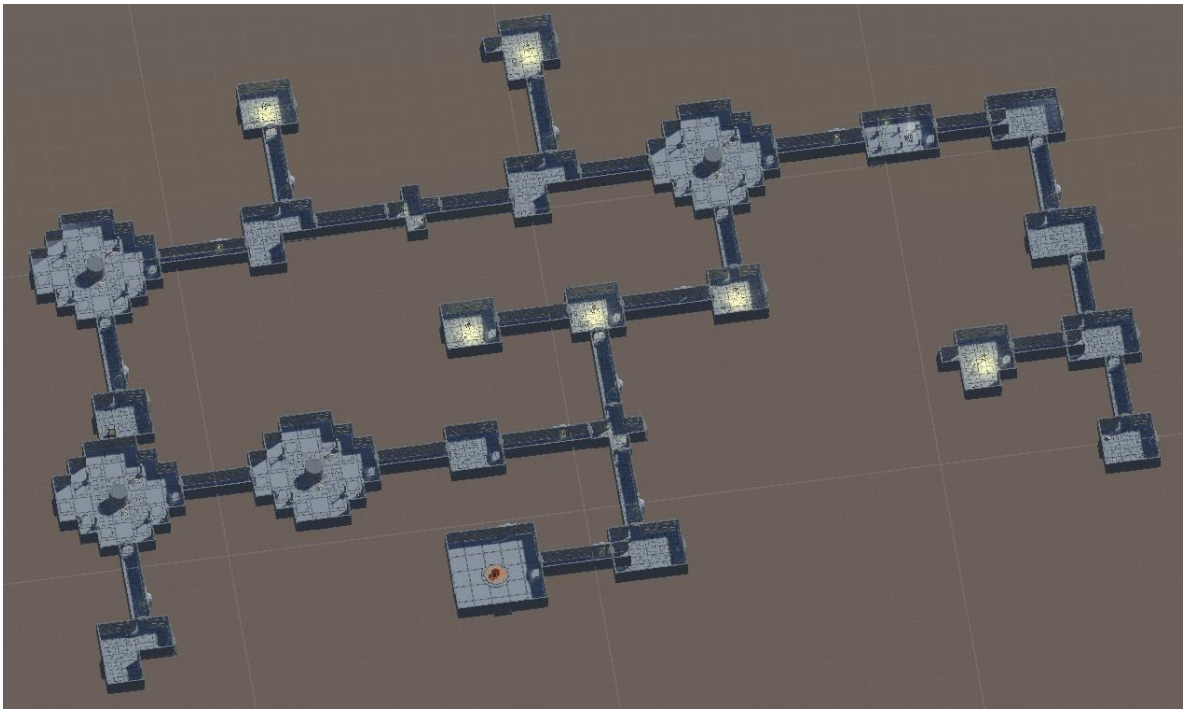

Figure 6. Picture of a generated dungeon.

Each room has up to 4 connection points, although some rooms have only one connection. For example, the **End Room** (a room which serves as the exit point for the given level, prototype shown on Figure 7) must have exactly one connection to the rest of the dungeon. Depending on the level, the end room can contain either a puzzle, a boss or just a treasure. Each level has a starting point - the **Start Room**. This room is where the player will start the game. During gameplay, the player's main objective is to reach the End Room from the Start Room. Thus, it makes sense to have them not connected directly to each other. For this, **Standard Rooms** fill out the rest of the dungeon. These rooms come in different variations and contain most of the enemies and treasure.

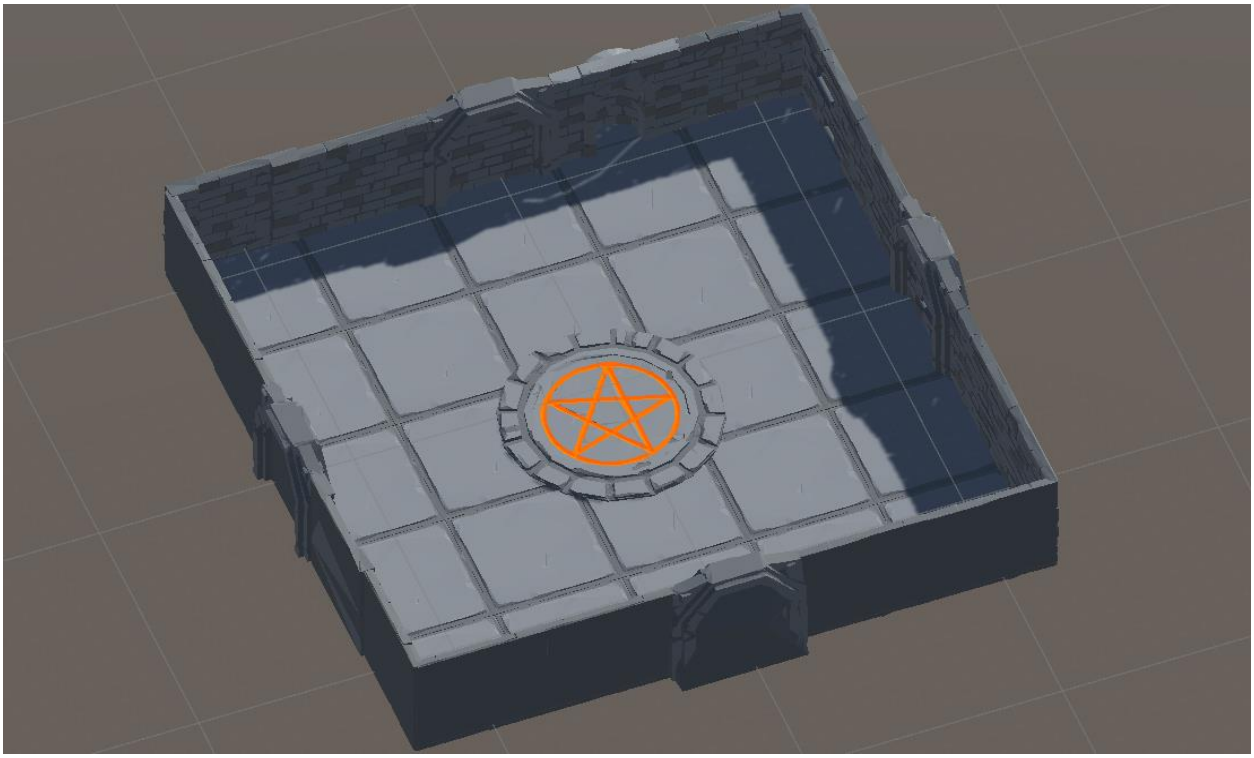

Figure 7. Prototype of an End Room

In general, the level progression of *Pikseon* comes down to the player starting from the Start room and progressing through exploration towards the End room. This gameplay loop has been designed with modularity in mind, as more rooms can be added without much changes to the underlying algorithms.

With this design, it is required that there is always a connection from the start to the end, as well as requiring all rooms to be connected to the whole dungeon. The levels of *Pikseon* are designed to be without loops – as a graph the rooms should lay out as a spanning tree. Some rooms are designed to only have one entrance, such as the End room, but possibly in the future for example also a Shop room.

3.3 Initial Plans

When the plans for this project were laid out, the ideas were quite grand. The initial plans included multiple stages of testing and a release on itch.io[24], possibly even on Steam[25]. Initially, the game was envisioned to contain a central hub and multiple different 'stages' or areas, which would have different and distinctive visual styles.

At around the midpoint of the development of *Pikseon* for this thesis, it became apparent that those ideas were out of scope for this project. The focus was narrowed down to developing a roguelike-style game with a single visually distinct 'stage' with a special emphasis on the procedural generation of levels.

For this thesis, *Pikseon* will not reach a release-ready state on either itch.io or Steam, but the design and development have been held with a wide goal in mind. In chapter 5, the possibilities for future improvements are discussed further.

[24] https://itch.io/
[25] https://store.steampowered.com/

4 Implementation

This chapter covers the software implementation of *Pikseon*. An important part of the early stages of game development is selecting an engine to work with [13].

