BACHELORARBEIT

GAME PROTOTYPE: ONLINE PROCEDURAL MAP GENERATION SUPPORTING MULTI LEVEL BARRIERS

Verfasserin ODER Verfasser
Lukas Höwarth

angestrebter akademischer Grad
Bachelor of Science (BSc)

Wien, 2020

Studienkennzahl lt. Studienblatt:	A 521 [4]
Fachrichtung:	Informatik - Medieninformatik
Betreuerin / Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. Helmut Hlavacs

Contents

List of Figures

Abstract

This thesis shows the concept and implementation process of a video game using different procedural content generation concepts. The content is generated using an online approach. The generated map supports multiple types of barriers. Multi-level barriers are supported while keeping every room accessible. Additionally, gameplay elements are shown supporting dynamic difficulty adaption. The presented algorithms show the benefits of using an online approach for Procedural Dungeon Generation (PDG). Everything combined to created a fully playable game presenting the interactions of these components.

1 Introduction

1.1 Motivation

Developing Video Games has become more accessible than ever. Small game studios, so-called Indie game developers, create and publish games nowadays. These studios consisting of only a few people or even only one individual. Nonetheless developing a game takes many resources, much effort, and is time-consuming. Procedural Content Generation (PCG), see 2.1, is widely used to streamline different aspects of the development cycle. This relieves the development team from handcrafting every single aspect of a project. Not only does it speed up the development process but it also enables vaster worlds and in general more diverse and varied content. So to say, more content in general. One major aspect PCG is used for is the creation of game maps and even whole worlds populated with generated content. Dungeon Crawlers, more specifically rogue-likes are a popular genre that heavily uses PCG generated content. To offer more diversity and replay-ability while keeping the game interesting and engaging, by endlessly generating different dungeon layouts.
Procedural content is either generated offline, see 2.2, or online, see 2.2. The usage of the first approach is more common among those two. If used for map generation most offline approaches need an additional search algorithm to guarantee playability throughout the map. One topic that few methods include in their generation is the support of barriers, see 2.4, and more so multilevel barriers. On the one hand, this adds extra complexity to the map creation process but on the other hand, offers new gameplay possibilities to integrate into the gameplay.

1.2 Objectives of the thesis

The goal of this project is to combine multiple Procedural Content Generation concepts to design and implement a new dungeon map generation algorithm. For the map, an online approach with multi-level barrier support is combined with dynamic difficulty adaption, see 2.4, dependent on player progression and performance. To showcase the implementation of these concepts a playable prototype was created. This combination was chosen to show how using an online approach can help to get rid of using additional algorithms to guarantee play-ability by keeping track of generation parameters during run-time even when using barriers. Furthermore, it is shown how using the online approach can improve difficulty adaption by adapting it during gameplay in short intervals.

2 Literature and Terminology

Considering Literature, the base work of the research was done using multiple papers exploring the current landscape and standards used for Procedural Content Generation, especially focusing on Procedural Dungeon Generation. The referenced papers used as a guideline for this thesis.
A brief overview of the PCG landscape is given in [27]. Multiple papers of the past years are analyzed according to the solution presented for certain PCG problems. Findings are recorded and features and approaches listed that are widely used as well as hardly explored topics and combinations of different techniques. An overview and analysis are given what many works Incorporated and especially which research topics have been explored little. This was the first paper taken as a reference and in the further process used to narrow down this thesis topic.
One of the best resources for PCG in general, providing fundamentals, terminology, and presenting different approaches regarding this topic is provided in [20]. It gives definitions for the most commonly used terms using different approaches. It also explains many PCG algorithms and discusses game design at a base level. Different use cases for PCG are discussed by breaking game mechanics and assets used for games down to their smallest components and analyzing their use case and functionality.

2.1 Procedural Content Generation (PCG)

PCG can be defined as the algorithmic creation of game content with limited or indirect user input[25]. The game content can be created with or without the help of humans using algorithms. The definition is further explained by giving a separate definition for "content", "procedural" and "generation". Content is said to be what is contained in the game, from maps, levels, and textures to story, quest, and music. "Generation" and "Procedural" imply that algorithms and procedures are used to generate things.
One of the main reasons for using PCG is that it saves on resources. When developing a game without using any PCG methods every single step has to be done by a human. Developers, artists, level designers, all those aspects and many more are expected to be included in games nowadays. The current goal is not to take the work away from those people but to support them in creating PCG that helps streamline processes and enable smaller studios with less budget to create content-rich games.
Further ideas are games that never end. Using PCG, in theory, an endless amount of diverse content can be generated giving the player an endless amount of replay value. Combining this with the possibilities to adapt content to players' preferences can maximize the effect of serious games or tailor the addictive nature of casual games.
Designing new PCG methods leads to a better and deeper understanding of the process of the game content. This enables the developer to create and design more diverse content manually. This leads to more sophisticated PCG

algorithms and so on [20].

2.2 Terminology

These terminologies were taken from [20]

- PCG System
  A system that incorporates different PCG methods in its implementation.
- Online Generation
  Elements of the game are generated during gameplay. This opens up ways for content that adapts to the player.
- Offline Generation
  Content is generated before a level starts. This is useful when creating complex parts like environments. Furthermore adaption to the player can be achieved between levels by generating behaviors depending on the last play session.
- Necessary Generation
  Everything generated that is needed to complete a level or the game. This for example includes game maps that have to be playable from beginning to finish.
- Optional Generation
  All the thing generated that are not necessarily needed to complete the game. This content is interchangeable with other content and would not hinder the player progression if the content is not correct.
- Degree of dimensions of control
  Either by using certain seeds to recreate the creation process or by providing changeable parameters to influence the generation process.
- Generic Generation
  The content is created without taking players' actions and behavior into account.
- Adaptive Generation
  Takes player behavior into account and adapts certain game elements on the fly like the difficulty or pacing.
- Stochastic Approach
  The generation of the same content twice is not possible.
- Deterministic Approach
  Given the used parameters and starting point the content can be generated again.
- Constructive Approach
  The content is generated once. Further modifications of the generated content are not permitted.

- Generate-and-test Approach
  Go through the generation loop multiple times until a satisfactory solution is generated.

- Automatic generation
  The content is generated automatically with minor input from the developer. Some smaller parameters e.g. room size can be changed.

- Mixed Authorship
  The generation algorithm and the game designer work hand in hand. They complete each other works e.g. the designer starts drawing a map and the algorithm finishes it.

2.3 The PCG Landscape

[27] analyzes the current state of this topic. The data was manufactured by surveying 31 papers published in the last eight years focusing on dungeon and maze-like level design. By using taxonomy from [13] different approaches are classified. In addition to the definitions given in Terminology, "Dungeon", "Game Features and Mechanics" and "Solution Strategies" are defined.
The definition of "Dungeon" is split into three categories, later on, it is suggested, that papers working on procedural dungeon generation should define the level structures to make classification and use cases clear and easier to categorize. These three are:

- TDML (Top-down-mansion-likes)
  For example, the original "The Legend of Zelda" NES game dungeon was designed in that manner. Rooms of fixed sizes interconnected by doors. Figure 2.1

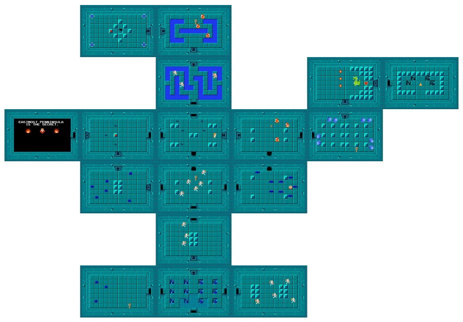


Figure 2.1: The Legend Of Zelda Map TDML Map
Source: [28]

- TDCL (Top-down-cave-likes)
  Caverns don't present a clear structure but have passageways and uneven

cave parts. Rooms are not separable from each other. An example is the ”Pokémon Mystery Dungeon” games. Figure 2.2

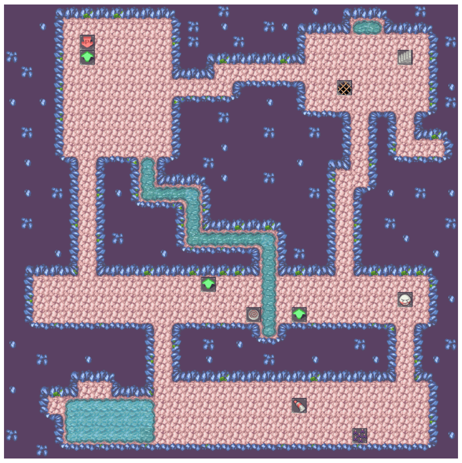


Figure 2.2: Pokemon Mystery Dungeon TDCL Map
Source: [8]

- SS (Side Scrollers)
  In general, any of the above types could be used for Side Scrollers. The distinction is present in the scenario structure aka the room content. One of the more recent games in this category gaining traction is ”Dead Cells”. Figure 2.3

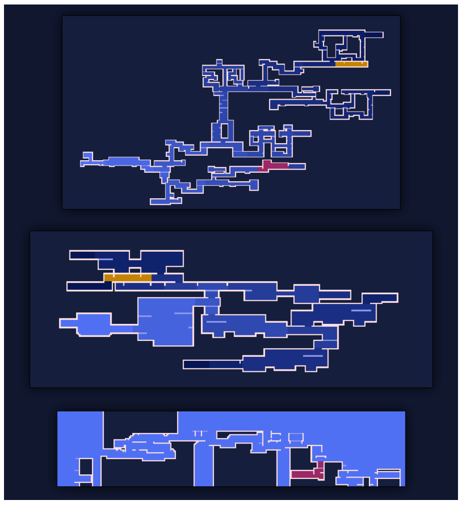


Figure 2.3: Dead Cells Side Scroller Map
Source: [26]

"Game Features and Mechanics" defines game elements most commonly used in association with procedurally generated dungeons and the rogue-like genre. The major problems found, which were tackled in this thesis are:

- Balancing of difficulty was not present in any paper
- Only a few papers taken barrier into consideration
- No paper used online generation

The conclusion that was taken from analyzing these peer-reviewed papers further supports the decision to combine the algorithm with a game on top. This narrows down the constrains and content used for the level creation. The importance of further researching barriers and presenting new solutions is highlighted especially in the case of top-down mansion-likes.

2.4 Online Generation, Barriers and Adaption

Online generation was chosen mainly because according to [27] and [13] generating dungeons online still is a largely unexplored field. Furthermore online generation and guaranteeing playability is a show-stopper according to [2]. The method created for this thesis ticks both marks. It is an online approach while still guaranteeing playability throughout.
[13] states that the nature of a dungeon might discourage an online approach because it works as a narrative progression towards a fixed goal. By taking a closer look at how two of the most popular games in the Dungeon Crawler/Rogue-Like Genre, "The Binding of Isaac" [15] and "Crypt of the Necrodancer"[4] are using PCG for map generation and other assets it paints a different picture. Map layouts are randomized and the focus is set on generating a fixed amount of certain room types per floor. In "The Binding of Isaac" for example, the boss room can be anywhere on the map. If encountered early during gameplay, it is encouraged to explore the rest of the dungeon to get stronger and have better odds in defeating the boss. This gameplay loop and map generation can also be realized using an online approach enabling other possibilities for adaption by generating on a room to room basis.
Barriers are discussed in very few papers. One common approach is to include multi-key barriers. These are barriers that all require the same kind of key to unlock. Therefore one just has to keep track of closed doors and spawn the same amount of keys. One key fits all. No paper described the combination of an online approach including barriers. Let alone different kinds of barriers, not only represented by anything other but locked doors are even rarer. For example, [19] describes a locking door mechanism but it uses an offline generation and presents only one type of barrier being locked doors.
Another big topic of PCG is adaption. Not only the player gets better at knowing the mechanics of the game but the game also gets better in understanding the abilities and behavior of the player. The player changes the approach to the

game depending on the wanted outcome examples would be to be more entertained or even to be challenged more. A game should be able to adapt to the way the player plays it in certain ways. The simplest form of adaption is DDA being "Dynamic difficulty adaption".
This is simply the game getting harder if the player is doing well and the game getting easier if the game is too hard and the player makes little progress. This kind of adaption was implemented when designing the game. It is taken into account during gameplay and adapts in a room to room basis. This represents one use case for the stats tracked during gameplay [21].

3 Technologies used for Implementation

The following section describes the used technologies and the main reasons they were chosen. Moreover considered alternatives are mentioned.

3.1 Godot

Godot was chosen for a multitude of reasons. During the planning phase, multiple engines were taken into consideration. After deciding to make the project a 2D game, Godot stood out with its rich capabilities in this regard. Godot, like many other engines, supports 2D and 3D, but in contrast to the other ones, it offers dedicated nodes, see 3.1, for each mode. In addition to that if working in 2D space, things are expressed in actual pixels. Whenever manipulating an object, pixels are used as actual units. This simplifies the creation and optimization process since everything can be scaled and resized without quality loss. Another big aspect is that it is completely open-source with a huge community supporting and improving it every day. The documentation for the whole engine is very good in quality and updated regularly. If something is not working or features are missing, one can simply extend the engine by adding modules. Modules allow additional functionality at any level without modifying the core codebase. The Interface of the engine is built as a GoDot project, this opens up the possibility to add functionality to the Engines Interface.
Regarding the creation of content, everything created with Godot is a Node. Nodes are the building blocks making up a project. Combining multiple nodes creates a Scene, see 3.1. Scenes can be saved as templates and instanced from other nodes at any given time. Adding nodes during run-time and the other functionalities discussed seemed to be convenient for a PCG project.
One small feature that turned out to be a game-changer, maybe due to personal preference, is the built-in IDE. Having everything in one place and seamlessly switching between GUI, editor and the game helped streamline the workflow.
Another reason Godot was picked, is that after trying out different engines on the hardware the game prototype was developed on, it was working the smoothest. Without any issues, from the get-go. The engine itself is super lightweight and by using the dedicated 2D mode for developing the overhead of loaded modules is kept low.

In Godot, Nodes are the base classes for all scene objects. Everything created is a node. Nodes can be children of other nodes. This results in a tree structure. This tree structure is called a scene tree. Scenes can be saved on disk and instanced in other scenes, therefore, streamlining the process of instancing objects of the same class. This concept was heavily used thought-out the whole project. Instancing pre-built scenes was very useful during the procedural generation process. [14].
A multitude of alternatives were considered:

- Unity [24]

Unity offers an extensive documentation and on top of that, due to its popularity, an enormous community and tons of courses and tutorials. Because of the high amount of studios using this engine, a lot of work is done to improving the engine. Furthermore it provides support for nearly all platforms available.

- Unreal Engine [10]
  Unreal Engine provides a high-end, real-time engine using state of the art rendering, physics and animations. Therefore being mostly used for 3D games and popular with Triple-A studios. The engines source code is accessible therefore being highly modifiable.

- Game Maker Studio [11]
  Game Maker Studio provides many of its features without the need to program. Nonetheless coding experience can be used to improve the experience. It offers an intuitive and simple interface, therfore making it quick and easy to prototype simple games.

These three engines were considered as alternatives. Mainly due to the open source nature, the great 2D support and that everything one creates is 100% the property of the creator, Godot was chosen over the rest.

3.2 Graphics and Audio

Tools to create assets were chosen with two criteria in mind. Firstly they had to be open source and the content created had to be free to use in the thesis scenario. Secondly, they had to be easy to use.
After trying a multitude of tools for creating animation and sprites, it became clear that many programs offered way too much functionality for this simple use case. Krita and Gimp for example have such complex interfaces, that learning the interface and features would have taken way too much time. The two options specifically designed for creating sprite art and sprite sheets were Piskel [5] and Aseprite. In the end, Piskel was chosen mainly due to it being free to use and offering all the basic tools needed for this project.
A multitude of alternatives were considered:

- Aseprite [3]
  A tool especially designed for creating pixel art. It is easy to use but still feature rich. One big adavtntage is that animating is supported from the get go.

- Krita [9]
  Krita is completely open source and mainly focuses on digital art, illustration and painting. It supports many different file formats to work with, without the need to convert them first. All in all it is lightweight feature rich and multi platform.

- Gimp [23]
  Gimp, an open source image editor and art tool. It supports integrating with many programming languages therefore being highly extendable and flexible. The main focus is image manipulation.

With the exception of a few sound effects, every asset used in this project was created from scratch. For about 90 percent of the sound effects, the tool Bfxr [7] was used. All sounds created with this tool are free to use in any way. With some trial and error, it is fairly simple to generate fitting sound effects. A few sounds were taken from OpenGameArt [1]. This website has tons of different content free to use under specific licenses, mostly crediting the author of the content.
At first, BeepBox was used to create music but this took way too much time and the learning curve to create something sounding borderline good was too steep. If there was more time it would have been used nonetheless due to its open-source nature
The used alternative is Soundation [22], used to create the main music loop of the game. It offers a free and a premium plan to create music. The free plan has fewer features, but no matter which plan is chosen the created music can be used royalty-free. This tool offers predefined music loops that can be mixed and matched using multiple audio channels and filters. Using this approach also took some time and effort but yielded higher quality results in a much shorter time frame.
A multitude of alternatives were considered:

- BeepBox [17]
  Using BeepBox one can create music loops by simply dragging and dropping notes. Created music patterns can be saved and used simultaneously. A simple interface coupled with the essential mechanics to create loops is provided. On top of that everything created is free to use.

- FlStudio [18]
  FlStudio offers a complete digital work station. It supports mixing, digital instruments and sound effects. On top of that a multitude of plugins are available to enhance the experience. Its a high end tool for all kinds of music creators.

- DanielX.net Paint Composer [16]
  DanielX.net Paint Composer provides easy to use tools combined with a playful interface. Inspired by video game sounds, music loops can be created by dragging and dropping different sound effects. It is an interesting tool to play around with and get a feel for the music creation process.

3.3 Version Control

Github was used as the version control tool. With every major addition and change to the code, a new commit was pushed including a fitting changelog.

When rebuilding the project or reconstructing the design process, having multiple versions representing different states the game was in, can be very insightful. Furthermore, it is a fail-safe during development [12].

4 System Architecture

This chapter gives a brief overview of the main game components and offers a deeper look at the main game processes using flow charts. For creating the diagrams [6] was used.

4.1 Implementation Goals and Architecture

The goal of the implementation is to create a playable prototype of a game combining multiple PCG techniques. The ones combined are online generation, adaption, and barriers. The online approach showcases a possibility to create multi-level barriers without the need of a search algorithm.
There is no final goal to the game, one plays until defeated or quitting the game. The game is playable until one of those events occurs. The map generation algorithm was created in such ways, that every room has to be accessible, therefore no dead ends and soft locks can occur. Depending on the initial game settings, whenever starting a new game, the map generation and difficulty change and adapt in different ways. The main influence for the adaption is the players' progression. using a multitude of tracked stats. The finished prototype includes all aspects of a modern video game including visuals and sound. This is to showcase the created algorithms in a natural habitat.