Majority of the development of *Pikseon* was centred around building a procedural generation system. This system is designed to generate dungeons with near infinite possibilities. While designing this generation system, a big emphasis was put on making it very modular, with easily configurable parameters and components. The generation system uses a modified Wave Function Collapse algorithm as its foundation. The output of the algorithm is processed and then used to generate a dungeon in 3D.

4.1 The Engine

Pikseon has been developed using the Unity[26] game engine. Scripts were written in C# using Visual Studio 2019 and Visual Studio 2022. Game engines are frameworks built to ease the development of video games [13]. They usually contain many essential systems, like physics, graphics, audio and AI, removing the need for the developers to build these systems from scratch. Selecting an engine often dictates much of the future of a game's development, because most of the work will be done within the engine [13]. Some developers build in-house engines, while others use popular existing engines [13]. The selected engine also has a wide selection of third-party tools available, the use of which is detailed in chapter 4.2.

One such popular game engine is Unity. As of 2022, according to a survey by SlashData [14], Unity is the most popular game engine among game developers. 38% of developers use Unity as their primary game engine, with another 28% saying they also use it, although not as a primary [14].

Here is a list of features provided by Unity: [15]

- Rendering works with both 2D and 3D graphics, with 3D cameras supporting perspective and orthographic projection viewpoints.

[26] https://unity.com/

- The physics engine of Unity allows developers to easily include gravity and manage collisions between objects. Objects using physics rules can be moved by applying forces, instead of manually translating their coordinates in world space.
- Developers can create custom behaviours for objects through scripting in C#.
- A built-in sound engine with tools and 3D audio support is provided.
- Animation workflow is made easier by Unity's *Mecanim* animation system, which makes it easy to use humanoid animations on objects with humanoid avatars. Custom avatars and animations are also supported.
- The interface of the engine enables many of the engine's functions to be used without having to write code. Shown on figure 8.
- For AI navigation, Unity provides a system based on NavMeshes.
- Many user-made tools and assets for developers to extend the functionality of the engine are available on the Unity Asset Store[27].

Unity was chosen for this project, because the author of this thesis has previous experience with Unity and it is a very versatile and easy to use engine[28].

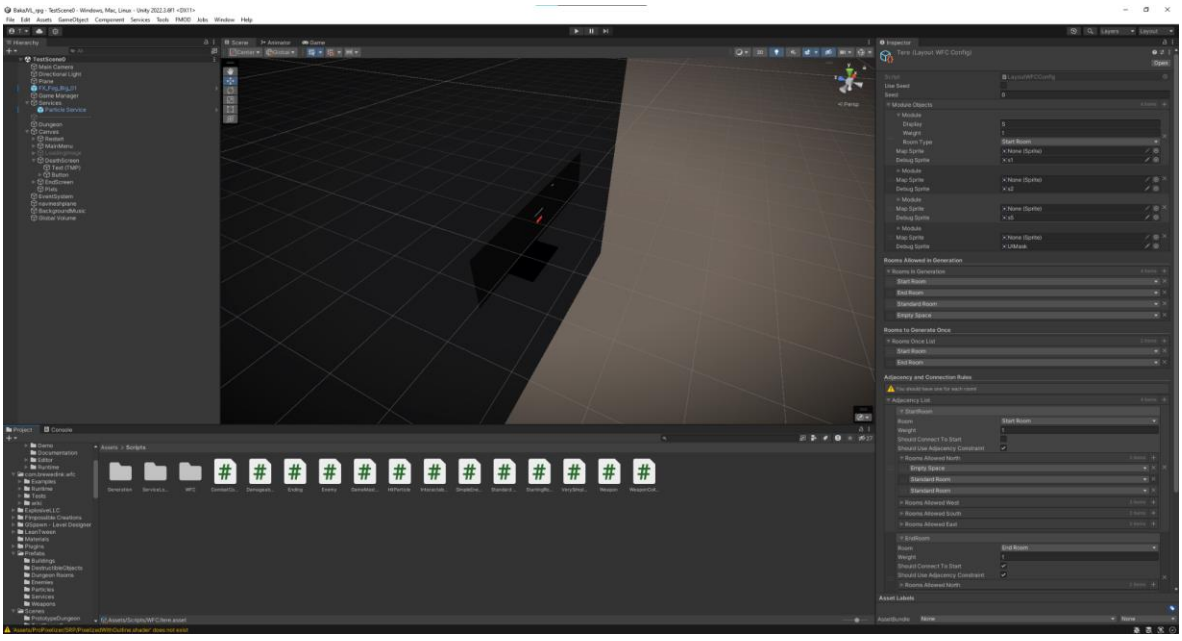

Figure 8. Interface of the Unity Editor.

[27] https://assetstore.unity.com/

[28] https://www.zazmic.com/choose-game-engine/

4.2 Third Party Assets

When developing games as a solo developer, it is often advantageous to make use of tools already in existence, as it saves much valuable development time[29]. Use was made of many different third-party tools, many from the Unity Asset Store, during the development of *Pikseon*. The author has all the rights and licences to use all of the assets listed below commercially. Assets with licences prohibiting redistribution are not available with the source code in the accompanying files, more information in appendices II and IV.

Table 1. Used third-party assets with descriptions and URLs.

Asset	**Description**	**URL**
Synty Studios 3D Model packs	Multiple packs of 3D models from Synty Studios. Prototype and Dungeons are used.	https://syntystore.com/collections/polygon-series
ProPixelizer	Rendering asset used to achieve pixelation effects which are essential to the aesthetic of the game.	https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/propixelizer-177877
Legs Animator	Animation asset, used to automatically adjust the position of legs on humanoid characters through an implementation of Inverse Kinematics (IK).	https://assetstore.unity.com/packages/tools/animation/legs-animator-154245
Wave Function Collapser	General base implementation of WFC, has been heavily modified to be of use for *Pikseon*.	https://assetstore.unity.com/packages/tools/modeling/wave-function-collapser-193890

[29] https://retrostylegames.com/blog/what-are-assets-in-game-design

GSpawn - Level Designer	World creation tool, used for assisting in the creation of the rooms and their variations that will be spawned procedurally.	https://assetstore.unity.com/packages/p/gspawn-level-designer-45021
LeanTween	Animation tool, used to programmatically induce and control tweening, animating the keyframes between fixed positions [16].	https://assetstore.unity.com/packages/tools/animation/leantween-3595
Ultimate Game Music Collection	Collection of ambient music and sound effects.	https://assetstore.unity.com/packages/audio/music/orchestral/ultimate-game-music-collection-37351
RPG Character Mecanim Animation Pack FREE	Generic animations for use with various characters.	https://assetstore.unity.com/packages/3d/animations/rpg-character-mecanim-animation-pack-free-65284
FMOD for Unity	Unity plugin for using the FMOD audio system. More details in chapter 4.4.	https://assetstore.unity.com/packages/tools/audio/fmod-for-unity-161631
Beautify 3	Post-processing asset, which enables certain release-ready visual effects to be achieved with only minor tuning.	https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/beautify-3-advanced-post-processing-233073

4.3 Dungeon Generation

In order to generate dungeon levels for *Pikseon*, multiple steps are taken. First, the configured rooms are fed into a modified Wave Function Collapse algorithm. Then, the generated grid is filtered and processed to be usable as a dungeon layout. Lastly, the processed layout is used to generate the dungeon level in 3D space.

Multiple different ways to generate dungeons were considered at the beginning of the development. The first considered approach was to generate the whole dungeon, including placement of 3D models, with Wave Function Collapse. This approach was shelved due to the heavily increased difficulty of the needed algorithm and my opinion that the dungeons, although being more random, would feel less random and with less variation. Consequently, an approach was taken which splits the generation of the layout and the generation of the rooms apart. The layout is generated with Wave Function Collapse and is very easily modifiable and the rooms are randomly picked from a predefined list, with additional random generation within the rooms possible.