In Godot all components are nodes and scenes. This project consists of the game scene tree contain the whole project, a node for all globally used variables, a scene used for the title screen and a world scene tree containing the main game. When a new game is started, the title screen scene is unloaded and the main game scene is loaded. On game exit, the same thing happens vice versa. All nodes and scenes instanced during gameplay are added as children to the world scene tree.
In Figure 4.1 an overview of the general gameplay loop including generation, player actions and adaption is presented.

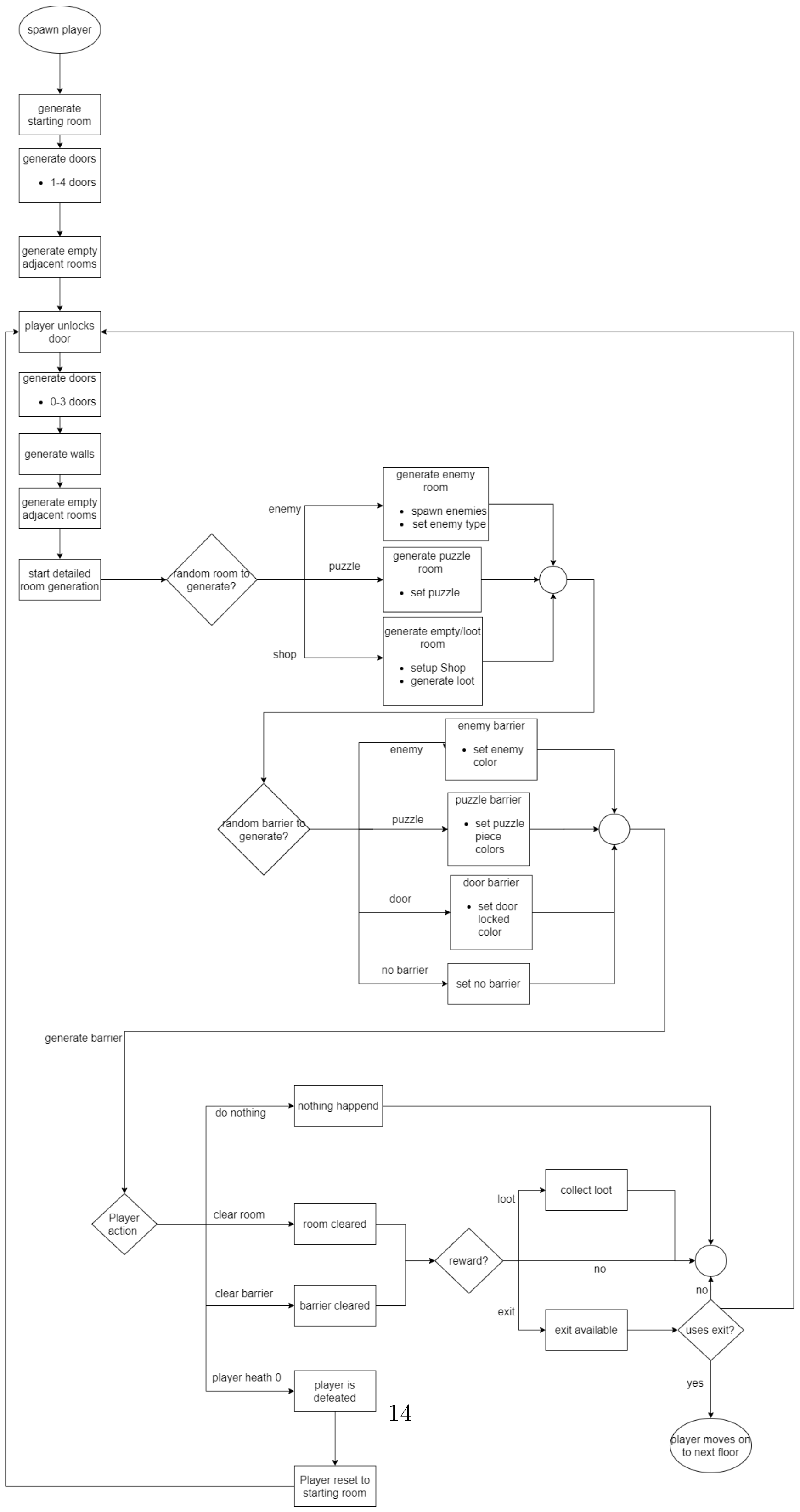


Figure 4.1: General Gameplay Loop

4.2 Map Generation

The map generation process in Figure 4.2 shows how a new room is generated and added to the map.

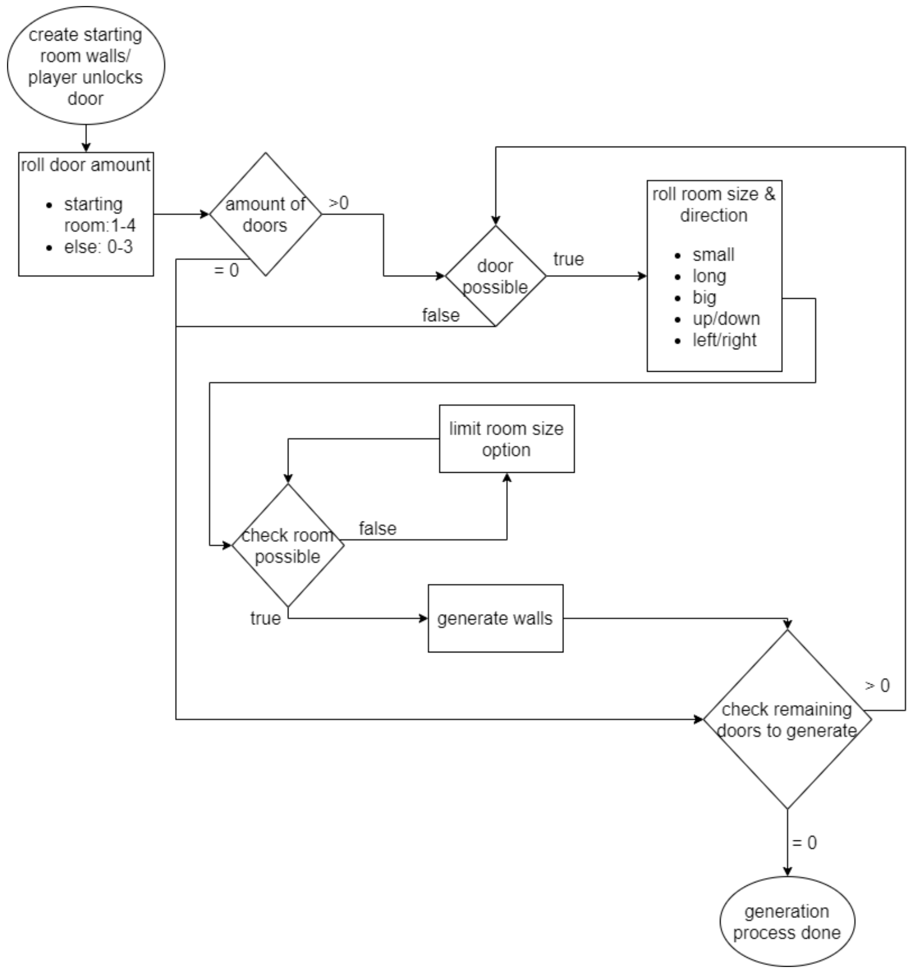


Figure 4.2: Map Generation Process

4.3 Barrier Generation

Figure 4.3 shows how it is decided what and how game objects can become a barrier during the generation process.

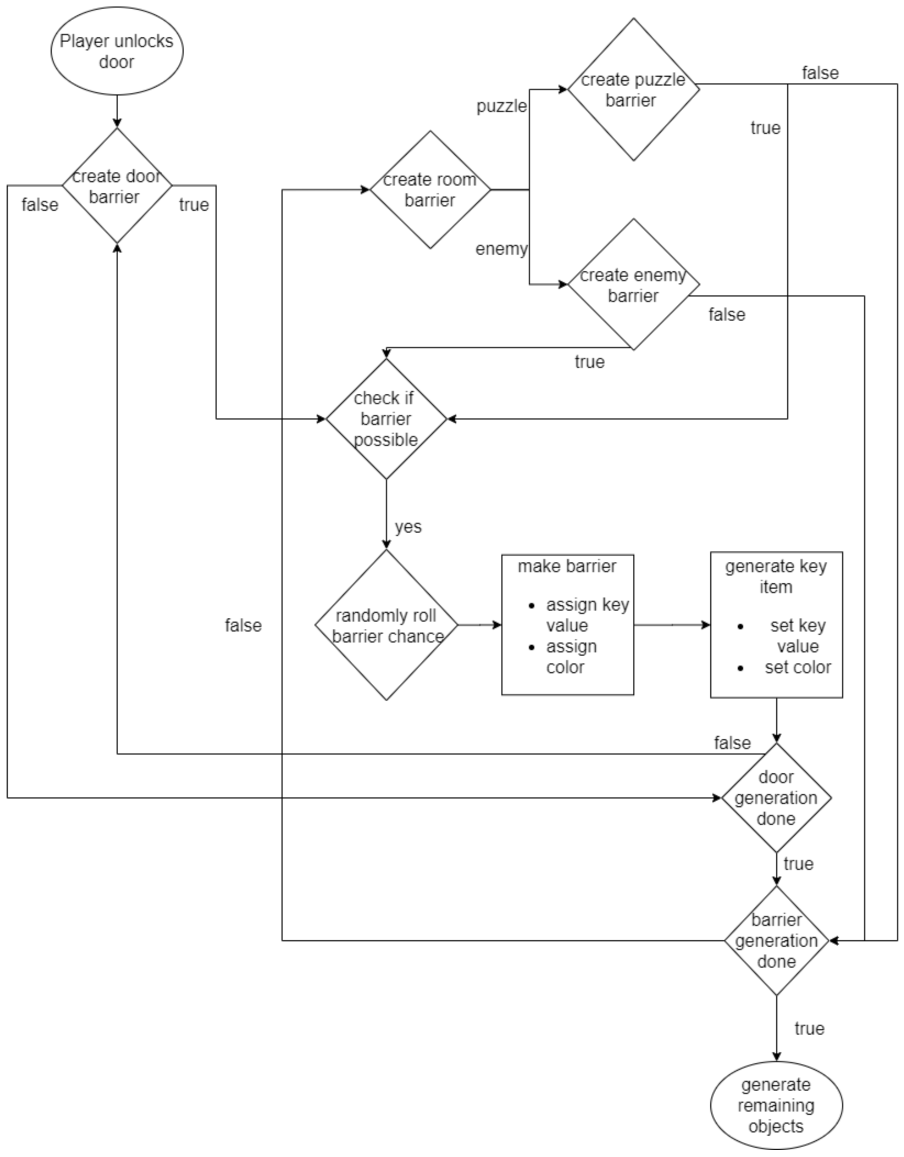


Figure 4.3: General Barrier Generation

4.4 Turn Based Design

A compact overview of the turn based design used in the prototype is given in Figure 4.4

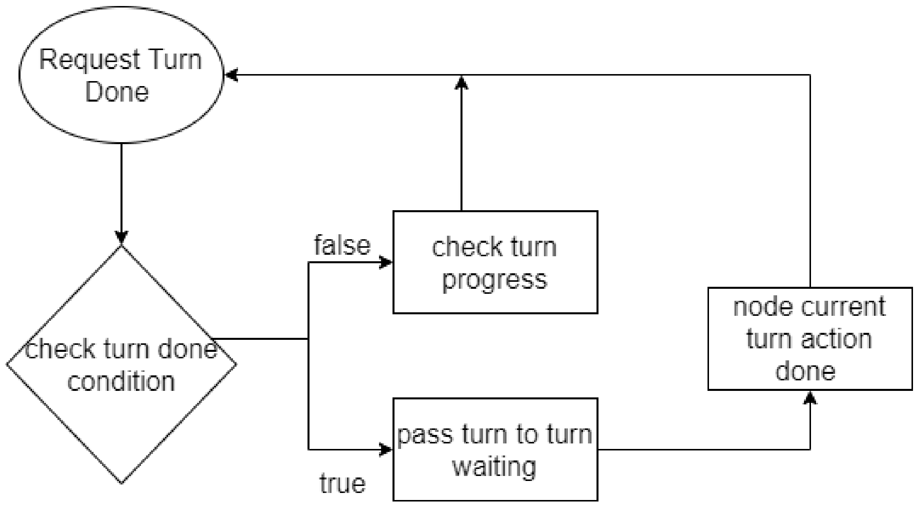

Request Turn Done
check turn done condition
false
true
check turn progress
pass turn to turn waiting
node current turn action done

Figure 4.4: Turn Controller Flow

4.5 Adaption

The implementation of the dynamic difficulty adaption used in the prototype is shown in Figure 4.5

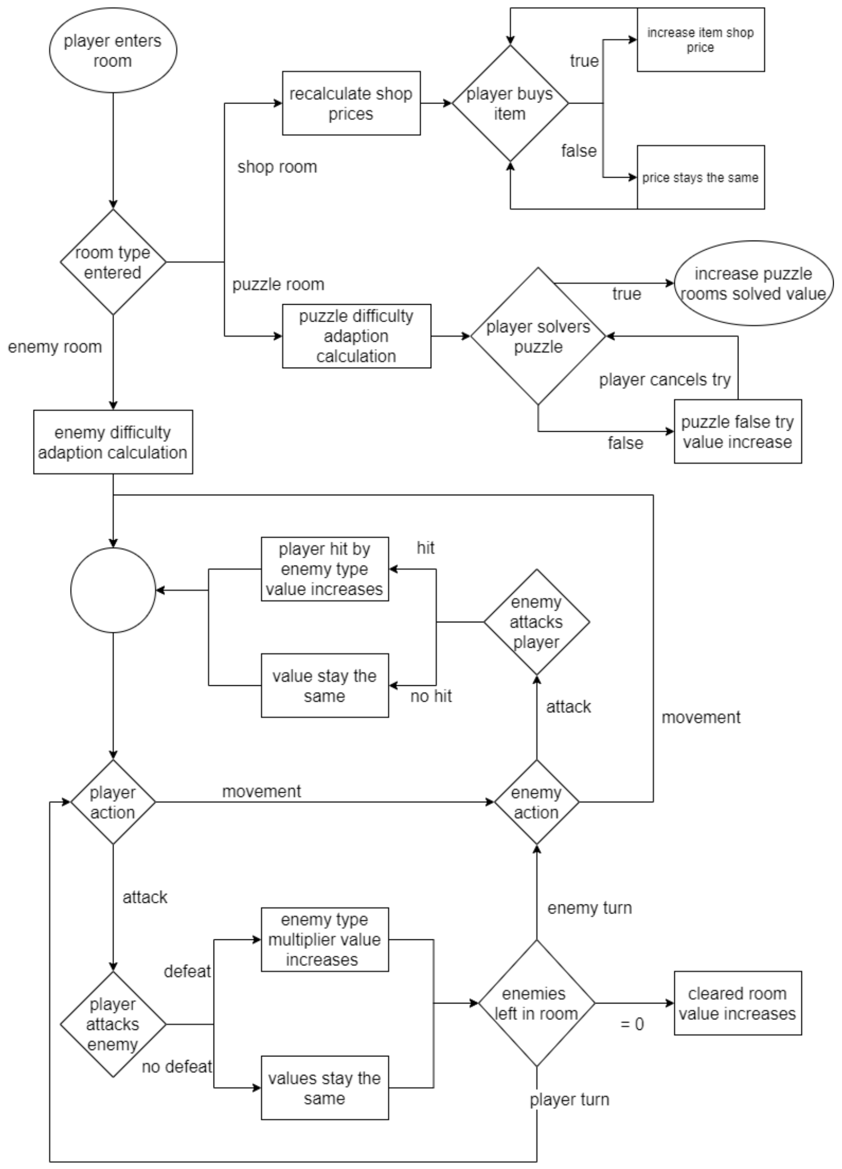


Figure 4.5: Adaption Process

5 Game Prototype Design

In order to combine and showcase the chosen PCG concept and how their mechanics play into each other a playable prototype of a game was created. Originally it was planned to only design a map generation algorithm supporting multi level barriers but to test it in a proper way the decision was made to design a game around it. The following section describes the design and decision making process of each essential element. Furthermore alternatives that were taken into consideration are discussed. Screenshots of different states during development are shown in the A to give a sense of progression. B shows some of the sketches created on pen and paper during the developement process.

5.1 General Design Philosophy

Most of the design process was done using pen and paper. Write all ideas down, work them out, see if they fit the project and are in the range of what is possible considering the preset time frame. For the map generation, different room sizes were drawn in multiple combinations to get an idea of the generation process and to look for edge cases. While testing the game all bugs and errors were documented in check list form. If not obvious what caused a bug multiple possibilities were listed and evaluated. This process helped pinning problems down and keeping an overview. By testing interactions that were not intended or not necessary for playing the game, the overall stability was improved.
To implement new features, all gameplay elements already created were taken into consideration. The idea was narrowed down to the base functionality. After this worked, further edge cases were tested and additional optional features added.
During development multiple features had to be redesigned completely. With better overall planing this could have been avoided to a certain degree. The process of redesigning included finding all nodes that are connected to the redesigned node and changing them within this process. Testing throughout the redesign process, is essential to keep track of the changes and notice bugs as early as possible.

5.2 Tiles and the Generation Process

The game started out only using tiles for walls and doors but after deciding on a turn based system, tiles were a logical way of implementing this system.
Every node instanced during gameplay, is placed on top of a tile. One big advantage of tiles is that they are designed to create big level structures and are optimized for that purpose. Enumeration values are assigned to each tile type. This simplified the creation of a turn based gameplay system.
For this project the size of each tile was set to 32x32 pixels. A small form factor was chosen because the sprites created use a 32x32 pixel grid. A bigger form factor would have been unnecessary.

The map is generation is designed to use online generation. The algorithm was created step by step using the starting room as the first step. It was narrowed down how much of the map should be generated at what stage of the gameplay loop. This resulted in only generating the adjacent rooms.
The number of doors is randomly rolled. This decides how many rooms have to be generated. The content of the room is not created during this phase. The type of the room and the content is generated when the player enters the room for the first time. In general the map generation is designed in such way, that it would continue to generate new rooms until the game is closed. For a Dungeon Crawler this wouldn't be practical in most cases, therefore a room limit can be set. When this limit is reached, no further rooms are generated and an exit to the next floor is spawned.
The first step of the generation process creates the starting room. This Room has walls of equal length. The first concept of the algorithm only supported square rooms of equal dimensions for all rooms. For the starting room, this rule stayed was kept until the end of the project. This restriction can be changed by including the checks used for other layout types.