The generation system is designed with modularity in mind. It is easy to add dungeon rooms (some examples shown on figure 9) to the generation and modify the parameters of how the Wave Function Collapse works. This was a major emphasis while designing the system, as this enables major future improvements to be built on this foundation.

Figure 9. Prefabs of dungeon rooms that are generated in Pikseon.

Adding different styles of dungeons or more custom types of rooms to the generation is made easy by being configurable from an interface within the Unity editor itself, shown on figure 10. Sprite options are currently not used, the feature is implemented to allow for easy map generation as a future improvement for the game.

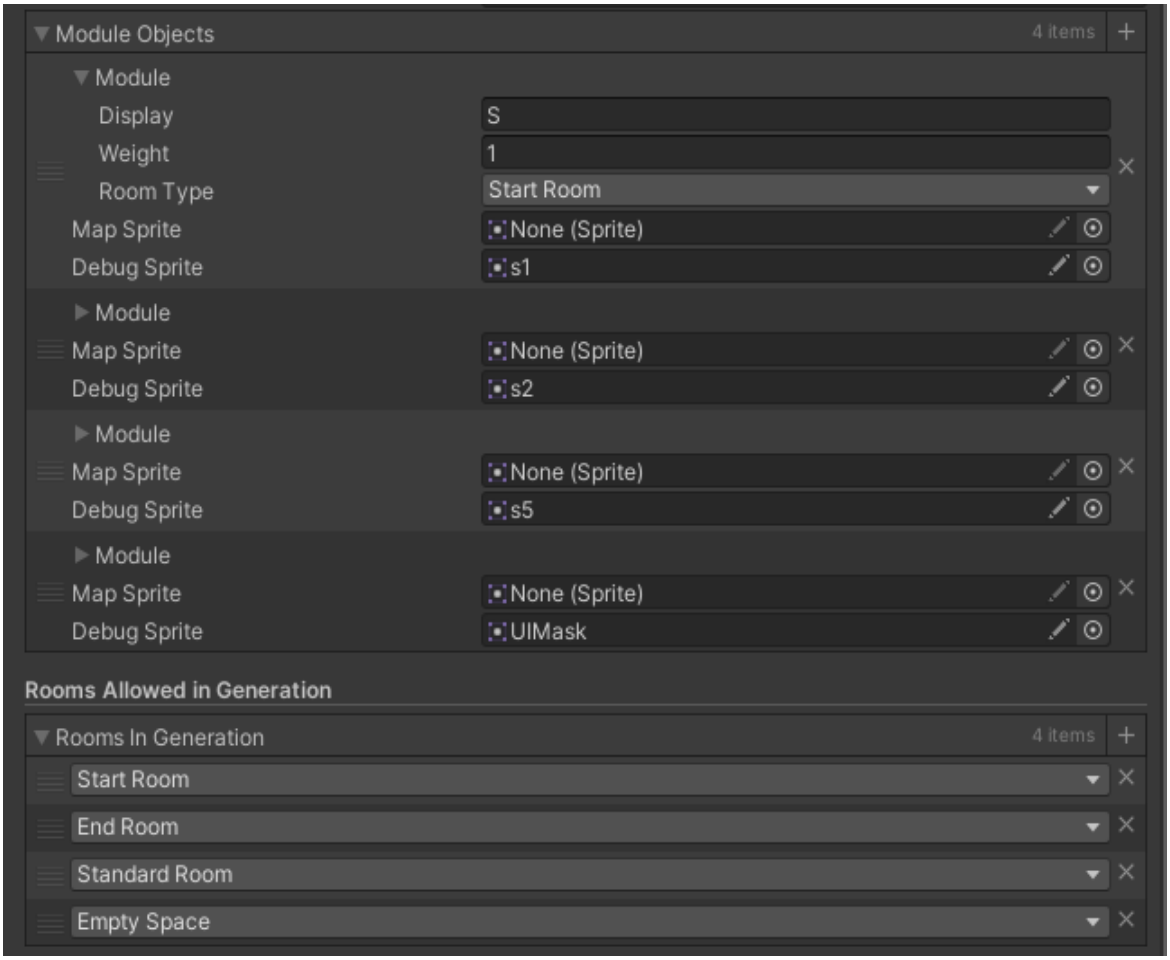

Figure 10. Interface for changing the room generation options for the Wave Function Collapse algorithm within the Unity editor.

4.3.1 Wave Function Collapse

Wave Function Collapse (WFC) is an algorithm initially developed by Maxim Gumin[30] to generate larger bitmaps that are locally similar to an input bitmap [17]. WFC is a constraint solving algorithm (CSA), the likes of which are uncommon for procedural content generation. The goal of a CSA is to solve constraint satisfaction problems and return a result where no constraints are violated [18]. In many implementations including this one, WFC propagates using a minimal entropy heuristic (illustrated by figure 11), solving the slots with the smallest superposition first [17].

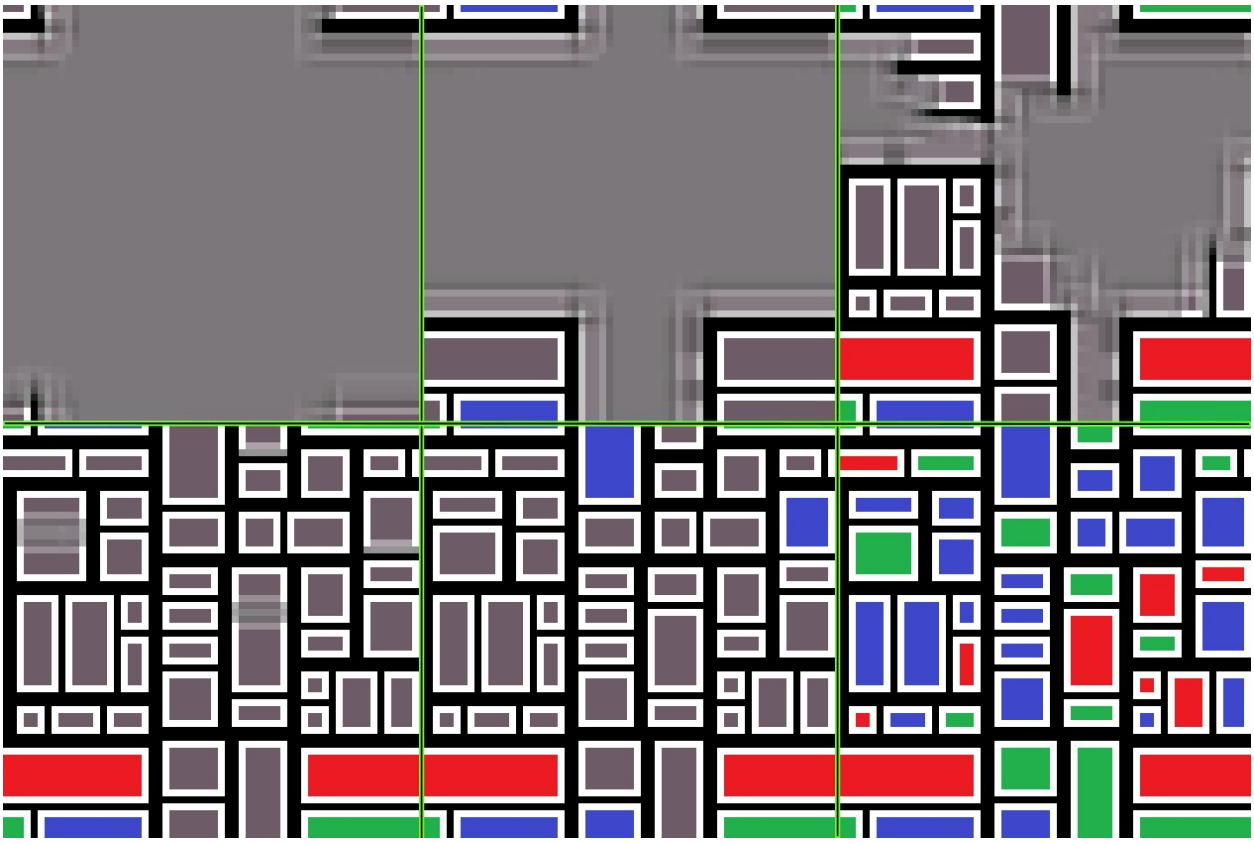

Figure 11. Illustration of the minimal entropy heuristic in WFC propagation [WFC].

The working principle of Wave Function Collapse is inspired from quantum mechanics, although the coefficients in the superposition are real numbers instead of complex numbers [17].

[30] https://github.com/mxgmn

For understanding the WFC implemented in this thesis, the following definitions are needed:

- **Space** - The structure, essentially a directional graph, where the Wave Function Collapse takes place. This structure holds slots. In the graph data structure, these can theoretically be connected in any way, but this thesis uses a grid-based connection structure. In the example of figure 12, the space contains all the slots.
- **Slots** - A location in the space, initially containing every single possible module in a superposition. After the algorithm has finished successfully, every slot will contain a single module. In figure 12, most slots contain a superposition of different modules, with only slot 5 having collapsed to a single possibility.
- **Module** - A value that each slot could have. This could be a number, a group of pixels or any other data object. In this thesis, the modules are different dungeon rooms.
- **Constraint** - A rule attached to a module, which eliminates the module from the slot's superposition if the rule is not followed. This thesis implements a few different constraints, but the simplest example is an adjacency constraint, which checks the state of neighbouring slots.

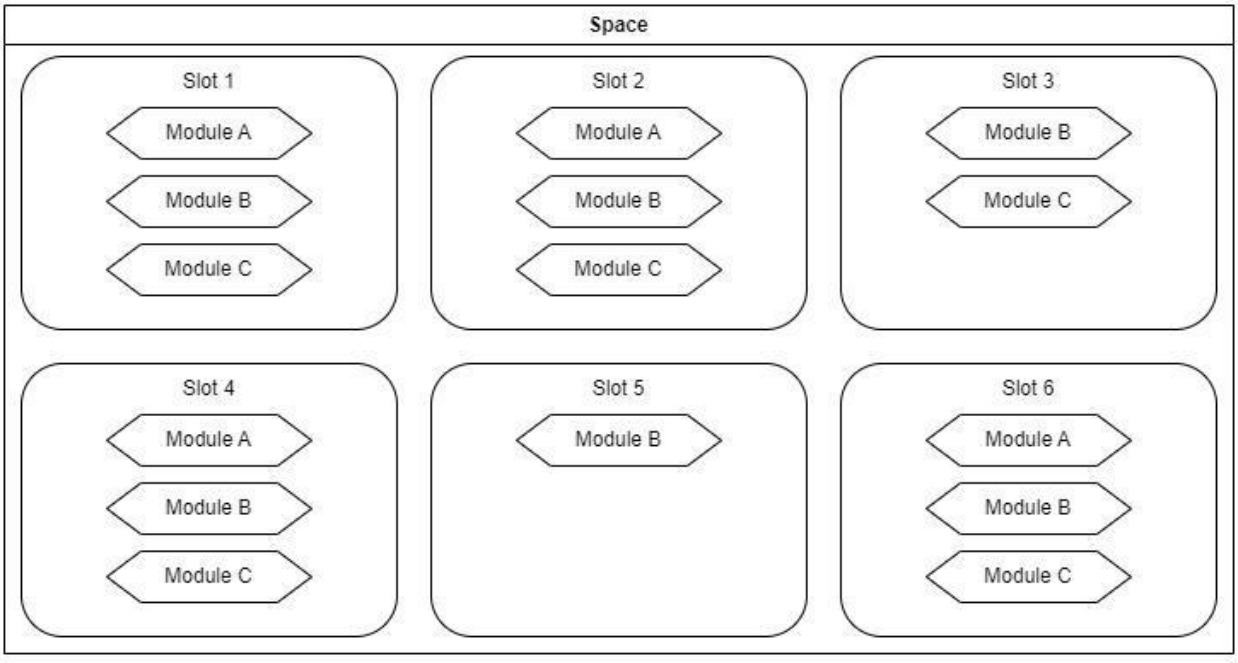

Figure 12. Example representation of the internal data structures of WFC.

For *Pikseon*, the WFC implementation is built using the Wave Function Collapser asset (see table 1 in chapter 4.2) as a base. The asset provided propagation functionality and basic necessary data types. An overview of the flow of the WFC algorithm used is as follows:

- The space is initialised with each slot containing a superposition of all modules.
- If desired, selected slots can now be manually collapsed to certain modules.
- From here on, the algorithm will select the slot with the lowest entropy and collapse it, causing a wave of propagation events through neighbouring slots.
    - Each event will check and remove modules which have violating constraints from their respective slots. Each removal will cause an event to propagate to neighbouring slots.
- When all the slots have been collapsed, each slot has exactly one possible module left and no constraints are currently violated, the algorithm has finished and the resulting space (a graph of slots or modules) is returned. An illustration of a finished WFC algorithm is shown on Figure 13.
    - If there are any slots left with no possible modules or there are any constraints violated, the algorithm finishes but nothing is returned.

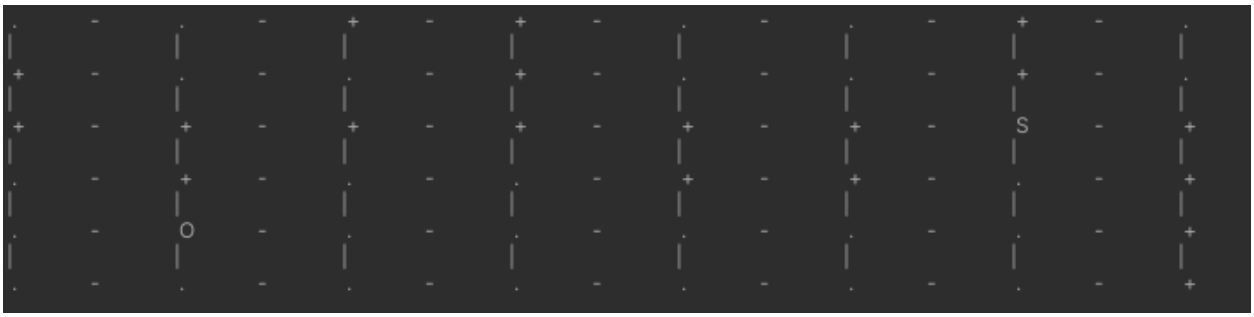

Figure 13. An example of a space after a completed WFC algorithm with dashes added between modules.

Since some rooms should only have one connection and the dungeon should be able to be represented as a spanning tree, the algorithm should not create a scenario where the one-connection rooms isolate some part of the graph. To combat this and to guarantee the generation of a spanning tree compatible graph, *Pikseon* uses a custom version of WFC.

The algorithm differs from a standard WFC implementation in the following ways:

- When a slot collapses to any module that is not an Empty Space module, a breadth-first[31] search is conducted through the space to find whether the collapsed slot is connected to the slot containing the Start Room module. The breadth-first search ignores all slots which have collapsed to an empty room **and** all slots which have collapsed to a room which should only have one connection. If there is no connection between the collapsed slot and the start possible, the algorithm will generate an error, which will force a re-generation of the space.
- When it has been made certain that there is a path to the Start Room, all the possible paths from the collapsed slot to the Start Room slot are found and then sorted to be ascending in length. This resulting shortest path is then traversed and the empty room module is removed from all of the slots in this path. This is in order to guarantee the connection of the collapsed slot to the rest of the dungeon.

The WFC of *Pikseon* makes use of two constraints which are also configurable from the Unity editor, as shown on figure 14. These constraints are the Only One Constraint and the Adjacency Constraint.