Early during development the decision was made to support multiple room layout types. The effort was taken to show a variety of possibilities and flexibility of the generating process. To keep the game from having to make too many checks during run-time only multiples of the base room size are supported. To showcase this four different layouts were created. The layout types implemented are:

- Small Room
  A room of base dimensions. Every other layout on the same floor is a multiple of this.
- Big Room
  A Big Room is the base room size taken times 2 on both axis.
- Long Room Up Down
  A "Long Room Up Down" is a small room taking the y component times two.
- Long Room Left Right
  A "Long Room Left Right" is a small room taking the x component times two.

The room layouts are chosen at random and checked if possible to create. If there is not enough space, room types are disabled until one fits or none is left. In addition to random room layouts the combination of layout types can be set at the beginning of each floor. In this implementation it is chosen at the start of each new game and kept until restarting. These combinations are:

- Small
  Only small rooms are created.

- Big
  Only big rooms are created.
  For this layout configuration one extra check is implemented and one edge case defined. Firstly it is checked if a big room has space to be generated. Secondly if the starting room generates more than 2 doors, the direction to which the rooms expand have to be the same for all rooms, otherwise it would overlap or use a smaller room to fill the space.

- Long
  Only long rooms with either x double length or y double length are created.

- Small Big
  Small or Big rooms are chosen randomly.

- Small Long
  Small or Long rooms are chosen randomly.

- Big Long
  Big or Long rooms are chosen randomly.

Doors are created whenever the player enters a new room. For the starting room one to four doors are generated. This room has to have at least one door otherwise the no map generation would take place. For any other room zero to three doors are generated. If the current number of rooms generated is lower than the maximum number of rooms set to be generated then at least one door is created.
A door is created in the middle of a wall. If the wall consists of an odd value of tiles, the middle wall tile is replaced by a door tile. If the wall consist of an even value of tiles, the door can't be placed exactly in the middle and not be across two tiles. In that case, the door is placed off center. This is also taken into consideration when creating the adjacent room for the walls to line up correctly. The direction the player enters a door is disabled because a door room has to exist in that direction. Rooms bigger than base size have additional spawning locations for doors.
Rooms with walls double the base size have two spawn locations for doors. The locations are determined by halving the wall length and once again halving the resulted length. More locations for doors can be implemented, the bigger the rooms get.
The number of doors to be generated per room could be set to a fixed value. For this implementation the door generation algorithm tries to get as close as possible to create the set amount of doors without breaking the set rules.
One reason square rooms and only sizes of multiples were chosen was to keep the computational cost low. This makes creating more layout patterns and check extra conditions feasible.
If the generation algorithm is set to allowing all layout types to be generated, the door generation checks the tile beside the location of the new door. If there is no wall at least a base size room can always be created.

The option to choose a layout type to keep throughout the floor was implemented as one of the last things of the project. It showcases an attempt to further build upon a mixed approach generation and how simple it is to include new layout variations. For example for creating solely big rooms only one additional checks is needed.
Starting rooms are special cases. The order of door and wall generation is reversed. In this instance walls are generated at first and one to four wall pieces replaced by a door. The amount of cells to be checked for the creation process was narrowed down during refactoring.

After the location of the doors is set, walls are generated. If all layout types are allowed, the room layout is randomly chosen and checked if possible to create. If a small room was randomly chosen it needs no further checking and can immediately be spawned. All calculations to check if room layouts are possible are made from the cell of the room top left corner.
Big and Long rooms have multiple possibilities how they append to a room. Depending on the layout of the room appended to, layout options have to be disabled to avoid overlapping. The type of the tiles around the new rooms edges are checked. If there is a wall tile the option is disabled. This is done until a possible option is found that fits the restrictions. In the worst case until a small room is created.
The walls are created by looping the dimension of the new room and setting all edges to wall tiles. All tiles surrounded by the walls are set to floor tiles. Additionally one wall tile besides the door connecting the current and the new room is replaced with a floor tile, to make traversing rooms possible.
To choose the top left corner as a point of reference was decided early on. This streamlines the calculation process of each room no matter where it is in 2D space. Door locations were considered as an alternative point of reference but depending on their locations the calculations had to be different.

5.3 The Turn Based System

Due to the tile based nature a turn based approach to the gameplay was decided early on in development. All the other aspects of the game are build upon this system. To keep everything performant, hit-boxes and collisions were avoided. Checking cell values against each other for interaction was sufficient. Turn based action enabled an easier to observe way of difficulty adaption.
The turn based system was redesigned and refined multiple times throughout development. At the beginning only enemies and player had to signal each other when their turn was done. By introducing new room types, new kinds of interactions and animations it became way too complex to be handled by each class some helper variables and functions. To solve these issues a global turn controller was created.

Depending on the current room type the turn controller was adapted to fit the interaction. The turn order changes to incorporate the present mechanics.

If new room types were to be introduced to the game, the base turn system is already in place. It is possible to build upon it without breaking existing code.

5.4 Barriers

After the base map generation was up and running the first kind of barrier was implemented. Barriers are elements in the game that prevent the player from progressing. They require some task to be completed to progress. Every Barrier has their own rate of appearance and way of hindering the player to progress. Three different kinds of barriers were implemented to showcase ways of utilizing this concept. All barriers have to be solvable and dead ends have to be avoided. This is handled by keeping track of the amount of barriers created, barrier solutions spawned and not cleared rooms without a barrier. For all barriers a unique key is created consisting of five digits, each reaching from 0 to 9. This can be expanded by adding more digits or by expanding it using letters and other symbols. Whenever a barrier is created the associated key item is created as well.
Barriers that when solved give the solution for another barrier are called multi level barriers. Online PCG in combination with barriers allows for an easy way to enable multi level barriers by checking the current status of solved and unsolved barriers.

Door Barriers are the only kind of barrier that block off access to a new room. They are represented by a colored door. Corresponding keys with the same color drop to unlock these doors. The inventory and the map show key values in addition to the color. When created every door has a certain chance to be a barrier. During testing the barrier rate was adapted multiple times.
Color coding the door and key was a decision made from the get go. Using random colors for matching can be difficult if colors are too similar. Showing the key value in the inventory and map helps the player find fitting combinations.

Enemy barriers have a lower chance of appearing than door barriers. Every time an enemy is created it is assigned one of four enemy types. Only one of these types has a chance to become a barrier. Appearance wise barrier enemies work like door barriers. The key item is represented as a colored sword instead of a key. It is only possible to defeat a barrier enemy with the corresponding sword in the inventory. Barrier enemies prevent the player from clearing a room, consequently the player can not aquire the loot of this room.
A different matching system for enemy and weapon was considered but was discarded to keep the gameplay from getting too complicated.

Puzzle Barriers work like enemy barriers in regard of preventing the player from progressing. These barriers are represented by colored puzzle pieces that need corresponding colored puzzle key items to be solved. The thing that differentiates the puzzle from the the other barriers is that they are remote activated.

This shows a different approach to pickup and use. The player can check the map before picking up the key item to see which barrier will be unlocked. After clearing all barrieres in a puzzle room, the room can be solved like normal.

5.5 The Player

The player and available abilities were redesigned and build upon multiple times during development. The first draft the player had a fixed amount and a fixed order to execute actions during a turn. Move and attack was the only option. Some tools needing more actions to be used effective, enemies getting stronger and attack and movement pattern getting more complicated two actions per turn were not sufficient.
The player turn was redesigned. A turn consists of a certain amount of actions. This amount can differ depending on difficulty and upgrades. The player is free to mix and match any actions that are available. It is on the player to manage the action to optimize efficiency.
The player has a multitude of different stats. The amount of stats had to be increased during working on the prototype. The attack damage was one uniform stat for all tools. It was split into a separate stat for each one to incorporate the change of the difficulty adaption and the upgrade system.
Additional player stats include max turn actions, max and current health, max potions slots and current potions in possession were added. These stats, in contrast to the attack stats, are shown in the GUI.
It was considered to restart the whole game on Player defeat but to showcase adaption and for the purpose of the prototype the player is reset to the starting room. The stats and acquired items are kept intact.

Different tools were implemented with the focus on giving each multiple use cases by enabling different usages. Depending on the room type and object used on, the way tools interact with the world change.

- Sword The most basic action. It inflicts damage to the enemy one cell apart from the player.
- Magic This action was implemented to mix things up and give an alternative to the sword. This attack inflicts damage two cells apart from the player. It can last multiple turns and travel a path on its own. Depending on the room used in it either serves as an attack or to begin the puzzle solving sequence.
- Pick Axe Originally blocks were planned to be an object that was already in a room and had to be moved around in order to solve a puzzle. This idea was redesigned into the Pick Axe which is used to spawn and de-spawn blocks.
  - Puzzle Room Pick Axe
    In Puzzle rooms blocks are spawned representing a tree stump with

    an white circle in the center. By using the "Hand" tool directions can be added. These are visually represented by filling the circle inside with directional arrows. If a block is activated it will spawn one projectile in each activated directions.

  - Enemy Room Pick Axe
    To give blocks a purpose outside puzzle rooms they represent a bomb in enemy rooms. This block can be exploded to inflict damage. Using the "Hand" tool, counters can be put on the block. If the "Pick Axe" is used on the block it triggers an interaction. If no counters are on the block, it will disappear again, otherwise it explodes and inflicts damage multiplied by the amount of counters. By exploding one power block with counters on it it will activate all other power blocks with at least one counter on it, in area of effect range, creating a chain reaction.

- Hand The hand was introduced to make it easier to interact with power blocks. At first one had to hit a power block with the sword, walking around the block to activate its directions. This took way too long and required too many inputs. Furthermore some puzzles were not solve-able because one side ob the block was inaccessible by the player.
  During testing, the hand was given a new property. By using it on puzzle pieces, enemies or projectiles the player swaps places with the interacted object. This enables some new strategies in enemy rooms and keeps some small puzzle rooms solve-able

  - Puzzle Room Hand
    Locks the attack controls to the power block. By pressing in one direction the player can activate or deactivate this direction on the block. Player can swap places with a puzzle piece.
  - Enemy Room Hand
    Using the hand on a power block adds a counter to it. The number of counters is represented by a hand on the bomb counting up. The counts represent the multiplier applied to the base power block damage.

- Tools in Shops Every tool can be used to activate the shop blocks. Otherwise they work like in enemy rooms.

5.6 Magic Projectiles

Magic Projectiles was the mechanic that had to be redesigned and rebuild the most. The main problems included getting the right order of interaction between Magic Projectiles and the strength balancing. Magic Projectiles have a separate phase in the turn controller. At first, all projectiles moved after the player turn but this turned out to be weird visually and gameplay-wise. The problem was that when setting a projectile at the last action of a turn it immediately moved one cell. Because of this issue and due to the turn order redesign the projectile movement was changed.

- Puzzle Room Projectile
  Player Projectiles serve no purpose in puzzle rooms. A new type of projectile was created which is spawned when activating power blocks. Puzzle Projectiles do not interact with each other or the player. They interact with power blocks, puzzle pieces doors and walls. At first they had interactions with every other node but this led to many edge cases and racing conditions. When moving puzzle projectiles do not change the type of cells.
  With this projectile interaction was working correctly as long as there was at least one projectile on screen. Without any projectiles on screen but interactions still happening, an overflow happened. This was caused by blocks activating each other indefinitely every frame. This resulted in infinite loops and game crashes. After different tries to maintain the interaction order, a new type of projectile was introduced.
  The "Tickingprojectile", is spawned when a power block is activated and deleted when all other projectiles in the room are deleted. Its purpose is to move one step after all other interactions are done and signal that. The ticking projectile is invisible, has no interactions with anything and serves the purpose to maintain a set sequence for everything else happening.

- Enemy Room Projectile
  In enemy rooms projectiles of two types can be spawned being Player and Enemy Projectiles. Player projectiles interact with each other and can cause side effects. Side effects include, splitting into two smaller projectiles or merging when colliding. These interactions were added to give the projectiles uniqueness and make combat more engaging.
  One redesign was to move projectiles one after the other. This took too long visually and slowed the flow of the game down significantly. Moving all projectiles at once resulted in additional edge cases and bugged interaction caused by racing conditions. To solve these issues projectiles are sorted by their position in the room and movement and interactions are pre-calculated. Visually the movement can be shown all at once while keeping the interaction order intact.
  After multiple redesigns of the mage enemy type, enemy projectiles do not move anymore. The functionality to move them is still in the code and can be activated if required. Enemy projectiles interact with each other by deleting each other on interaction. If the movement direction is a zero vector projectiles are deleted on spot in the move projectile phase.

5.7 Room Types

Different room types were planned from the beginning to showcase the generated map and different difficulty adaptions. Each room has a room type randomly assigned when the player enters it for the first time. The room type defines how nodes interact with each other and defines a distinct clear condition. For the prototype three different room types were implemented to showcase diversity of the gameplay. Whenever a room is created the probability of all room types to be generated is adjusted. This balances the amount of the same room type appearing in succession.

- Enemy Rooms
  Enemy rooms were the first room type implemented. They have the highest chance of appearing because they offer more different combinations of nodes and therefore a greater content variety.
  Depending on the room size and current difficulty modifiers the amount and variety of enemies is adjusted. An enemy amount threshold was included depending on room dimensions. The cell to spawn an enemy on is set randomly and recalculated until all enemies found an empty cell. After the position and enemy type is set the enemy itself is generated including stats, behaviour, barriers and visuals.
  To clear an enemy room all enemies in it have to be defeated. All doors can be interacted with without having to clear the room.
  Enemies support dynamic difficulty adaption. Difficulty multipliers change the enemy difficulty globally. These changes not only take effect when new enemies are generated but also whenever the player reenters a room. This was changed from adapting the difficulty on the fly during combat to keep the gameplay consistent and movements and attacks predictable. Generated enemies adapt when reentering a room because otherwise the player could walk past rooms, spawn enemies in the process without defeating them and keep the difficulty low. When playing the game as intended difficulty is adapted according to the players performance.
  To add an additional challenge without breaking the current gameplay loop, help enemies were created. If help enemies survive until the room is cleared they drop extra rewards. To save them is optional and encourages to play in a risk reward manner.

  Enemies were the first gameplay elements implemented. To give an overall idea of the turn based combat system, only one enemy type was planned. After focusing on difficulty adaption multiple enemy types attack and movement patters had to be created.
  Enemies are only active in the room the player is in meaning interactions are only possible within the current room.
  To showcase forms of adaption four different enemy types were designed. Each type has different base stats, different movement and attack patterns. The visual design is distinct to help the player memorize movement patterns and differentiate them.
  Each enemy type has different movement patterns. Patterns were redesigned multi times during development to better fit the adaption and for a balancing purpose. One enemy type for example moves towards the player, another one moves to predefined positions in the room after be-

ing hit. Movement patterns are adapted by changing the amount of cells passed per move or unlocking new positions in the room to move to. The patterns were created to showcase a possible adaption of the enemy game element.
Movement is pre-calculated for each enemy, they visually move at once while interactions are handled one after the other beforehand.
Attack patterns were really simple at the start of the project and could not be adapted in any meaningful way. A separate algorithm was created to streamline the pattern creation process. The cells are chosen in one direction and then mirrored in up to three directions. Using this method patterns can be created by the game designer or randomly by the game. For the prototype it was chosen to create pre-defined patterns to better evaluate the adaption approach. Depending on the current difficulty multiplier, patterns the amount of covered cells changes. On every cell an enemy can attack to, a colored texture is placed. The player can toggle this visual cue on and off. It is possible to see the cells the player would take damage on. These areas are updated whenever an enemy changes cell.
One kind of enemy can also spawn as a barrier. Different types of weapons were planned to act as key items. To streamline the combat, showcase the mechanics and keep things from getting over complicated this approach was scrapped.
During designing coins and tool upgrades a new way to engage with rooms had to be created. This was done to enable dropping additional items and making the gameplay more versatile. Help enemies are instanced using the same scene as normal enemies. They are assigned a help enemy variable, look different visually and have no means to attack. To incorporate this mechanic the ability of enemies attacking other enemies was added. The amount of help enemies saved in total influence the difficulty of the game.

- Puzzle Rooms
  Puzzle rooms were created to show another use case of the procedural generated rooms. Additionally they provide variety for the player.
  To find a way to procedural generate puzzles, that are always solvable and do not get monotonous too quick was a challenging part. Many ideas were scraped. One of the first ideas was to move blocks around to form a specific pattern. After designing and rethinking this idea it would have been a lot of pushing objects around and taken too long to solve. The main goal for the design was to make it appealing to find a somewhat optimal solution for the puzzle.
  Puzzle pieces are randomly placed in the room. A pattern is played whenever entering a room. This pattern can be replayed on button press. The feature to replay the pattern was added during game testing to speed up the process.
  To clear a puzzle room, power blocks have to be used to activate puzzle pieces in one sequence. The fewer blocks are used the less time it takes

to build the solution therefore it is encouraged to build efficiently. The order puzzle pieces are activated in is saved. If all are activated this is compared with the initial order. If the order is correct the puzzle is solved. Otherwise the puzzle solve sequence is interrupted and all pieces are reset to be deactivated for the player to try again.
A puzzle piece is assigned a random color at creation. Different colors makes it easier to remember the pattern. A puzzle piece can be activated only once per solving sequence. If all pieces are activated they play an animation to let the player know. Solving a puzzle can be canceled by the player. In that case the piece return to its initial state.
In older drafts the solving process of puzzles could not be canceled. When creating a loop that repeated endlessly or it became obvious that the solution was built wrong, the player could not continue the game or had to wait and watch until all interactions were done.
Puzzle pieces can be barriers. They do not play an animation and the sprite is represented by a cracked darker colored stone. If the barrier is solved on entering the puzzle room the barrier becomes a normal puzzle piece.
The power blocks were designed in a way to support looping. Meaning, a block can be activated multiple times during the puzzle solving phase. This mechanic is not necessary for the normal puzzle solving but taken the design of the blocks into account it seemed like an obvious feature. Getting this mechanic to work properly was really difficult and time consuming but opens up many different ways of designing puzzles. To showcase one use case of this property counting blocks were created.
A counting block has a counter on it showing how many times it has to be activated during the puzzle solving process. Depending on the difficulty multiplier this number varies. If the number hits zero, minus one or one when the last puzzle piece is activated and the puzzle is solved correct, a bonus reward is dropped. Originally the counter had to reach exactly zero but the scope was changed for difficulty reasons.
Counting Blocks add complexity to the clearing process but in return offer a way to gain additional rewards.