[31] https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

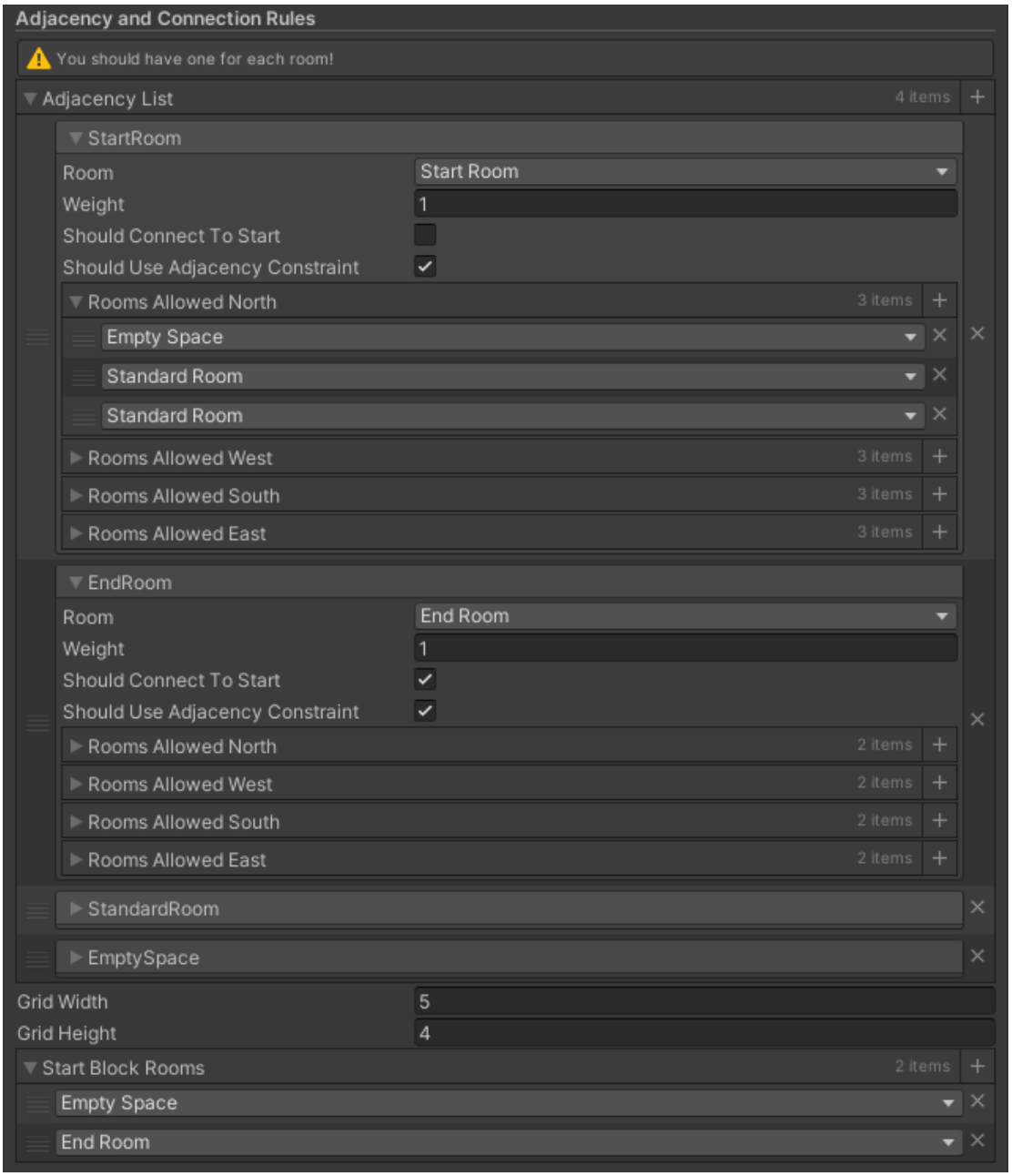

Figure 14. Screenshot of the Unity Editor Inspector window, showing the adjustability of WFC constraint rules.

The **Only One Constraint** is meant to block certain dungeon rooms from generating more than once. The constraint is checked at all times when the WFC propagation event happens. In effect, when a slot is collapsed into a room that should only appear once, the removal of the module corresponding to the room propagates through the space, removing it from all the superpositions where it is present.

Adjacency Constraints are for checking whether any given module can exist in the slot next to the slots that are adjacent to it. In this case, adjacency means that the slot in question is connected via an edge of the graph to its adjacent slot. Because the WFC in this project is essentially a grid, a slot can have at most four adjacent slots, one for each direction.

Each adjacency constraint is created with two parameters: the positional delta and a set of modules called *NeedsOneOf*. When a module (the source module) is tested for a slot (called the source slot), for each adjacency constraint attached to the module, each slot (called the target slot) will be checked. If the positional delta between the source slot and the target slot equals the delta for the current adjacency constraint, the constraint will check whether any of the modules in the superposition of the target slot exist within the *NeedsOneOf* module set. If none are found, the source module is removed from the superposition of the source slot.

4.3.2 Post-Processing the Graph

In order to make any use of the graph of slots which is returned by the WFC algorithm, some post-processing is needed. For proper functioning of the WFC, all the slots need to have a final module they collapse to. But since the resulting dungeon shouldn't be a grid, Empty Space room modules exist. In figure 15, the dots ('.') represent empty spaces.

These empty spaces, and all of their edges, should be removed in order to use the graph for generating a dungeon. This is achieved by iteratively looping through all of the slots and removing those which contain the empty space module. A similar approach is taken with the edges, all of the edges which connect to an empty space are removed. This is shown on figure 15.

Figure 15. Post-WFC Graph structure with all Empty Space rooms removed with their edges.

Since the rooms which should only have one edge are not a part of any path from any standard room to the start, it is possible to remove all of their edges without breaking any path to the start. The algorithm loops through all of the rooms which should only have one connection and removes all of their edges. This is needed for the spanning tree generation, since it could cause there to be a path leading through these rooms. By removing all of the edges, it can be guaranteed that the spanning tree generation does not create any such paths.

Because the dungeon should be without any loops, the graph has to be a spanning tree. For this, the graph is traversed breadth-first from the starting room, visiting each slot only once. After this is finished, each room which should only have one connection is iterated through and a random edge which connects to another room is chosen - that edge will be re-created. The end result is a graph of a dungeon, ready to be generated, shown on figure 16. The resulting graph is ready for the next step.

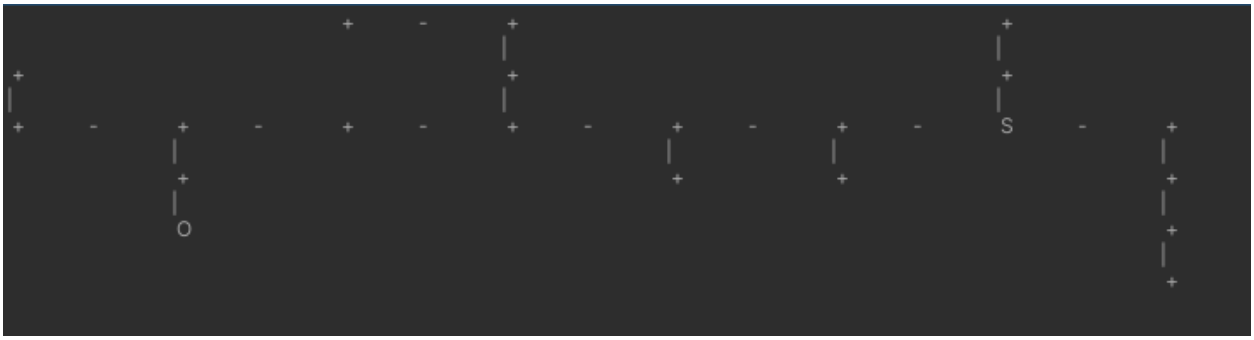

Figure 16. Post-WFC Graph structure after rebuilding it as a spanning tree and re-adding edges to rooms with one connection.