- Shop Rooms
  Shop rooms were the last room type designed for the game. It feels more rewarding choose what to upgrade than to simply adjust these stats in the background. The games keeps on getting harder nonetheless with the upgrades the player has a possible way to counteract that for a longer time.
  All the base stats of the player can be upgraded. The prices adapt globally each time an item is bought. More valuable upgrades such as an extra turn action cost more to begin with and rise in price faster.
  The upgrade process was redesigned multiple times on paper. One idea was to reward the player with a random upgrade after clearing a certain amount of rooms. Another idea was rewarding temporary upgrades after

finishing rooms that would last a certain amount of turns. The shop system was created to let the player personalize the way of approaching combat and to encourage the clearing of optional challenges.

5.8 Adaption

One of the core aspects of the project is adaption. The focus was set on difficulty adaption depending on user actions and progress. The more stats tracked, the more personalized and nuanced the adaption process can get. For the purpose of the prototype, stats were chosen in such way that the adaption process can be observed easily by the player.
For enemy rooms, stats include how many enemies of one kind the player has defeated, how many times the player was hit by a specific enemy type and how many enemy rooms were cleared in total. The more enemies of one kind are defeated the stronger this enemy type becomes. This also influences the difficulty of other enemy types, but the magnitude of influence is lower. Thresholds were defined to keep the game playable and to keep it from getting too easy. Using these factors, the game adapts according to the player performance.
In Puzzle rooms the adaption is not as detailed but it also influences the optional challenge. The more often the player activates all puzzle pieces in a wrong order the less puzzle pieces will spawn. This is also capped by a threshold to keep the player from abusing this adaption process. The more puzzle rooms are cleared, the more puzzle pieces a room will contain, therefore the pattern becomes harder. Counting blocks also adapt by setting a lower or higher starting value.
Alternative methods to capture player progress were thought through of but did not seem fitting. Stats like time spent on the floor can vary greatly depending on the position of barriers and their solution. The number of steps taken in a rooms were recorded but not used in the end. To play optimal and not take damage or build a working puzzle solution the amount of steps necessary can vary greatly. Due to the procedural nature of the game it is hard to estimate a threshold.
A mixed approach was introduced in the end by letting the player choose a difficulty at the start of the game. Auto difficulty is the default option. Using this option the game tries to balance in a way that the player neither struggle to complete a challenge nor rushes through the game because it is too easy. A global difficulty multiplier is adjusted during gameplay. Using one of the other three options, the global multiplier has a fixed value. The game is also adapted but should represent the chosen difficulty throughout.

5.9 Items

Items were introduced as a reward for clearing a room. In general they are a necessity for the way barriers in the game work. Non key item simply have a key value of zero and present other benefits to the player.
Whenever a barrier is created the corresponding key item is created and saved

for later. Key items have a certain chance of dropping. The fewer possibilities left for it to drop the higher the chance of dropping. The different key items include:

- Key
- Sword
- Puzzle Piece

This was the base concept of items used to solve barriers present in the first draft of the prototype.
To add more variety additional non-key items were created. One idea scrapped was to include temporary upgrades at items. The effect would last a certain amount of rooms. This idea was replaced by the coins and permanent upgrade system. The exit was implemented as an item to appear after a room is cleared. The chance of it appearing gets higher the less rooms are left to be cleared on the current floor.

- Coin
  Normal gold coins increase the coin counter by one. Nickles aka silver coins add five to the coin counter.
- Heart
  Half hearts and full hearts are available as items. They fill the respective amount of the players health.
- Potion
  Potions fill one potion slot in the inventory. They can be used during the player turn to fill up a certain amount of health.
- Exit
  On interacting with the exit, the player is transported to the next floor and a new map starts generating.

5.10 Camera, Map, Inventory, GUI

The camera was completely redesigned during the development process. It follows the player with a smooth room to room transition, zooms in and out according to the room size. For bigger rooms it zooms out a bit more than the room size to prevent the GUI from overlapping with the room.
The map shows all rooms currently generated including doors connecting them. Key items required and already cleared rooms are highlighted. The camera can be controlled freely to view the entire map, zoom in and out and to check where to go.
The map was created to keep even large scale maps with multiple barriers playable. Using the actual game world created by overlaying it with textures to highlight barriers and cleared rooms was an idea resulting from debugging and tinkering with the camera. The alternative idea was to track all rooms and

connections and create an extra scene drawing rectangles in relation to the room sizes.
The inventory was created to keep track of key items in the player possession. It can be opened on the map screen and during gameplay. Barriers and key item values can be compared to plan out a path through the dungeon. The inventory has two tab categories. One for keys and one for weapons to keep the menu tidy and clear.
The GUI gives an overview of relevant stats. Changes are updated and shown immediately. The layout and size of the GUI elements was changed multiple times to keep it readable and from overlapping with the room content.

5.11 Mixed Approach

There are many elements that can be used to offer more control over the generation. For example which room types should appear the most, which enemies to spawn in enemy rooms, selecting attack patterns. In the end, to not go beyond the scope, only a few were chosen as an influence of the main PCG algorithms. Firstly the room layout combinations can be chosen including room size and max rooms to be create and secondly the difficulty adaption can be influenced easily. Including and balancing other stats may take some extra time and effort.
The main menu lets the player change settings influencing the generation and difficulty. In combination with the pause menu it removes the necessity to restart the application to change settings or start a new game.

5.12 Visualisation and Immersion

The goal was to be distinct and coherent at the same time. No sprite animation or sound should feel out of place. In the end everything should come together naturally.
A prototype of most sprites was created to test size, colors and animations. They were later replaced by unique and higher quality variations. Distinct sprites are important especially to distinguish enemy types. This helps learning and remembering patterns.
All animations were created to be distinct and fitting to the sprite. They bestow an impact upon actions and make gameplay more enjoyable overall.
To set a mood and support the visuals, music and sound effects were created. The music loops in the background changes the pace depending on the current active scene. Sound effects add an extra feedback to actions and interactions.
The controls got a bit complicated over time. Therefore a button mapping is provided in the GitHub repository. All actions are bound to keys and the game is completely playable using a keyboard. However the controls were designed with a Nintendo Switch Pro controller in mind. .

6 Evaluation and Discussion

The main goal is playability throughout. While playing no dead ends are allowed to occur and every room has to be clear-able.
The main method of evaluation for the created game was testing throughout development. For the final prototype, additional testers were used to verify playability throughout. On top of that, a short questionnaire was created to gather additional insight. The questionnaires can be found in C.

6.1 Testing the prototype

Testing was given a big focus throughout development, due to the complexity of the game mechanics working in coherence. Since the first working prototype an alpha tester, in addition to the main developer playtested the game. Every major feature added or rework done was tested in a multitude of scenarios.
Playtesting sessions were supervised. Newly implemented features were tested in solitude. Whenever a feature was working without breaking the game, everything implemented until this point was tested altogether. Bugs and feedback were documented and considered in further iterations. The main goal of testing with another person, besides finding bugs, was to gather feedback on difficulty, game design, and gameplay pacing. One additional thing that was asked from the tester was to play the game in unconventional ways. Using the created game mechanics in other ways than intended, lead to finding many edge cases early on and therefore creating a more stable prototype.
Nearing the end of the project, two additional testers were introduced to the project. During this phase, the prototype was working according to the testing conducted. This additional phase was mainly done to gather more feedback and find unnoticed bugs. Furthermore, the feedback gathered was used to improve the usability of the prototype and provide a better more streamlined experience to the game mechanics for the final test group.

To test the final Prototype, six people in total participated. In addition to three that never played the game, the alpha tester and the two people testing the nearly finished one played once again. The controls were provided and a short explanation of the game mechanics was given.
Each tester had to play for 20 minutes. The game was played with all elements enabled, the room layout set to mixed, and the adaption set to auto. The max amount of rooms to generate was set to fifteen for the starting floor. This amount was decided on to avoid too much backtracking during the test playtime. The task was to keep playing until the time runs out. If this was possible, the main test was deemed successful.
In addition to that, after the play session, a short questionnaire was conducted regarding difficulty. This was mainly to find out if, the adaption the games try to achieve, is appropriately balanced. General feedback and difficulties during gameplay were asked for. In the end, it was revealed, that the map was procedurally generated and the testers were asked if this hindered their experience.

6.2 Results of the Evaluation and Discussion

During the final test, all test participants were able to complete their play session without any problems. Every barrier encountered was solve able and no dead ends were encountered. Therefore the main goal of the test was achieved. The online algorithm and barriers were tested in a hands-on scenario and everything worked as expected.
A short questionnaire was held after the play session concluded. This mainly focused on progression and difficulty. The alpha tester was excluded because all mechanics were presented to him openly in earlier stages of development. One of the beta testers, who already knew how to play the game, found it to be challenging at the beginning but managed to figure things out. This result most likely comes from them playing better, therefore the game ramping up the difficulty faster to start with. The resulting managing of the difficulty can be traced back to the balancing adapting.
The three testers new to the game found the difficulty neither to be too easy nor to be too challenging. The difficulty of enemy rooms was perceived to be normal to slightly harder by the testers. Puzzle rooms were conceived to be pretty balanced. Nonetheless, the main problem with those was not the difficulty but the complexity of the controls and the ambiguities of the puzzle room mechanics. The controls resulted in some wrong inputs and slowed down progression.
Overall the game was perceived well. The uniform assets engaged the players. The turn-based combat and strategic nature were appreciated. The answers can be summed up as being invested in the game mechanics and the gameplay flow. The last question revealed to the player that the map was online generated and if this hindered their experience in any way. For the testers, the content of the rooms was the focus, and the map just a tool to enable it. One tester answered that no matter if the map was created beforehand or afterward would have made no difference for the enjoyment.
The following table shows the stats tracked during the playthrough of each participant.

	Tester1	Tester2	Tester3	Tester4	Tester5	Tester6
rooms cleared	41	34	36	27	32	30
enemy rooms cleared	25	22	22	17	21	18
puzzle rooms cleared	10	7	9	7	7	8
shop rooms cleared	6	5	5	3	4	4
barriers encountered	24	20	19	17	16	18
door barrier	12	7	8	5	4	10
enemy barrier	9	10	7	8	5	6
puzzle barrier	6	3	4	4	7	2

Figure 6.1: Stats final Prototype Testing

During alpha testing, many features and functionalities were redesigned. Testing was an important part and helped during this process. For a project of this scale having more testers at an early stage would have slowed down development.
Due to the game being playable over the tested time frame, the functionality of the created online approach was deemed successful. The multi-level barrier mechanic was tested and confirmed working during this process too. Therefore confirming that online generation used for dungeon crawler games presents a feasible solution.
For this scenario the adaption worked well, balancing the difficulty to keep the player invested, by neither making it too easy nor too challenging. Nonetheless, further improvements and additional tracked parameters to improve the adaption flow are advantageous.
Regarding the map generation and the gathered feedback, the online generation had no recognizable influence on the enjoyment of the game. In many cases, the map provides a way for a game to present the game mechanics coherently.
These tests show that online generation is neither a show-stopper [2] nor is it unsuitable for the dungeon generated as stated in [13] . Additionally, the barriers were tested in all use case scenarios therefore providing an additional approach to this fairly underutilized mechanic while banking on the benefits of the online generation.
To further streamline and improve testing, a short tutorial section could be created. This would explain the controls and give an overview of the mechanics.

7 Future Works

The prototype game shows one way multiple PCG techniques can be implemented in coherence. It provides a use case for the map generation using an online approach and enabling different kinds of barriers. Some ideas for future works building on top of this thesis are discussed in the following.

Regarding room layout options, this implementation shows the baseline of what is possible. By keeping the concept of room sizes being multiples of the base size, many more layout options can be created by slightly altering the checked cells for each option. Furthermore, if the room size is at least double the base size, more door location options can be added. Multiple locations are implemented but limited to four doors per room, this can easily be extended. This allows for more rooms to be spawned adjacent to one room and therefore leads to more diverse layouts and interconnected paths.

The game prototype showcases one possible game the map generation algorithm can be used for. The turn-based system and even the cells used for the gameplay were design choices. Tiles support different properties like collision as base features. This property can be used for designing a game with real-time interactions using collision boxes. The pattern algorithm created for enemy areas can also be used without using cells, for example by giving an enemy different hitboxes. What was created are building blocks and a basic showcase of how they can fit together.

The use case of what a barrier can be and what action is needed to solve it can be altered and expanded upon. The important part created, is to keep every room accessible and playable while still supporting multi-level barriers.

The adaption was mainly focused on difficulty. The tracked stats and adaption present just the tip of the iceberg of what's possible. Room sizes and layout types could be included in this adaption depending on player movement patterns. By showcasing four difficulty settings a base idea was presented of how to change gameplay to adapt dynamically to the player.

To adapt the enemy difficulty an additional algorithm was created to draw different patterns with minimal developer input. These patters can be expanded for creating different game aspects. One use case for example would be placing obstacles in a room. Different patterns could be predefined or randomly generated. A base pattern could be created by the developer and evolved by the algorithm, checked again by the developer and so on, this would further support a mixed approach during the generating process.

Especially Power Blocks represent an interesting game mechanic letting the player find an individual solution to a puzzle. The puzzle mechanics were created in such way that, even though they are randomly placed in the room, they can always be solved. This puzzle mechanic can be further extended or the power blocks used for totally different gameplay parts.

8 Conclusion

The focus of this research paper was set on the lack of online approaches and barrier creation discussed in [27] and [13]. According to [2] online generation presents a show stopper when taking playablity into account and [13] states that the nature of dungeons might discourage an online approach. Limitations of using online generation were addressed by building an algorithm enabling barriers and always guaranteeing playability. The project created showcases the benefits of using an online approach. In particular on the concepts of barriers and content adaptation. Multiple algorithms were built to create a fully playable game combining these approaches.

To evaluate the prototype, multiple test participants were asked to play the game. The results showed that everything worked as anticipated. Therefore the online map generation and barrier algorithm can be seen as a valid approach. The conducted questionnaire shows that the difficulty was perceived as balanced, this suggests that the adaption approach works as intended. Summarizing the testers feedback, using an online approach by generating the map during gameplay, has no negative effect on the enjoyment of the game. For testing purposes, the fundamentals of each game element were implemented to present a use case for the map and barrier generation. Elements to expand upon and limitations are addressed 7. This, for example includes the presented room layout types and how they can be improved.
Additionally due to the nodes and scenes approach used by the Godot engine, as discussed in 3.1, the prototype can be expanded and modified resulting in a modular design.

Procedural generated Dungeons are one PCG technique, that is not just a concept but widely used in a multitude of video games. However, generating procedural dungeons still presents a complex problem in each building step. Every single step offers the possibility to explore different solutions. New approaches can enable diverse game designs and gameplay ideas but simultaneously complicate other aspects. Currently there is no prospect of a blueprint solution that will work for everything. Consequently creating new PCG algorithms for different use cases remains a field of high interest that offers many possibilities to contribute.

References

[1] Art, O. G. Opengameart.org. https://opengameart.org/. (Accessed on 07/26/2020).

[2] Barriga, N. A. A short introduction to procedural content generation algorithms for videogames. *International Journal on Artificial Intelligence Tools* (05 2018).

[3] Capello, D. Aseprite - animated sprite editor & pixel art tool. https://www.aseprite.org/. (Accessed on 07/26/2020).

[4] Clark, R., Trujillo, O., and Clark, K. Crypt of the necrodancer on steam. https://store.steampowered.com/app/247080/Crypt_of_the_NecroDancer/. (Accessed on 07/26/2020).

[5] Descottes, J. Piskel - free online sprite editor. https://www.piskelapp.com/. (Accessed on 07/26/2020).

[6] diagrams.net. Diagrams - diagrams.net. https://app.diagrams.net/. (Accessed on 07/26/2020).

[7] DrPetter. Bfxr. make sound effects for your games. https://www.bfxr.net/. (Accessed on 07/26/2020).

[8] Eaglegold. Pokemon mystery dungeon map by eaglegold on deviantart. https://www.deviantart.com/eaglegold/art/Pokemon-Mystery-Dungeon-Map-448926493. (Accessed on 07/26/2020).

[9] Foundation, K. Krita — digital painting. creative freedom. https://krita.org/en/. (Accessed on 07/26/2020).

[10] Games, E. The most powerful real-time 3d creation platform - unreal engine. https://www.unrealengine.com/en-US/. (Accessed on 07/26/2020).

[11] Games, Y. Make 2d games with gamemaker — yoyo games. https://www.yoyogames.com/. (Accessed on 07/26/2020).

[12] GitHub. Github. https://github.com/. (Accessed on 07/26/2020).

[13] Linden, R., Lopes, R., and Bidarra, R. Procedural generation of dungeons. *Computational Intelligence and AI in Games, IEEE Transactions on 6* (03 2014), 78–89.

[14] Linietsky, J., Manzur, A., and contributors. Godot engine - free and open source 2d and 3d game engine. https://godotengine.org/. (Accessed on 07/26/2020).

[15] McMillen, E. The binding of isaac on steam. https://store.steampowered.com/app/113200/The_Binding_of_Isaac/. (Accessed on 07/26/2020).

[16] Moore, D. X. Mario paint music composer - danielx.net. https://danielx.net/composer/. (Accessed on 07/26/2020).

[17] Nesky, J. Beepbox. https://beepbox.co/. (Accessed on 07/26/2020).

[18] NV, I. L. Fl studio. https://www.image-line.com/. (Accessed on 07/26/2020).

[19] Pereira, L. T., Prado, P. V. S., and Toledo, C. Evolving dungeon maps with locked door missions. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (2018), pp. 1–8.

[20] Shaker, N., Togelius, J., and Nelson, M. J. *Procedural Content Generation in Games*, 1st ed. Springer Publishing Company, Incorporated, 2018.

[21] Shaker, N., Togelius, J., and Nelson, M. J. *Procedural Content Generation in Games*, 1st ed. Springer Publishing Company, Incorporated, 2018.

[22] Soundation. Soundation — make music online. https://soundation.com/. (Accessed on 07/26/2020).

[23] Team, T. G. Gimp - gnu image manipulation program. https://www.gimp.org/. (Accessed on 07/26/2020).

[24] Technologies, U. Unity real-time development platform — 3d, 2d vr & ar engine. https://unity.com/. (Accessed on 07/26/2020).

[25] Togelius, Kastbjerg, J., Schedl, E., Yannakakis, D., and N., G. What is procedural content generation? mario on the borderline. *AGameComIn* (2011).

[26] Twin, M. Dead cells on steam. https://store.steampowered.com/app/588650/Dead_Cells/. (Accessed on 07/26/2020).