4.3.3 Generating a 3D World

When a graph structure without loops and with no empty rooms has been made, the next step is to generate a 3D world from the data structure. Since the design required allowing rooms of varying sizes, there was no option to take the easy route - the problem could not be solved by generating a grid-based dungeon with fixed coordinates. Instead, a system was created for a progressive, breadth-first procedural generation based on the input graph. An illustration of how the dungeon can handle rooms with various sizes and corridor placements is shown on figure 17.

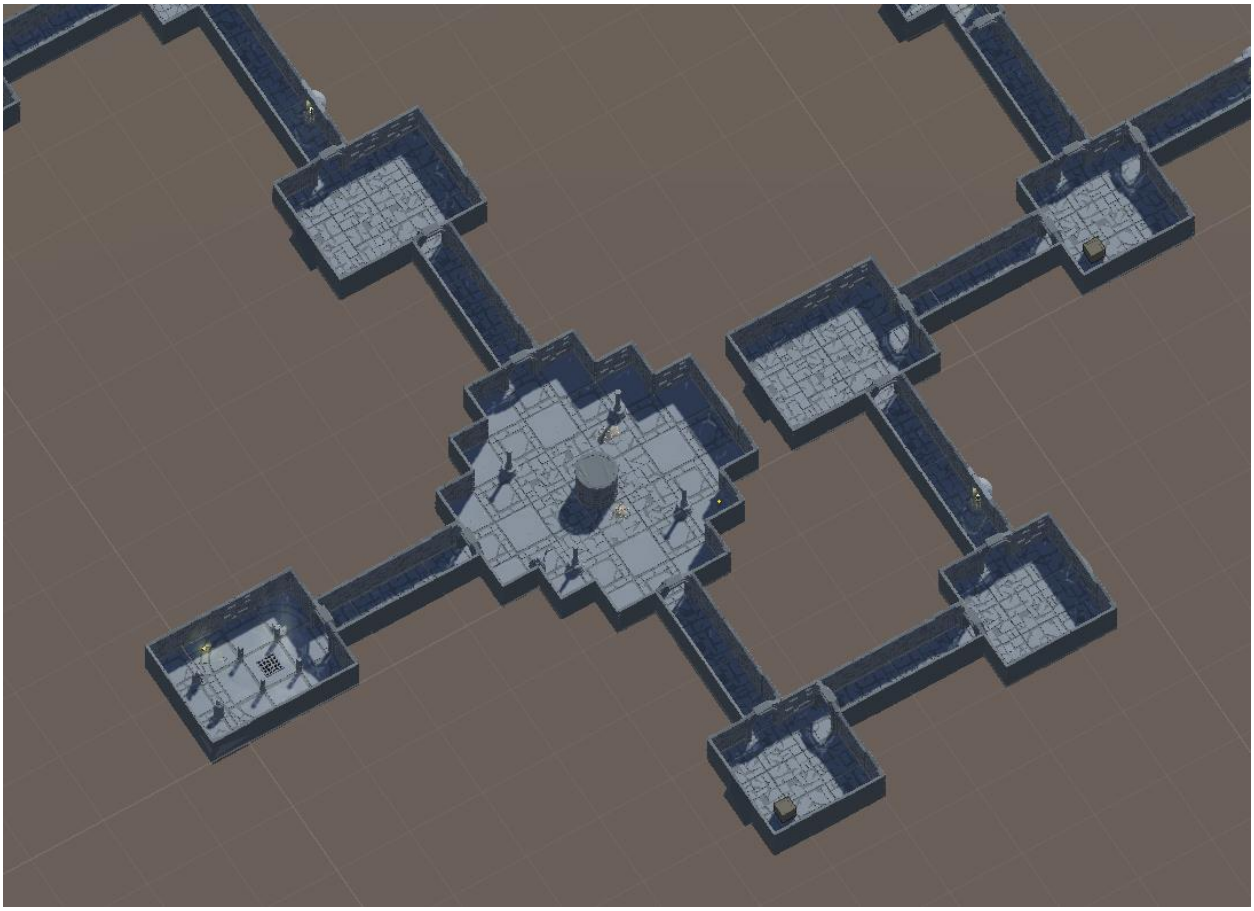

Figure 17. Illustration of how the dungeon rooms are not aligned to a fixed grid after generation.

In order to achieve this, a Unity Scriptable Object[32] was created as a configurable option set based on which the dungeon will be generated. It contains a List of dungeon rooms, categorised by their room type, and a List of corridors. The rooms and corridors themselves contain Unity prefabs[33] of pre-made rooms and corridors respectively. The model to generate will be randomly picked from a list of prefabs within the specific room object.

The input graph is iterated through in a Queue-based breadth-first method. An overview of the method is as follows:

1. Instantiate the first room, the start room, prefab at world zero. For each of its edges in the graph, open the door and instantiate a corridor with the direction of the edge at the corresponding point on the room prefab. Enqueue each of the edges' target rooms to be visited.
2. Dequeue the first room on the queue and instantiate its prefab. The prefab's location will be calculated based on the following info: the direction where the source edge was facing, the world position of the edge's connection point and the local position difference of the instantiated prefab's corresponding connection point.
3. Add this room's slot to the List of slots which have been visited.
4. For each of the room's edges, check if that edge leads to a slot that has already been visited. If not, generate a corridor in that direction and add the slot to the queue.
5. If the queue is not empty then go back to step 2, otherwise the algorithm is finished.

When the algorithm has finished, the player prefab will be instantiated at the spawn location of the starting room. For each room that should spawn hostile enemy entities, a random enemy amount will be picked from a predetermined range. That many enemies will be instantiated in the corresponding room, with a random spawn location preset picked for each enemy entity.

Although yet unimplemented, there is support to add another stage for generation before entity instantiation and after the main generation algorithm. The stage would be randomly generating scatter objects and extra features to rooms based on some defined set of possibilities.

[32] https://docs.unity3d.com/Manual/class-ScriptableObject.html
[33] https://docs.unity3d.com/Manual/Prefabs.html

5 Testing

Playtesting is an important part of game development, as it can help find bugs and usability issues.[34] According to Jakob Nielsen [19], testing with 3 to 5 players is enough to uncover the majority of usability issues and one should not test with over 5 people, due to harsh diminishing returns.

For this thesis, *Pikseon* was tested in-person with five participants and the testing sessions were recorded. After testing, a questionnaire was sent to testers. The results of the testing and the questionnaire were analysed. Based on the testing feedback, the discovered bugs are laid out. Also mentioned are future plans for improvements.

5.1 Methods

In line with Nielsen's study [19], it was determined that testing usability with five testers is sufficient for this thesis. The testing sessions were conducted in the Computer Graphics and Virtual Reality Lab[35] room in the University of Tartu Delta Centre. For the testing, a computer was set up with screen recording software and the game ready to be launched. Each participant waited their turn and then was asked to enter the room and play the game. They would receive no helping tips from outside of the game itself, as that would compromise the usability testing, but they were told that they can comment on their experience if they want to. The testing concluded when they either reached the end of the game or they were stuck somewhere for a long time.

[34] https://www.linkedin.com/advice/3/why-playtesting-crucial-successful-game-design-nqqme
[35] https://cgvr.cs.ut.ee/

Each participant gave their permission for them and their gameplay to be recorded. The recording was conducted using a Trust[36] webcam and an open-source recording software, OBS[37]. The recording started moments before launching the game and ended when the testing was over. During the testing, the participants were observed and notes were taken on any bugs and usability problems they encountered. The testing setup is shown on figure 18.