[27] Viana, B. M. F., and dos Santos, S. R. A survey of procedural dungeon generation. In *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)* (2019), pp. 29–38.

[28] zeldadungeon. Eagle-map.png. https://www.zeldadungeon.net/Zelda01/Walkthrough/02/Eagle-Map.png. (Accessed on 07/26/2020).

A Prototype

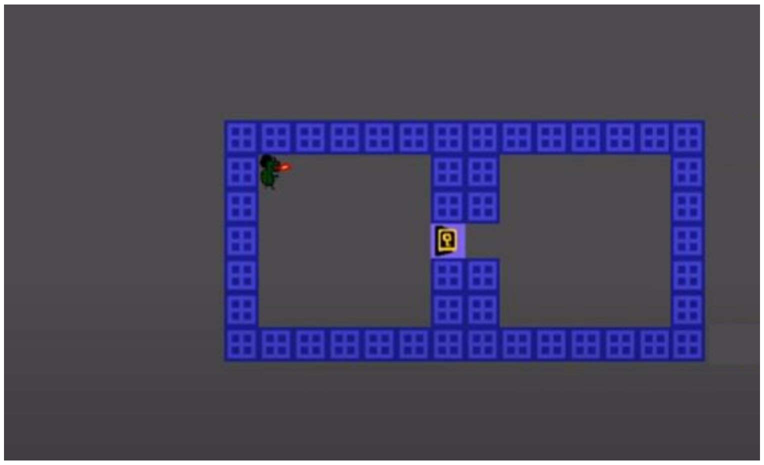


Figure A.1: First generated Room

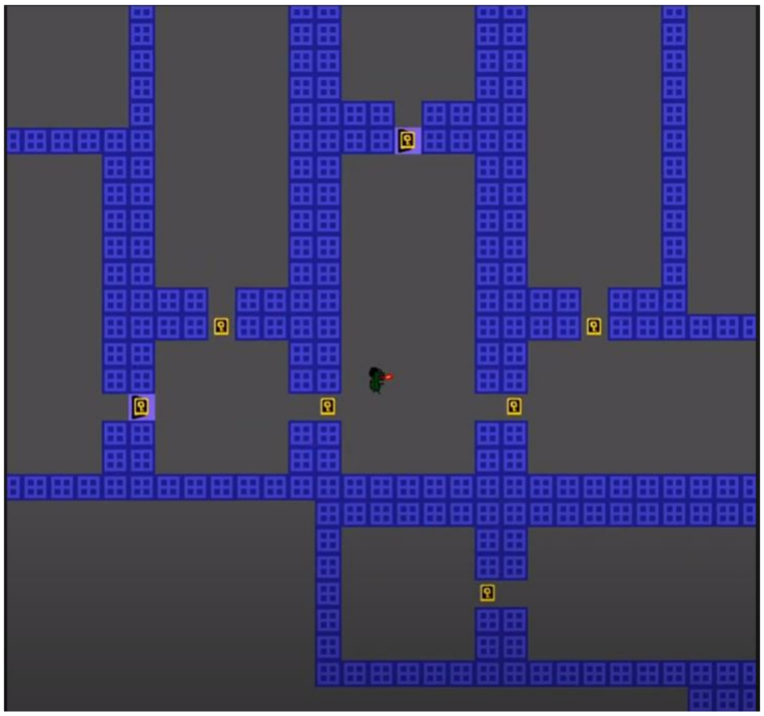


Figure A.2: First Map Generation Prototype

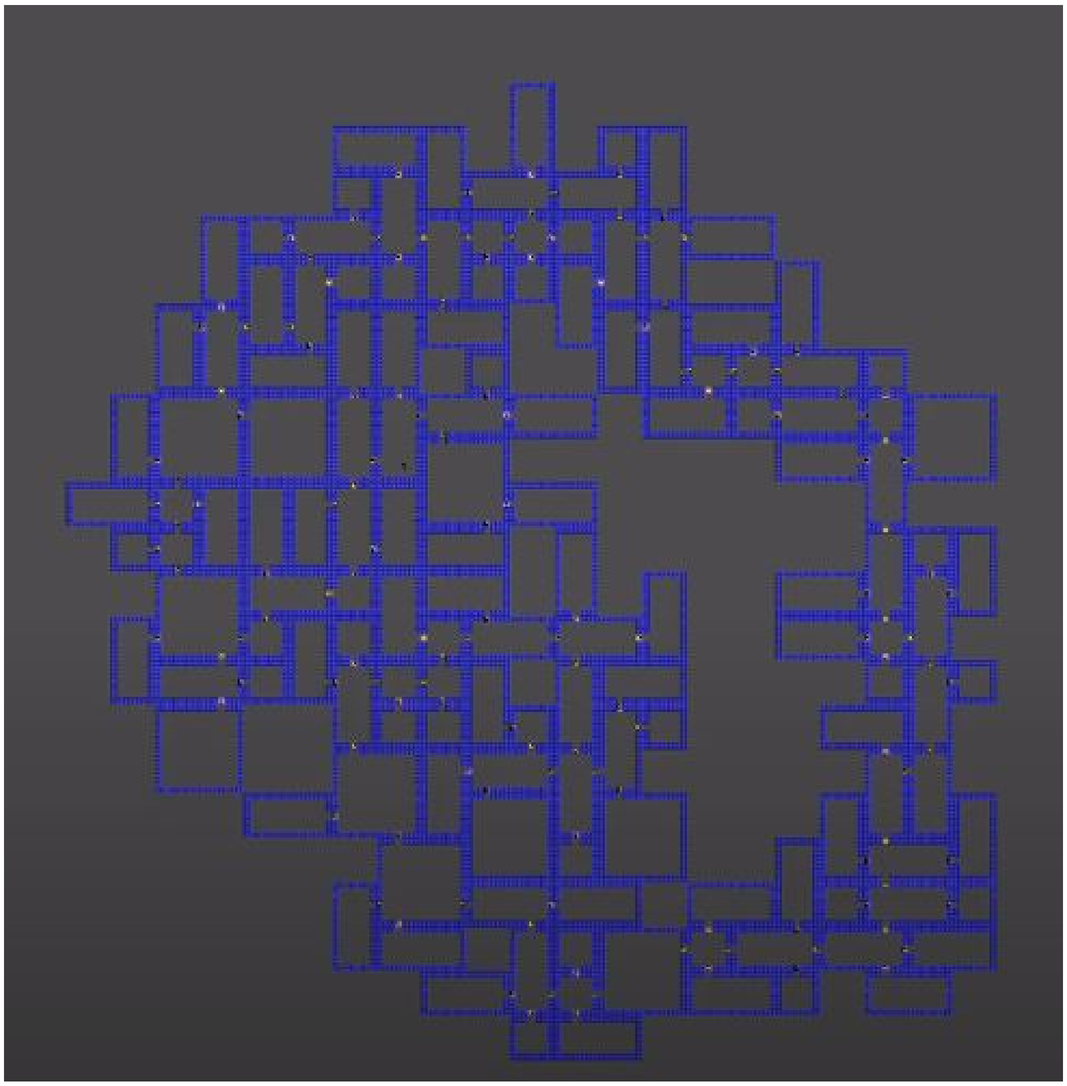


Figure A.3: First Prototype testing Map Generation

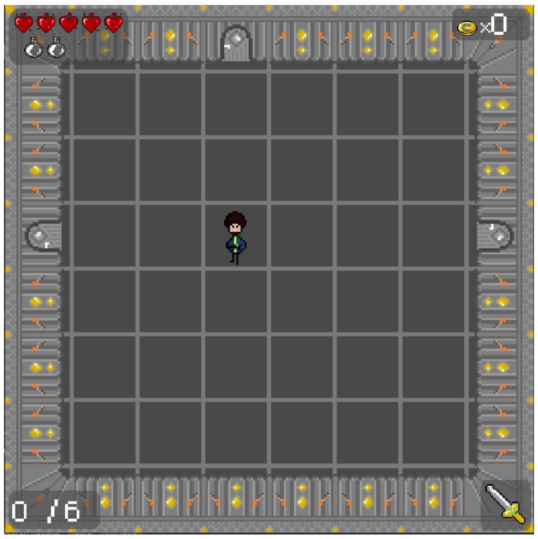


Figure A.4: Final Prototype Starting Room

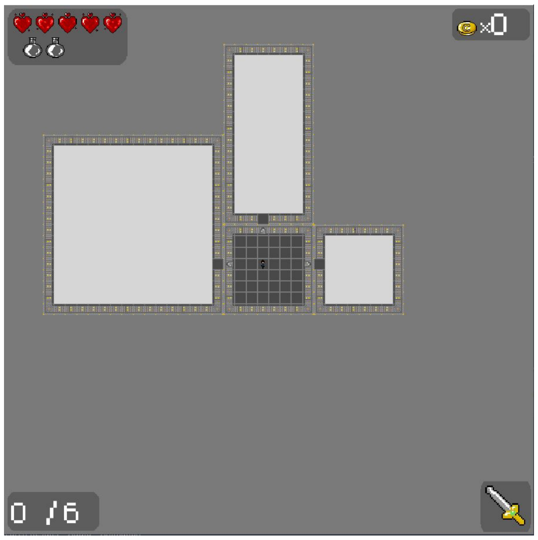


Figure A.5: Final Prototype Starting Room Map Generation

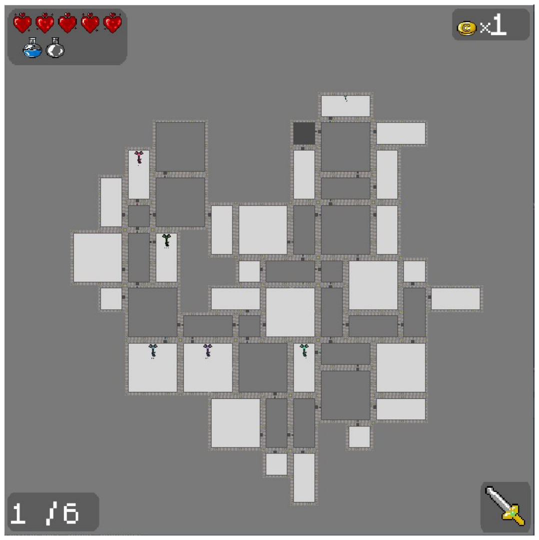


Figure A.6: Final Prototype Map Generation

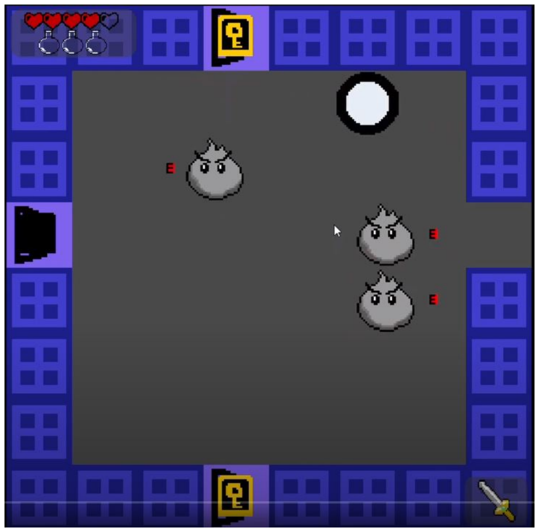


Figure A.7: Prototype Debug Enemy Room

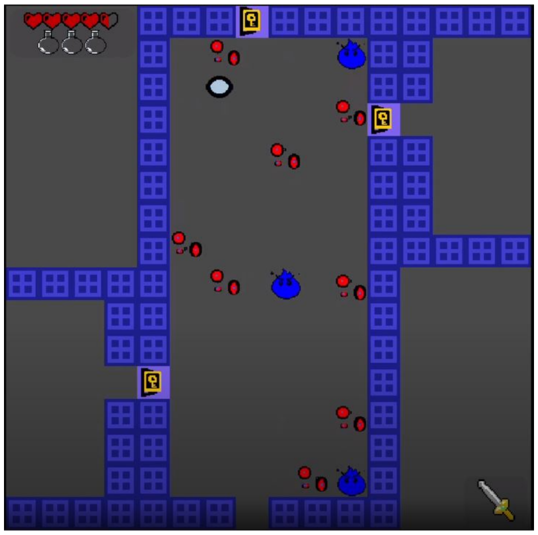


Figure A.8: Prototype Debug Enemy Room Projectiles

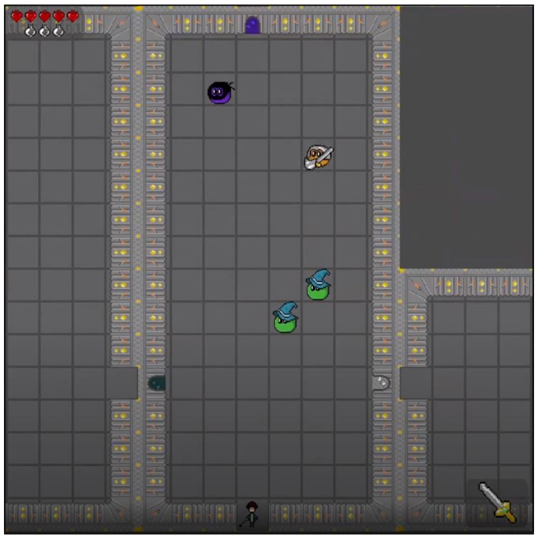


Figure A.9: Enemy Room Second Prototype

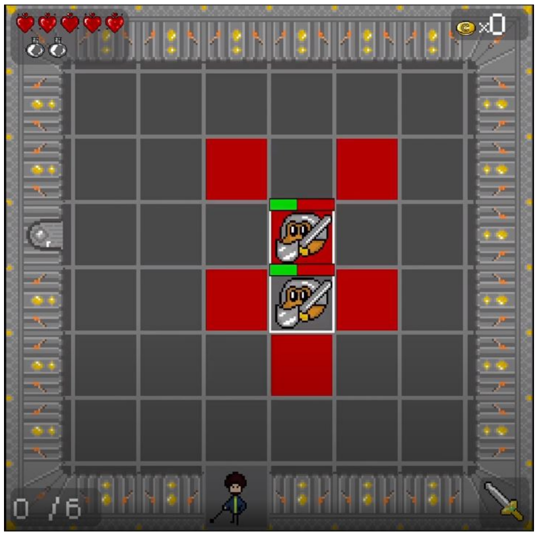


Figure A.10: Final Prototype Enemy Room

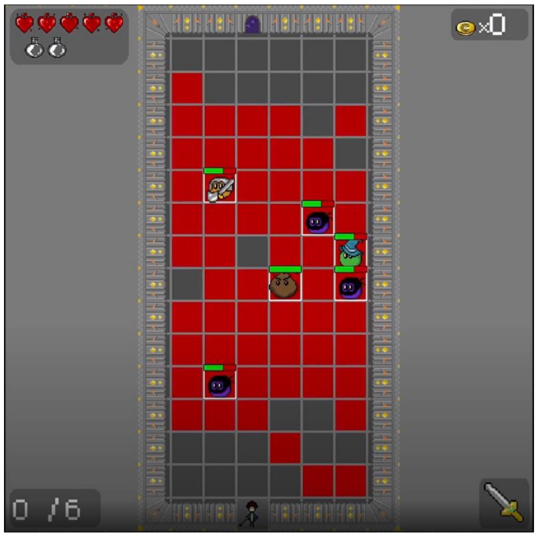


Figure A.11: Final Prototype Enemy Room (Mixed enemies & Attack Patterns)