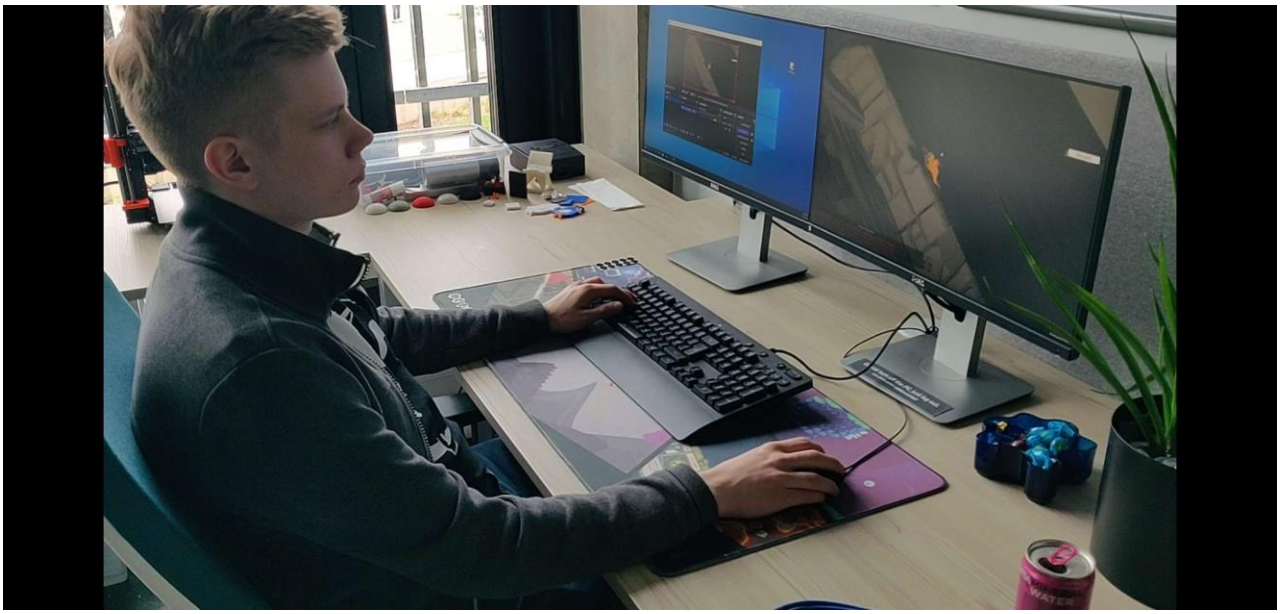

Figure 18. Usability testing participant testing the game.

After testing, a questionnaire was sent to all of the participants. The questionnaire is available as described in Appendix II. The questionnaire focused on how the participants felt about different aspects of the game and how they believed it could be improved.

5.2 Results

All of the testing sessions were conducted on 03.05.2024. During observation of participants during testing and when reviewing the recorded material, a number of technical and usability issues were uncovered. These issues are detailed in Table 2. Usability issues are coded with ‘U’ and technical issues with ‘T’.

[36] https://www.trust.com/en
[37] https://obsproject.com/

Table 2. Issues found during the usability testing procedure.

Issue UID	**Description**	**Priority**
U1	There is no feedback to the player about their current health status	High
T1	The walls do not go see-through as they should when the player is behind them, thus impeding the player's vision at certain positions.	Low
U2	The dungeon is a bit large for the current amount of content. It takes a lot of time to go from one place to another.	Medium
T2	Sometimes, when reloading the level, the player character's speed is lower for an unknown reason.	High
T3	When the level is loaded, the generated dungeon sometimes contains either blocked doors or no other rooms other than the Start Room. This is a serious issue, the root of which has not been found yet.	Very High
U3	There is no button to exit the game.	High
T4	Sometimes the light sources in the game flicker. This has been diagnosed as the point light limit being too low in Unity's project settings.	Low
U4	The glowing object in one of the randomly generated standard rooms attracted a lot of attention and had the appearance of being interactable, but no interactability was in fact present.	Medium
T5	Rarely, the player character is duplicated. Reasons are unknown and no known way to reproduce the problem.	High
U5	Feedback from the questionnaire. The movement of the character felt too sluggish and the animations were not appeasing.	High
U6	Feedback from the questionnaire. The game did not have audio cues for effects, they would have enhanced the feedback of actions taken during combat for example	High

Out of these issues, the only one with a priority of 'Very High' is T3, which describes problems of procedural generation. The exact reason for these problems has not yet been determined, nor has an exact way to reproduce them. That said, the only time when these issues have appeared has been when the level has been reloaded from inside of the game. That points towards the problem lying somewhere within the method of reloading the level.

Another remarkable issue is T5, which appeared multiple times during testing, although no certain cause is known. When this happened, the player character seemed to be completely duplicated, with both entities reacting to user input in exactly the same way. No way to reproduce this problem has been discovered yet. Sometimes, this issue appeared together with T2.

Based on the results of the questionnaire, as shown on figure 19, more than half of the participants thought that *Pikseon* belongs in the Dungeon Crawler genre, with others mentioning Roguelike. This means that the game indeed falls under the genres it was intended for.

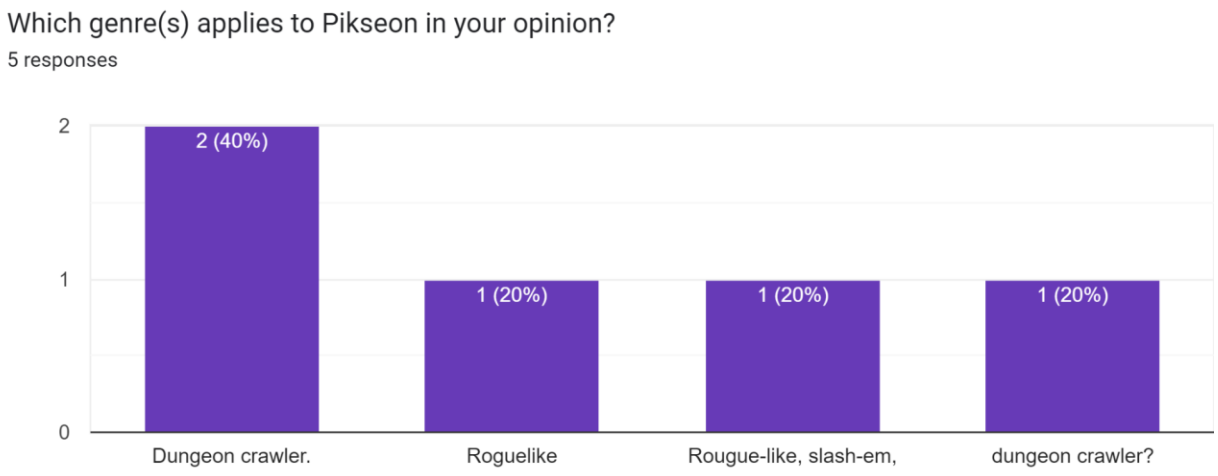

Figure 19. Responses to the post-testing questionnaire on the topic of genres.

The participants also felt that the generation method was mostly appropriate (4 out of 5 responses said 5 out 5 for the question “Did you feel that the dungeon generation was appropriate for a roguelike dungeon crawler?”). This is a good result, as it shows that the selected algorithm for dungeon generation is perceived well by players. They also felt that the perceived randomness of the generation was moderate and could be improved. This can be solved by adding more variation to the generated rooms and more different types of rooms.

5.3 Future Improvements

This subchapter provides an overview of bugs that were known before testing, bugs that were found during testing and further development ideas for *Pikseon*. The first course of action is working through fixing all of the usability and technical issues found during testing. After the issues have been fixed, a number of ideas for future improvements are also present.

As a loosely priority-ordered list, the improvement plans are:

- Fix the issues discovered during the usability testing.
- Develop another stage to the procedural generation for randomly placing scatter in the generated rooms. This should also support interactable objects, for either flair or content.
- Add more variety for rooms, more sizes and designs for each room.
- Develop new types of rooms, for example: Boss Room, Mini-Boss Room, Loot Room, Shop Room.
- Develop an inventory system and functionality for objects that are able to be picked up, upon which they should be added to the user's inventory.
- Develop new types of weapons and enemies.
- Add support for multiple different stages of levels and also a central hub system for navigating between them.
- Add a map of the dungeon to the interface.
- Fix any bugs that may have appeared and release the game on itch.io.

The description for these ideas is intentionally vague, as they have not yet been implemented.

6 Conclusion

The games industry as a whole and also roguelike games in particular have seen growth in recent years. This thesis covered the development of a roguelike dungeon crawler game called *Pikseon*. Other roguelikes like *Vampire Survivors, BPM: BULLETS PER MINUTE* and *Curse of the Dead Gods* were analysed. The design and development of *Pikseon* heavily focused on the aspect of procedural generation, for which an implementation of Wave Function Collapse was used. The initial plans for the game were more ambitious, which had to be dialled down in order to finish the project in time.

The design of *Pikseon* focused on independent gameplay loops on closed, randomly generated dungeon levels. The dungeons consisted of multiple different rooms, the placement of which was randomly generated. For achieving a unique gameplay element that set *Pikseon* apart from competitors, a healthiness system based on per-object pixelization of game entities was developed. The game was developed using the Unity game engine, with a number of third-party assets. The procedural generation based on a custom Wave Function Collapse algorithm involved multiple steps that were needed to produce a viable data structure for generating the three-dimensional dungeon in the game world.

The game was tested for usability near the end of the development cycle with five participants. The purpose of the testing was to find usability and technical issues present within the game. In total five technical and six usability problems were identified from the testing. Improvement plans for the future based on initial goals and feedback from the testing were laid out.

I would like to thank all testers who participated in testing *Pikseon* and for providing feedback with which the game could be improved. I would also like to thank the Computer Graphics and Virtual Reality lab of the University of Tartu, who provided the room and equipment necessary for the testing. Special thanks to my supervisor Daniel Nael, who gave guidance throughout the process and helped with narrowing the scope at the right time; and to Raimond-Hendrik Tunnel, who gave a lot of valuable feedback during the writing of the thesis and gave useful seminar presentations for writing the thesis.

References

[1] statista, “Video Games - Worldwide.” Accessed: Feb. 14, 2024. [Online]. Available: https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide#revenue

[2] E. Van Allen, “Roguelike vs. Roguelite: What’s the difference between the two?,” Destructoid. Accessed: May 07, 2024. [Online]. Available: https://www.destructoid.com/the-difference-between-roguelike-and-roguelite-games/

[3] J. H. Lee, R. I. Clarke, and A. Perti, “Empirical evaluation of metadata for video games and interactive media,” *J Assoc Inf Sci Technol*, vol. 66, no. 12, pp. 2609–2625, Dec. 2015, doi: 10.1002/asi.23357.

[4] Mark J. P. Wolf, “The Medium of the Video Game,” University of Texas Press, 2002, pp. 113–134. doi: 10.7560/791480.

[5] R. I. Clarke, J. H. Lee, and N. Clark, “Why Video Game Genres Fail,” *Games Cult*, vol. 12, no. 5, pp. 445–465, Jul. 2017, doi: 10.1177/1555412015591900.

[6] A. A. Garzia, *Roguelike Development with JavaScript*. Berkeley, CA: Apress, 2020. doi: 10.1007/978-1-4842-6059-3.

[7] K. Stuart, “Dungeon crawler or looter shooter? Nine video game genres explained,” The Guardian. Accessed: May 11, 2024. [Online]. Available: Dungeon crawler or looter shooter? Nine video game genres explained

[8] Jesse Schell, *The Art of Game Design: A Book of Lenses*. 2008.

[9] Elliot Bentine, “ProPixelizer.” Accessed: Mar. 05, 2024. [Online]. Available: https://sites.google.com/view/propixelizer/home

[10] G. Smith, M. Cha, and J. Whitehead, “A framework for analysis of 2D platformer levels,” in *Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, New York, NY, USA: ACM, Aug. 2008, pp. 75–80. doi: 10.1145/1401843.1401858.

[11] M. Salmond, *Video game level design: how to create video games with emotion, interaction and engagement*. London: Bloomsbury Publishing, 2021.

[12] S. Rogers, *Level Up! The Guide to Great Video Game Design*, 2nd Edition. Wiley, 2014.

[13] A. Crivello, "Game Engines," G2. Accessed: Apr. 30, 2024. [Online]. Available: https://www.g2.com/glossary/game-engines-definition

[14] SlashData Team, "Did you know that 60% of game developers use game engines?" Accessed: Apr. 30, 2024. [Online]. Available: https://www.slashdata.co/post/did-you-know-that-60-of-game-developers-use-game-engines

[15] Udemy Team, "What is Unity Game Engine?" Accessed: Apr. 30, 2024. [Online]. Available: https://blog.udemy.com/unity-game-engine/

[16] D. Phuc, "What is Tweening Animations? - An Animator's Ultimate Guide." Accessed: Apr. 18, 2024. [Online]. Available: https://animost.com/tutorials/tweening-animations/

[17] M. Gumin, "WaveFunctionCollapse." Accessed: Apr. 14, 2024. [Online]. Available: https://github.com/mxgmn/WaveFunctionCollapse/blob/master/README.md

[18] I. Karth and A. M. Smith, "WaveFunctionCollapse is constraint solving in the wild," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, New York, NY, USA: ACM, Aug. 2017, pp. 1–10. doi: 10.1145/3102071.3110566.

[19] J. Nielsen, "Why You Only Need to Test with 5 Users," Nielsen Norman Group. Accessed: Apr. 30, 2024. [Online]. Available: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Appendix

I. Glossary

Rhythm-based – Category of games where the actions of the user usually need to be timed according to an audible rhythm.

FPS – First Person Shooter – Category of games where the user's viewpoint is the same as the controlled character and includes heavy elements of guns and shooting.

Dungeon Crawler – Category of games which usually involve the player exploring through grid-like dungeons. See Chapter 2.1 for more information.

Roguelike – Category of games which share major design elements with Rogue. See Chapter 2.1 for more information.

Roguelite – Category of games which have one or more significant deviations from standard roguelikes but still share many aspects of them. See Chapter 2.1 for more information.

Game Engine – Software frameworks containing many useful tools for developing games.

WFC – Wave Function Collapse - Procedural generation algorithm inspired from quantum mechanics.

UID – Unique Identifier

Game jam – Event where games are developed in an intense and short time period.

ASCII – American Standard Code for Information Interchange - Common encoding format for digital text data[38]. Contains unique values for 128 different characters.

Spanning tree – Subset of a graph where all vertices are connected without any cycles[39].

Breadth-first – Graph traversal algorithm where nodes are visited level-by-level.

[38] https://www.techtarget.com/whatis/definition/ASCII-American-Standard-Code-for-Information-Interchange
[39] https://www.geeksforgeeks.org/spanning-tree/

Tweening – Filling the in-betweens of animation keyframes. In the context of this thesis, that usually means animating the transition of an object from location A to location B.

Superposition – In quantum mechanics, the ability to be in multiple states at the same time until measured.

II. Accompanying Files

The files accompanying the thesis are organized into the following files and folders:

- /Testing_Videos – This folder contains the videos of the usability testing sessions.
- /Source_Code – This folder contains the Unity project and source code with everything able to be freely shared.
- /Testing_Questionnaire.pdf – This file contains the questionnaire sent to the testers after the usability testing.
- /Testing_Questionnaire_Responses.xlsx – This spreadsheet contains the responses to the questionnaire.
- /Game_Binaries – This folder contains the compiled game, instructions for launching can be found in appendix III.

III. Game Launch Instructions

The instructions for opening the game are as follows:

- Navigate to the Game_Binaries folder in the files accompanying the thesis.
- Run *Pikseon.exe*.
- Firewall pop-ups can be ignored, as the game does not require network connectivity.
- The game will now open. The game can be closed at any time by pressing the Escape key.

IV. Source Code

The source code containing everything developed by the author and those assets where the redistribution is not prohibited by their respective licence is available with the thesis, as detailed in Appendix II.

For those wishing to access the whole source code, please send an email to johannes.langsepp@gmail.com with your request and an explanation for why you want to access the project.

V. Licence

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Johannes Voldemar Langsepp

grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis *Pikseon* – Procedural Dungeon Crawler, supervised by Daniel Nael, MSc.

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Johannes Voldemar Langsepp

20/05/2024