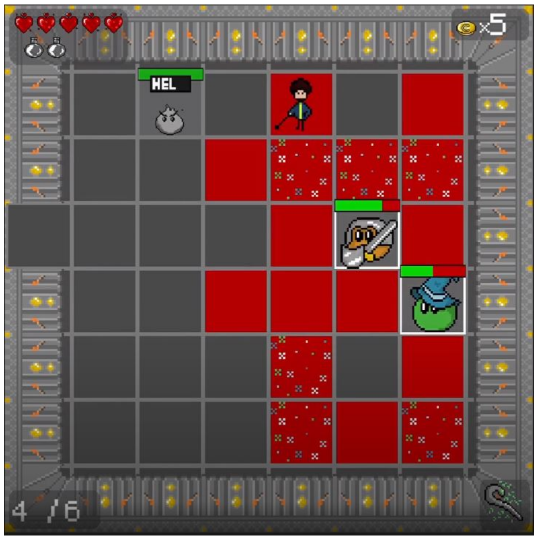


Figure A.12: Final Prototype Projectiles & Help Enemy

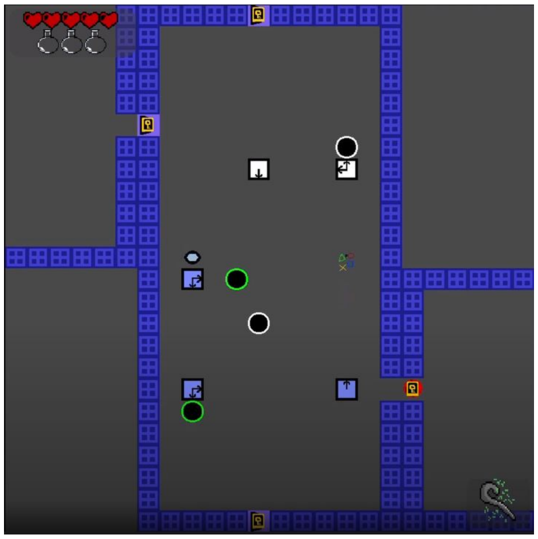


Figure A.13: First Prototype Puzzle Room

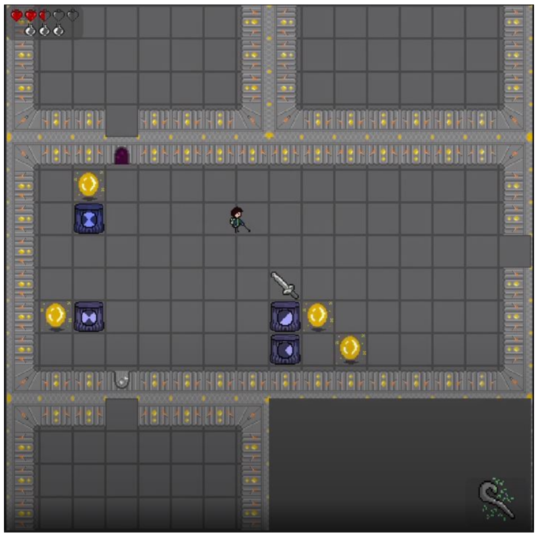


Figure A.14: Second Prototype Puzzle Room

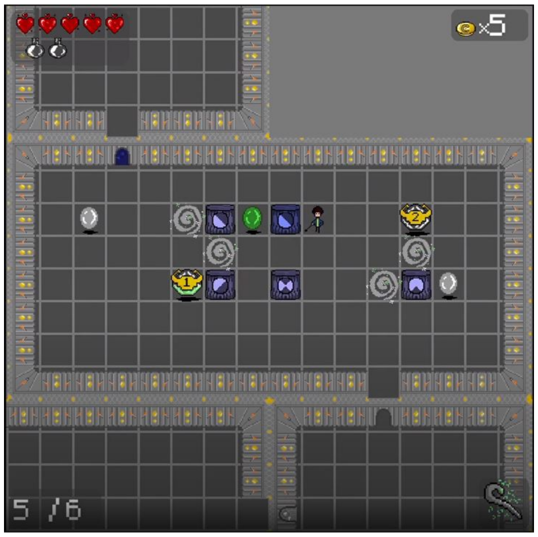


Figure A.15: Final Prototype Puzzle Room

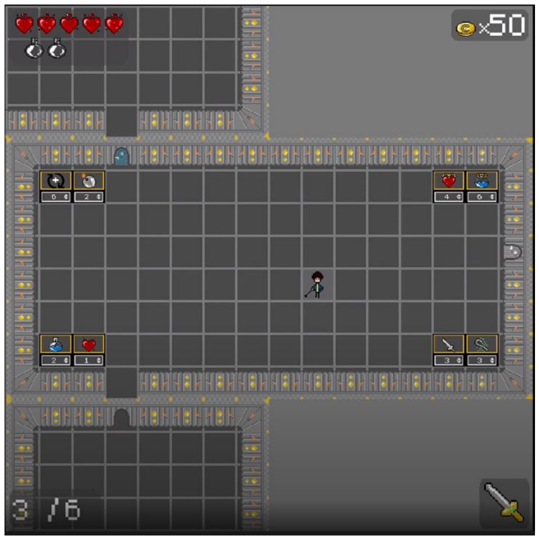


Figure A.16: Final Prototype Shop

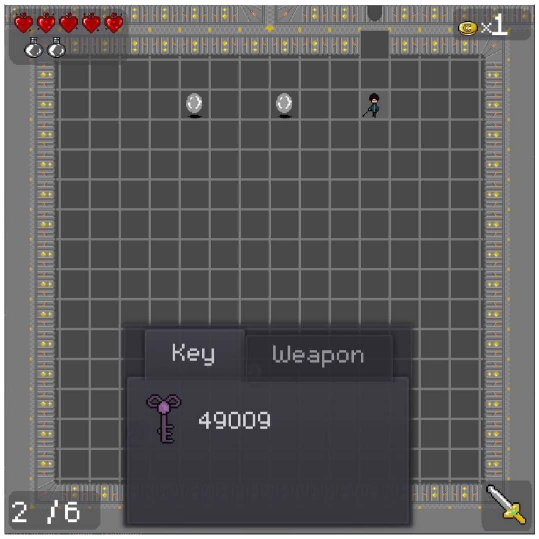


Figure A.17: Final Prototype Inventory Menu

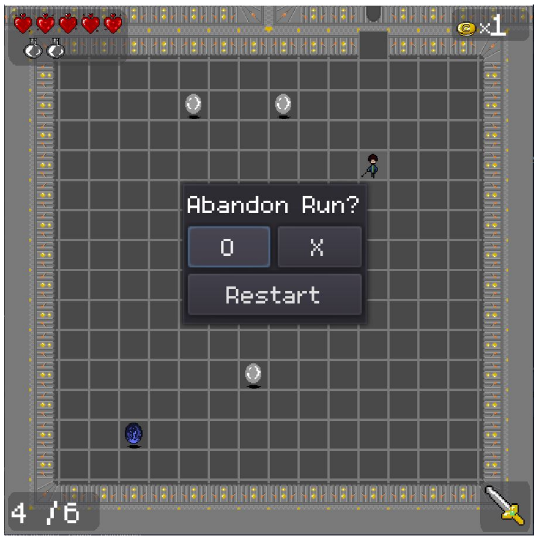


Figure A.18: Final Prototype Pause Menu

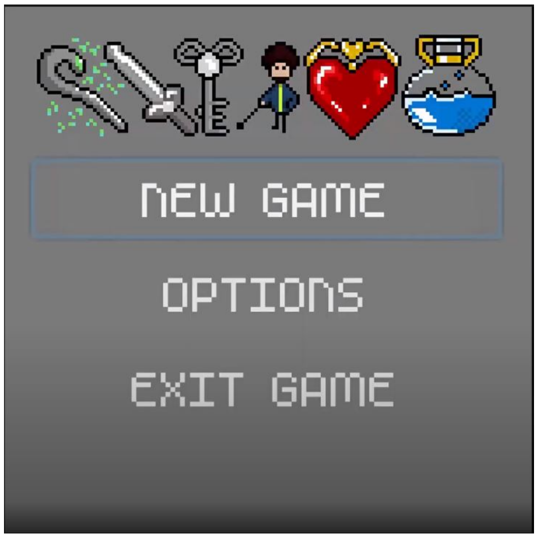
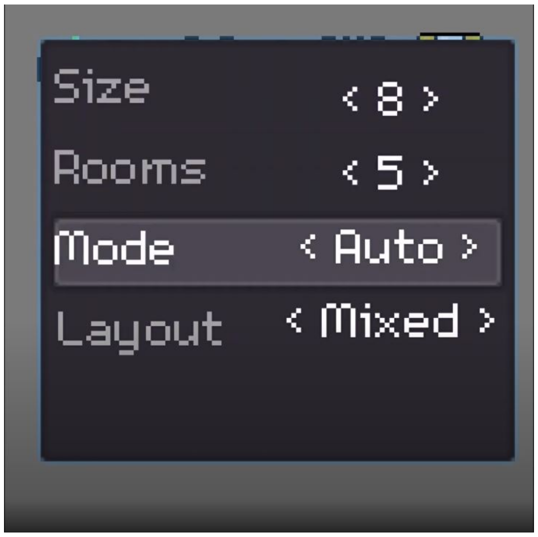


Figure A.19: Final Prototype Start Menu


Size < 8 >

Rooms < 5 >

Mode < Auto >

Layout < Mixed >

Figure A.20: Final Prototype Settings Menu

B Design

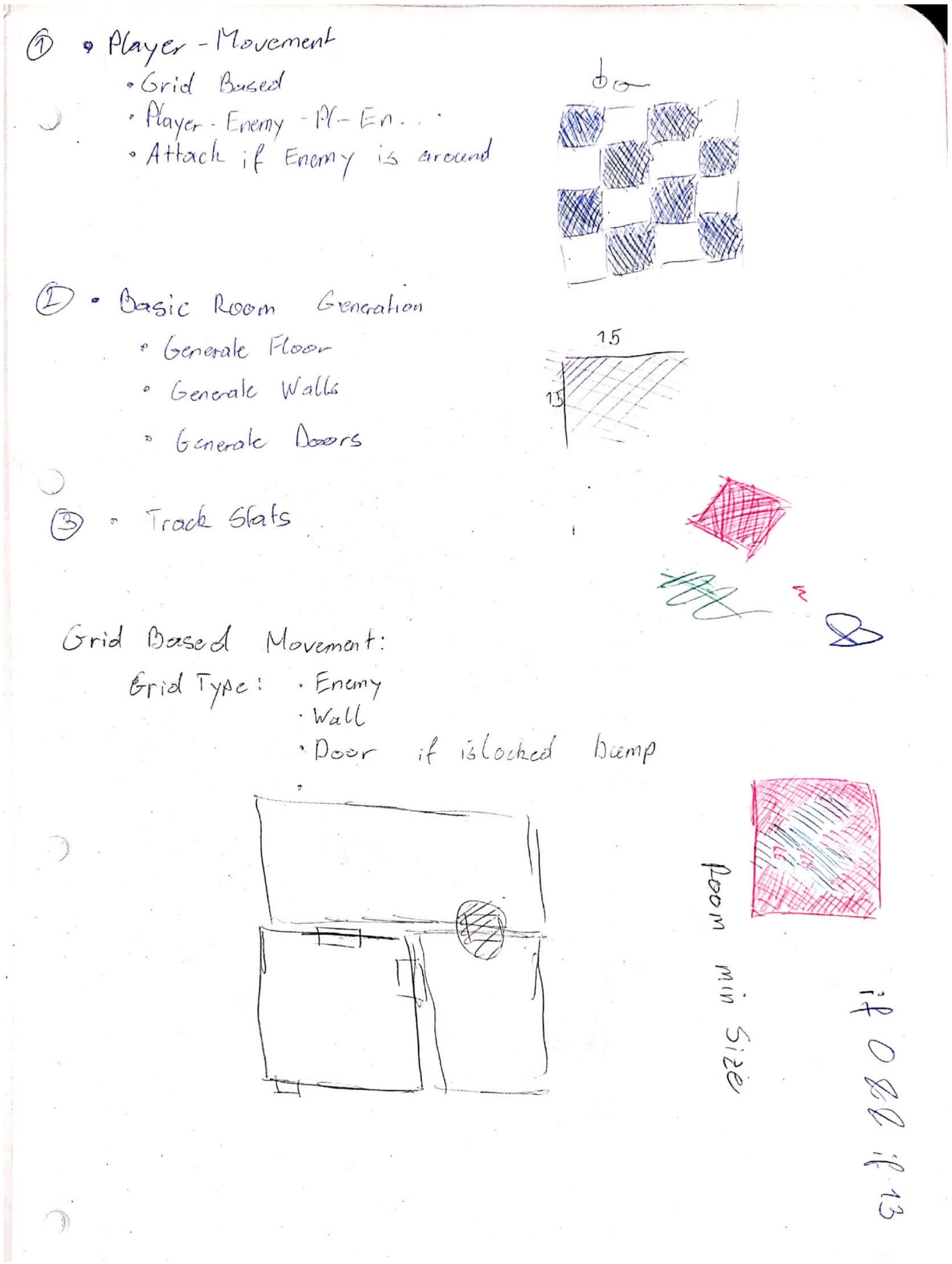
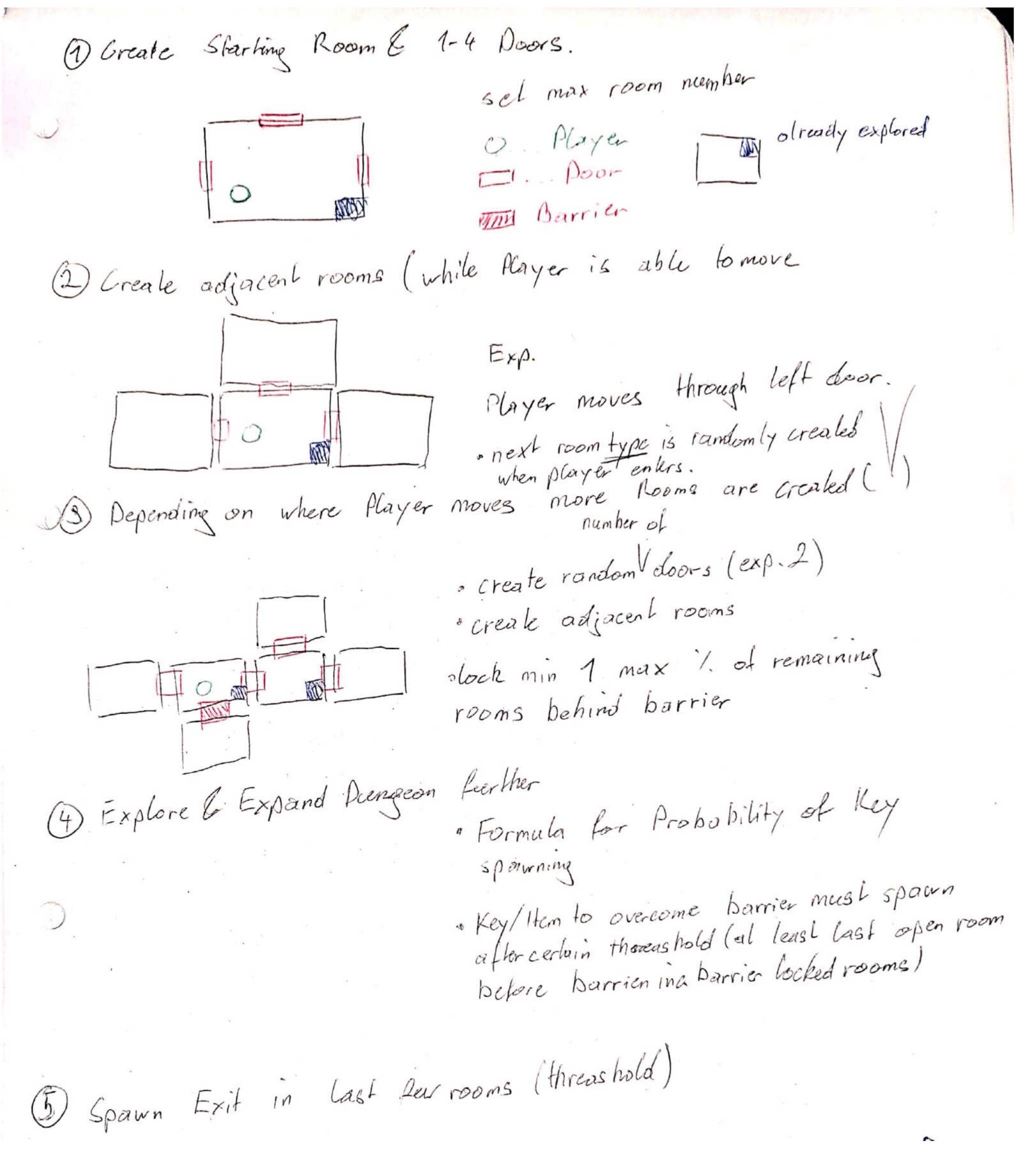


Figure B.1: Base Game Loop First Sketch



① Create Starting Room & 1-4 Doors.

set max room number

○ ... Player

▭ ... Door

▨ Barrier

already explored

② Create adjacent rooms (while Player is able to move

Exp.

Player moves through left door.

• next room type is randomly created when player enters. ✓

③ Depending on where Player moves more number of Rooms are created (✓)

• create random ✓ doors (exp. 2)

• create adjacent rooms

• lock min 1 max % of remaining rooms behind barrier

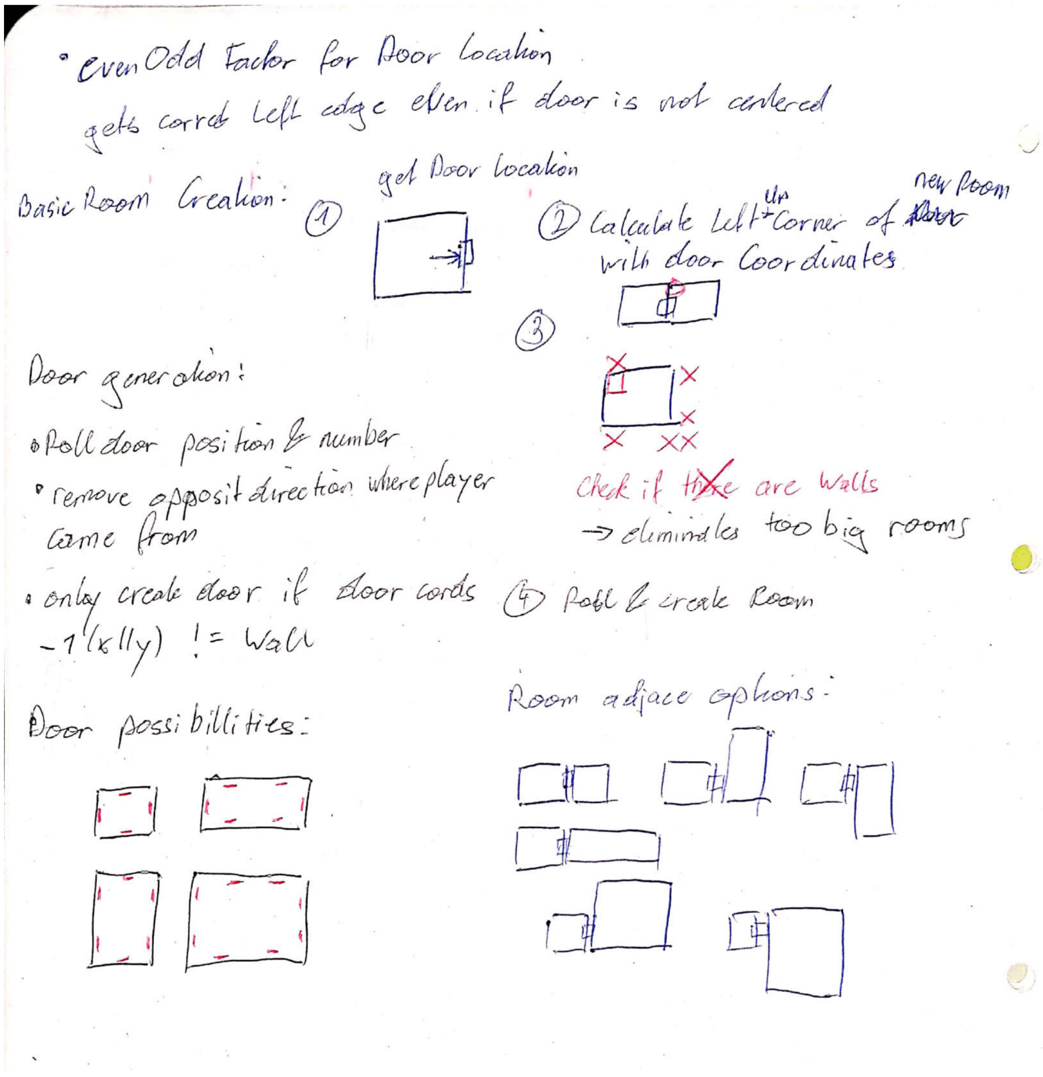
④ Explore & Expand Dungeon further

• Formula for Probability of Key spawning

• Key/Item to overcome barrier must spawn after certain threashold (at least last open room before barrien in a barrier locked rooms)

⑤ Spawn Exit in last few rooms (threashold)

Figure B.2: Map Generation Process First Design



• EvenOdd Factor for Door Location
gets correct Left edge even if door is not centered

Basic Room Creation: ① get Door location

② Calculate Left ^{Up} Corner of ~~Door~~ new Room with door Coordinates

③ Check if ~~these~~ are Walls
→ eliminates too big rooms

④ Roll & create Room

Door generation:
• Roll door position & number
• remove opposit direction where player came from
• only create door if door cords -1 (x||y) != Wall

Door possibillities:

Room adjace options:

Figure B.3: Door Position Design

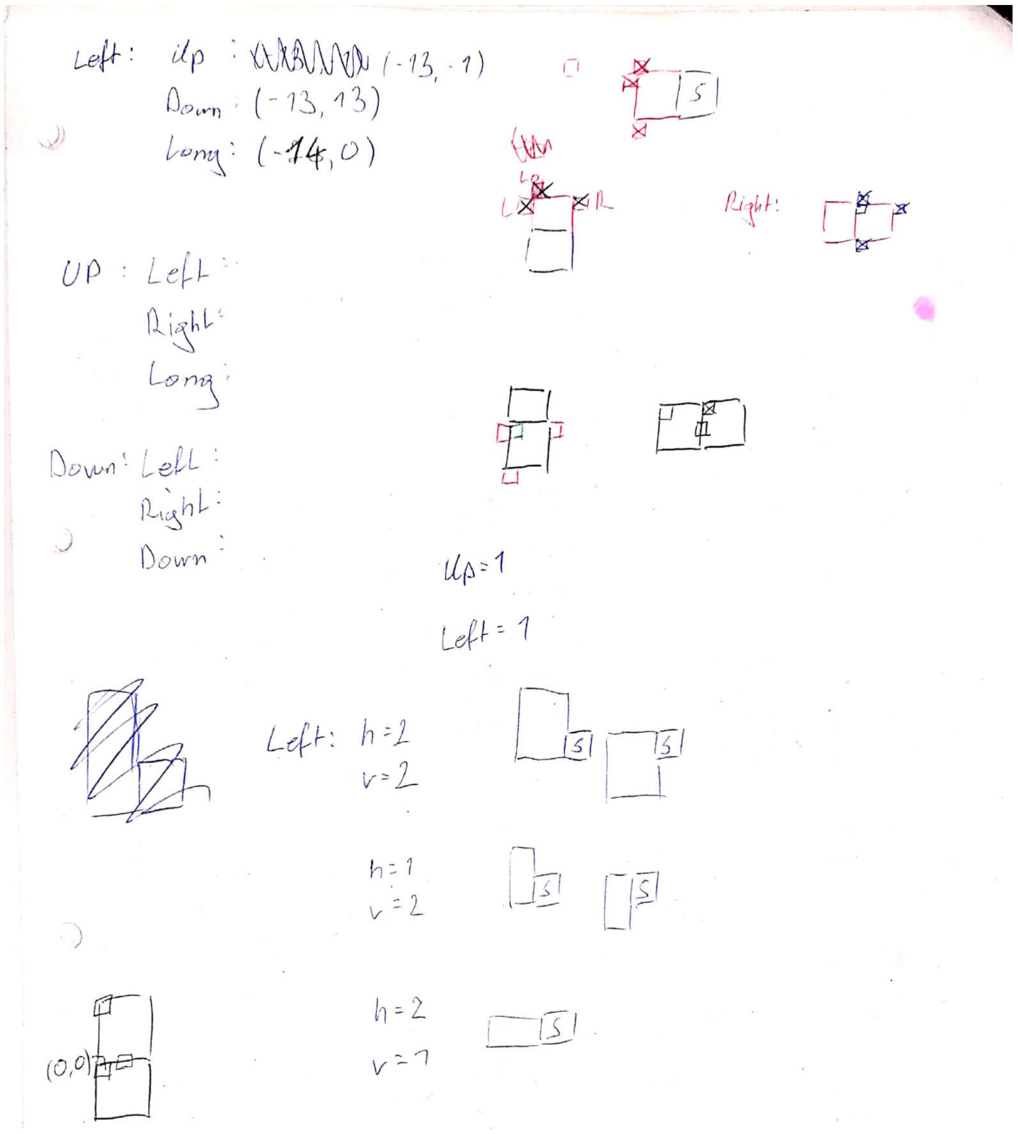
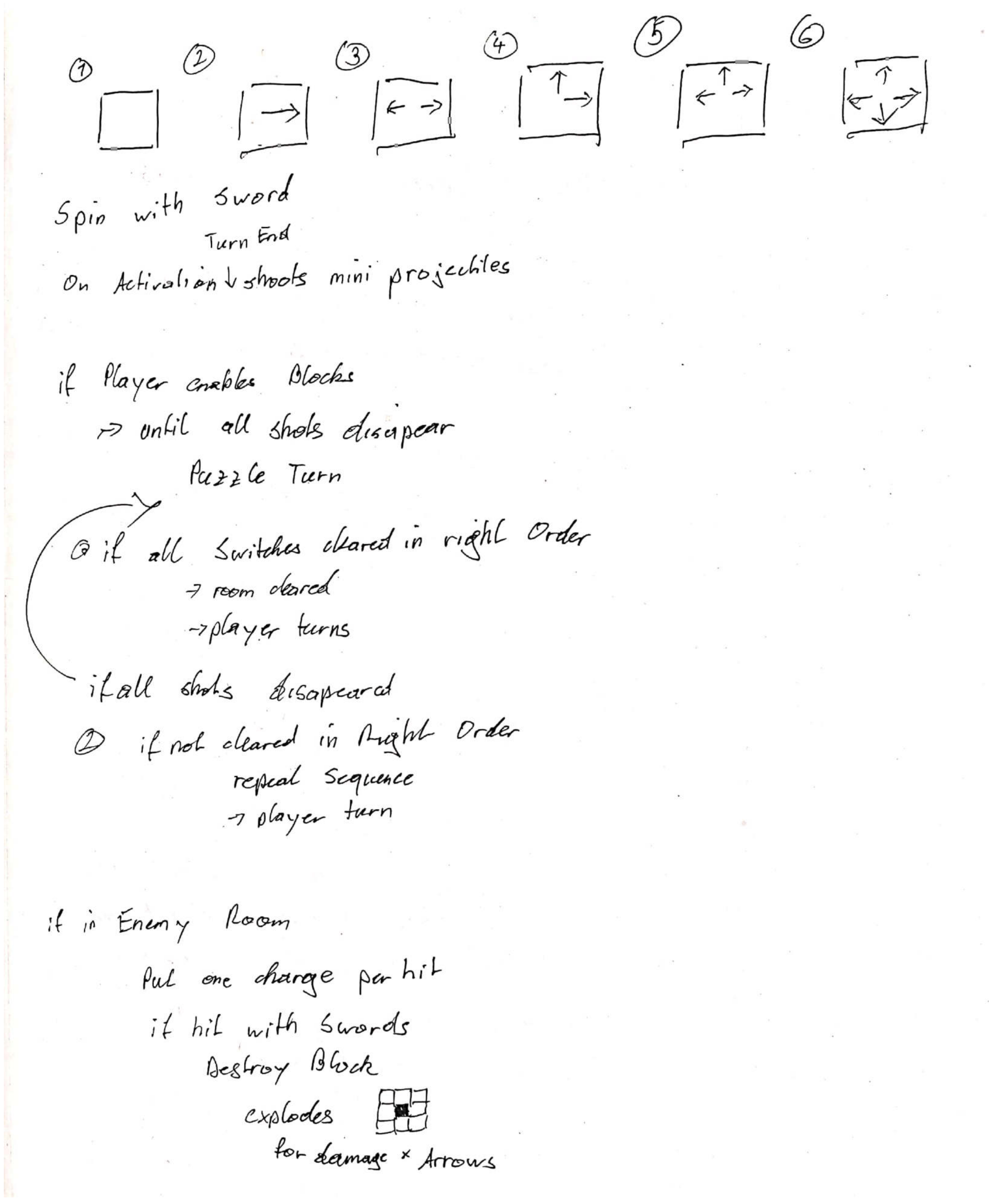


Figure B.4: Layout Options Position Design



Spin with Sword
Turn End
On Activation ↓ shoots mini projectiles

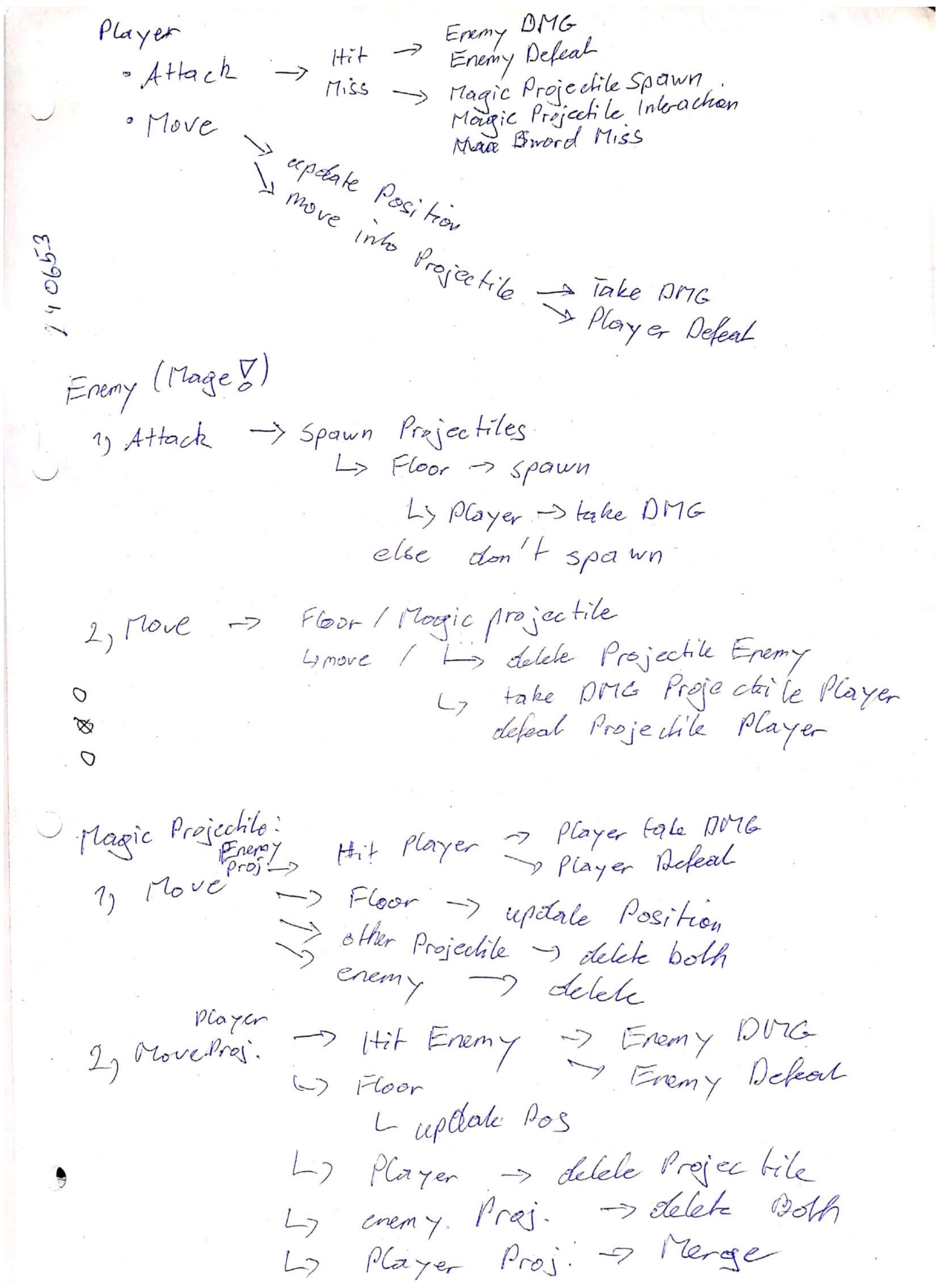
if Player enables Blocks
→ until all shots disapear
Puzzle Turn

① if all Switches cleared in right Order
→ room cleared
→ player turns

if all shots disapeared
② if not cleared in Right Order
repeat Sequence
→ player turn

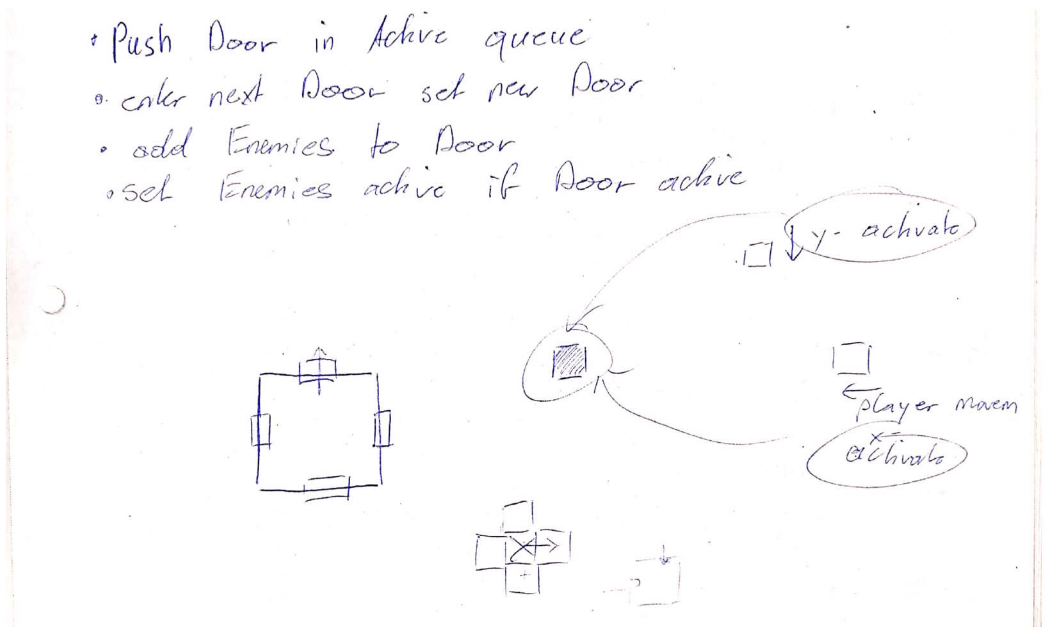
if in Enemy Room
Put one charge per hit
if hit with Swords
Destroy Block
explodes
for damage × Arrows

Figure B.5: First Power Block Interaction Design



```
Player
  • Attack  →  Hit  →  Enemy DMG
                        Enemy Defeat
               Miss →  Magic Projectile Spawn.
                        Magic Projectile Interaction
                        ~~Mace~~ Sword Miss
  • Move  → update Position
          ↳ move into Projectile → Take DMG
                                 → Player Defeat

140653

Enemy (Mage ▽o)
  1) Attack  → Spawn Projectiles
                 ↳ Floor → spawn
                     ↳ Player → take DMG
                   else don't spawn

  2) Move  →  Floor / Magic projectile
              ↳move /  ↳ delete Projectile Enemy
                       ↳ take DMG Projectile Player
                         defeat Projectile Player

Magic Projectile:
  Enemy Proj. →  Hit Player → Player take DMG
                            → Player Defeat
  1) Move  → Floor → update Position
           → other Projectile → delete both
           → enemy → delete

     Player
  2) Move Proj. → Hit Enemy → Enemy DMG
                            → Enemy Defeat
                ↳ Floor
                    ↳ update Pos
                ↳ Player → delete Projectile
                ↳ enemy. Proj. → delete Both
                ↳ Player Proj. → Merge
```

Figure B.6: Interactions Overview Design



- Push Door in Active queue
- enter next Door set new Door
- add Enemies to Door
- set Enemies active if Door active

Figure B.7: Player Moves Through Door Interaction Design

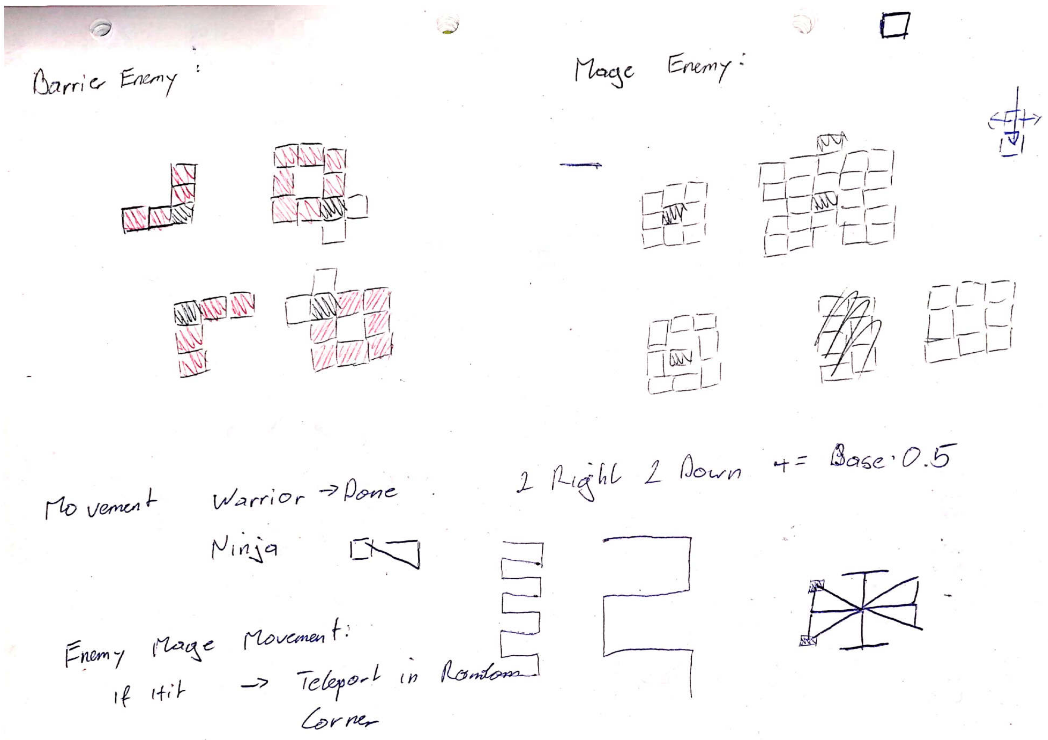


Figure B.8: Enemy Attack Pattern Sketch 1

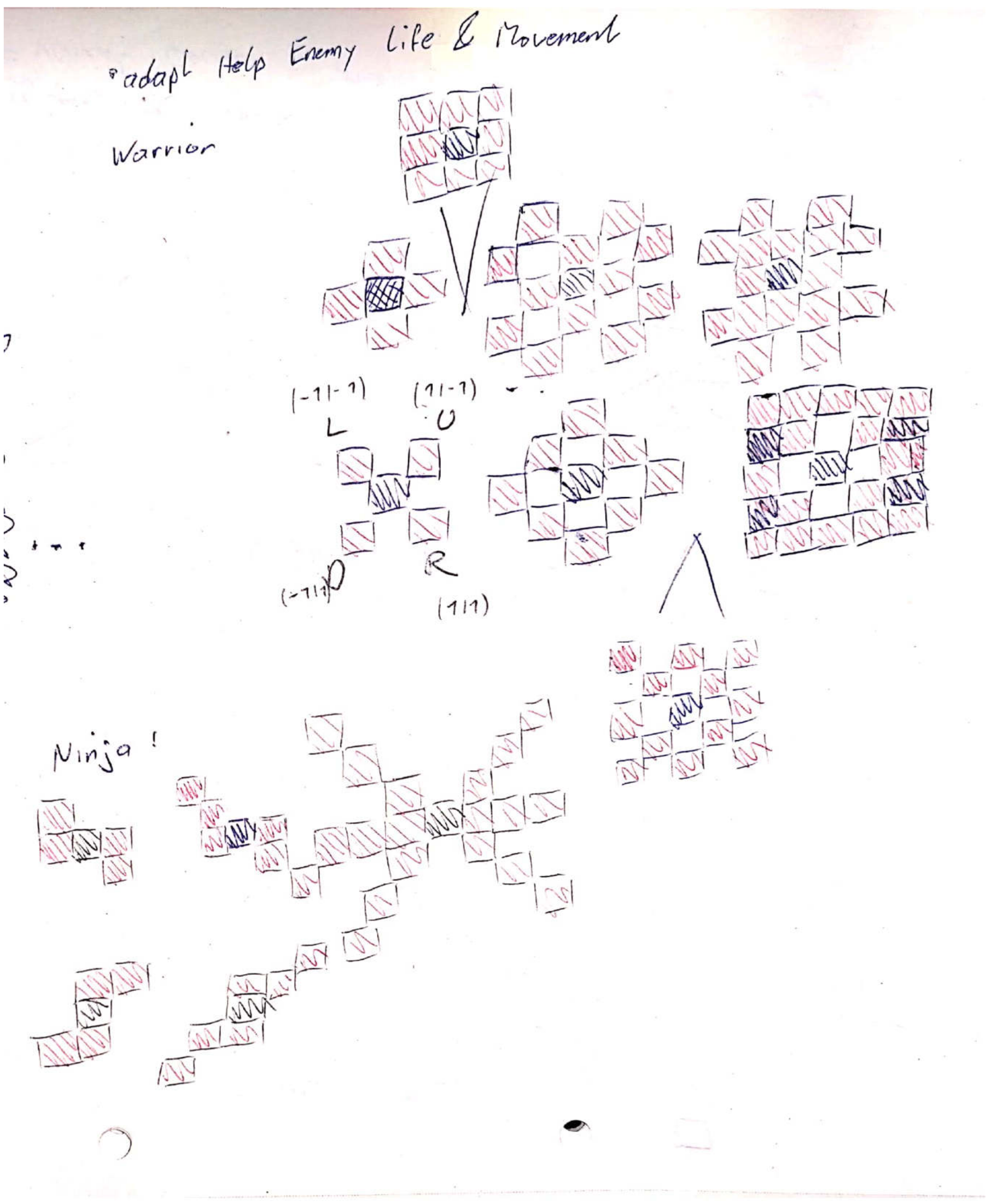
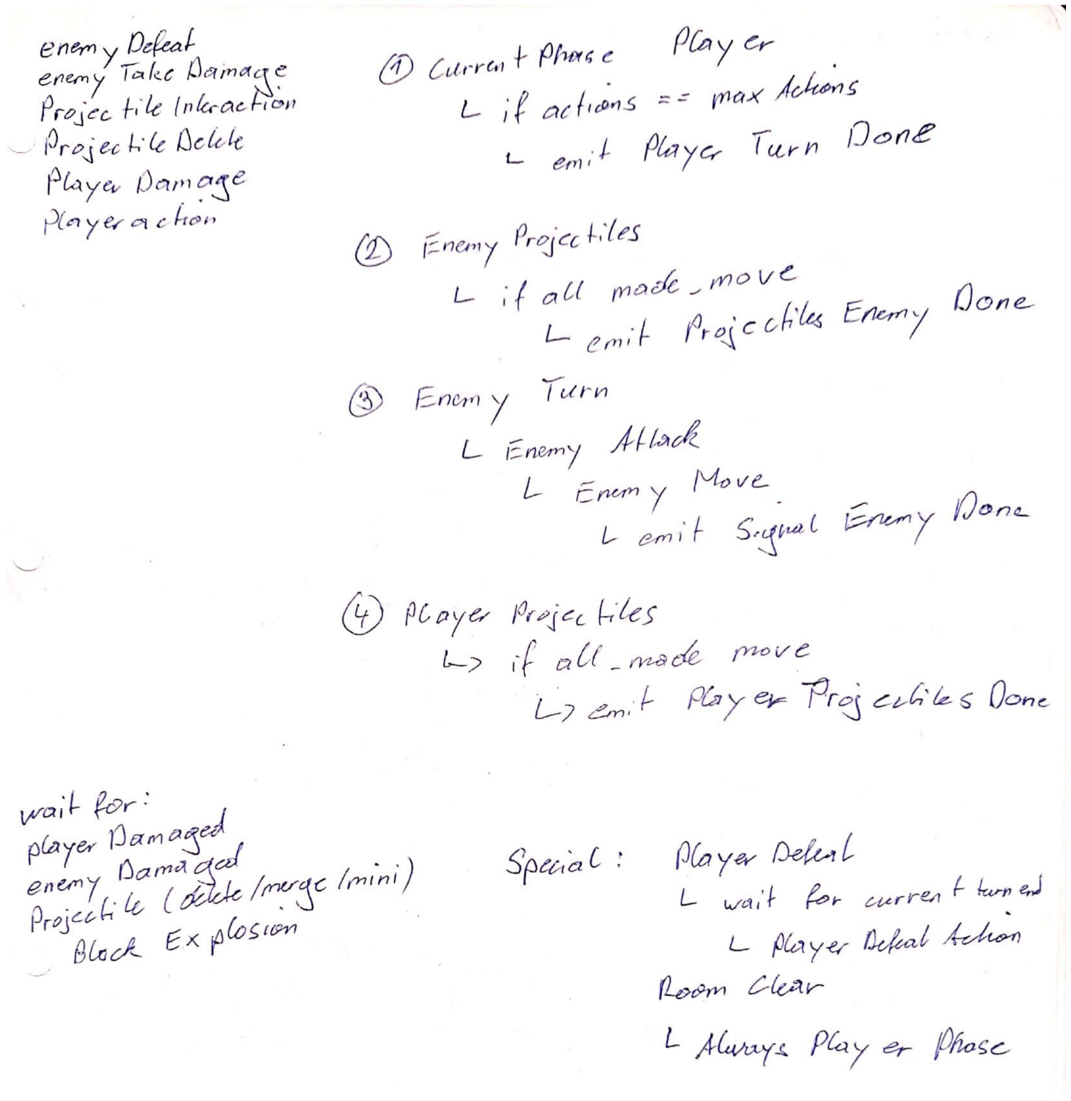


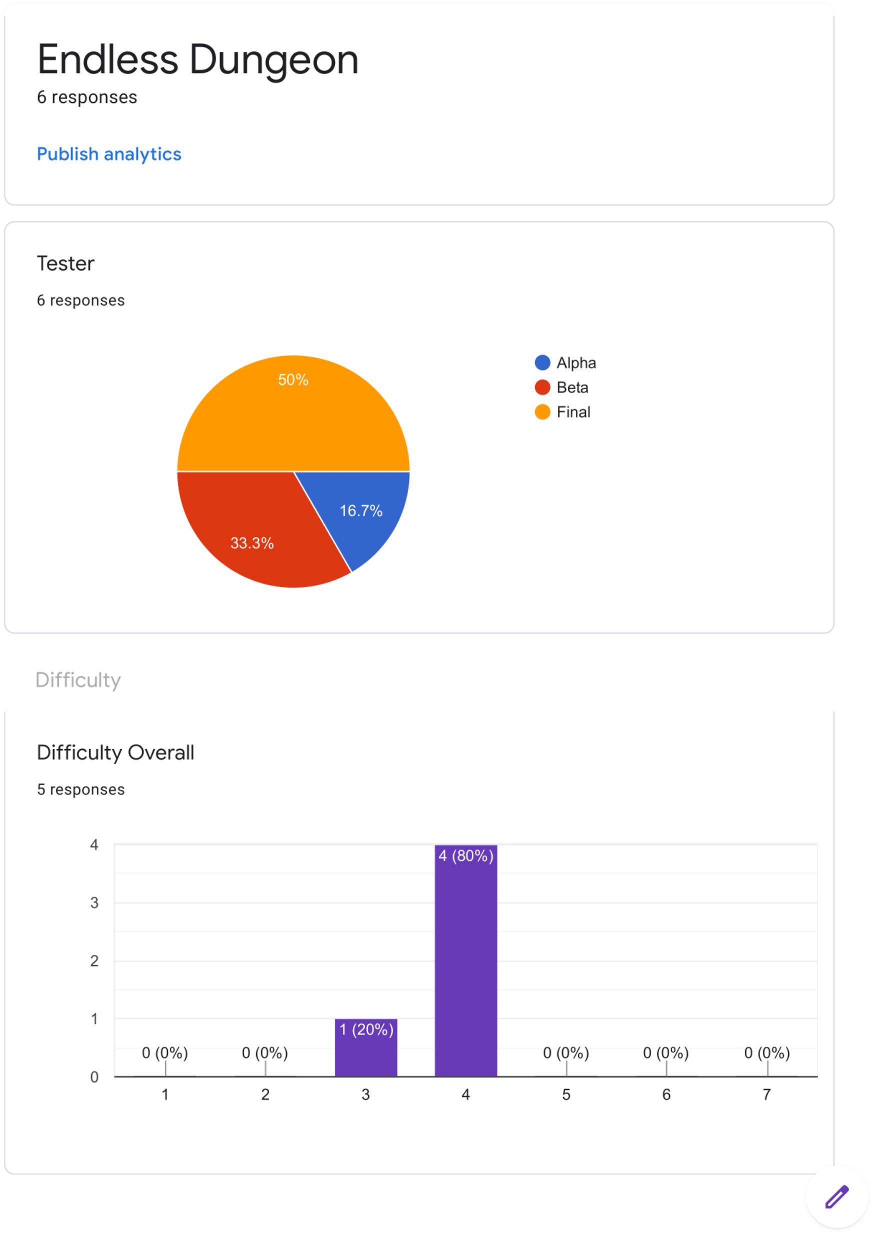
Figure B.9: Enemy Attack Pattern Sketch 2



```
enemy Defeat
enemy Take Damage
Projectile Interaction
Projectile Delete
Player Damage
Player action

(1) Current Phase    Player
      L if actions == max Actions
          L emit Player Turn Done

(2) Enemy Projectiles
      L if all made_move
          L emit Projectiles Enemy Done

(3) Enemy Turn
      L Enemy Attack
          L Enemy Move
              L emit Signal Enemy Done

(4) Player Projectiles
      L> if all_made move
          L> emit Player Projectiles Done

wait for:
player Damaged
enemy Damaged
Projectile (delete/merge/mini)
   Block Explosion

Special:   Player Defeat
             L wait for current turn end
                 L Player Defeat Action
           Room Clear
             L Always Player Phase
```

Figure B.10: Turn Controller Design

C Questionnaire

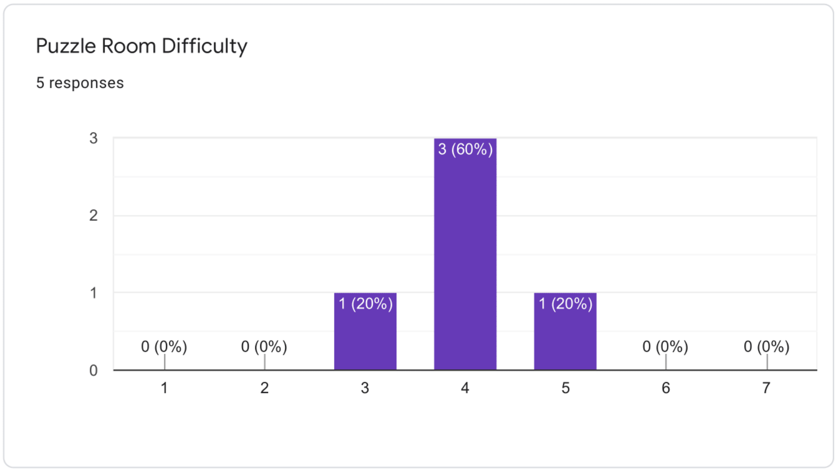

Endless Dungeon
6 responses
Publish analytics

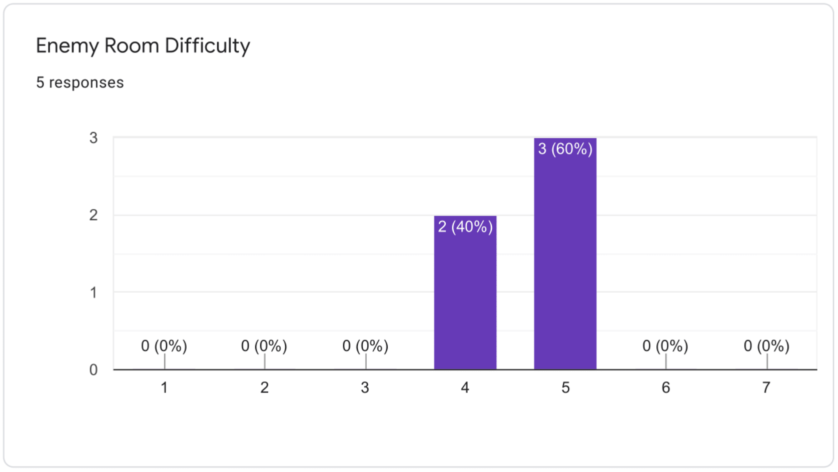
Tester
6 responses

Difficulty

Difficulty Overall
5 responses

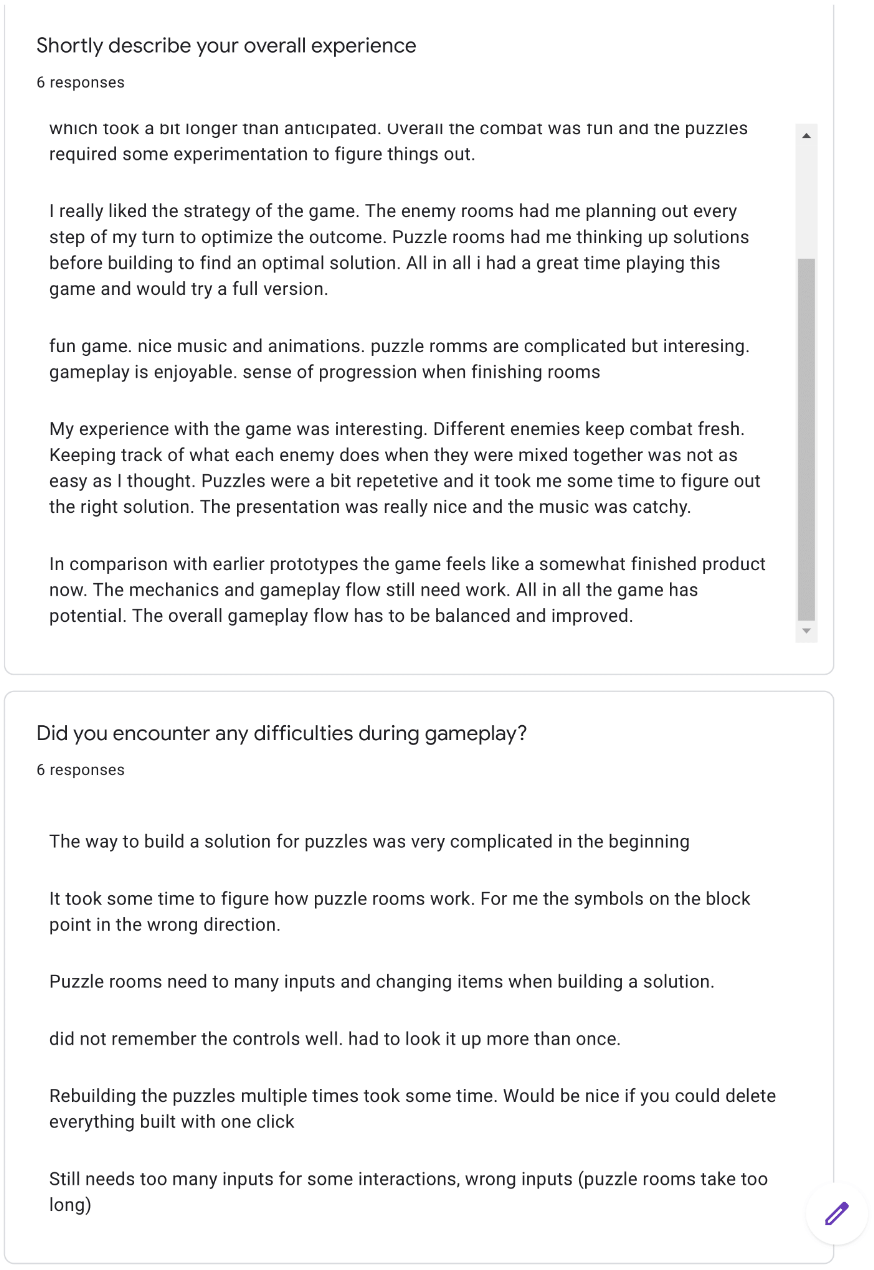
Figure C.1: Questionnaire Page 1





Experience

Figure C.2: Questionnaire Page 1



Shortly describe your overall experience

6 responses

which took a bit longer than anticipated. Overall the combat was fun and the puzzles required some experimentation to figure things out.

I really liked the strategy of the game. The enemy rooms had me planning out every step of my turn to optimize the outcome. Puzzle rooms had me thinking up solutions before building to find an optimal solution. All in all i had a great time playing this game and would try a full version.

fun game. nice music and animations. puzzle romms are complicated but interesing. gameplay is enjoyable. sense of progression when finishing rooms

My experience with the game was interesting. Different enemies keep combat fresh. Keeping track of what each enemy does when they were mixed together was not as easy as I thought. Puzzles were a bit repetetive and it took me some time to figure out the right solution. The presentation was really nice and the music was catchy.

In comparison with earlier prototypes the game feels like a somewhat finished product now. The mechanics and gameplay flow still need work. All in all the game has potential. The overall gameplay flow has to be balanced and improved.

Did you encounter any difficulties during gameplay?

6 responses

The way to build a solution for puzzles was very complicated in the beginning

It took some time to figure how puzzle rooms work. For me the symbols on the block point in the wrong direction.

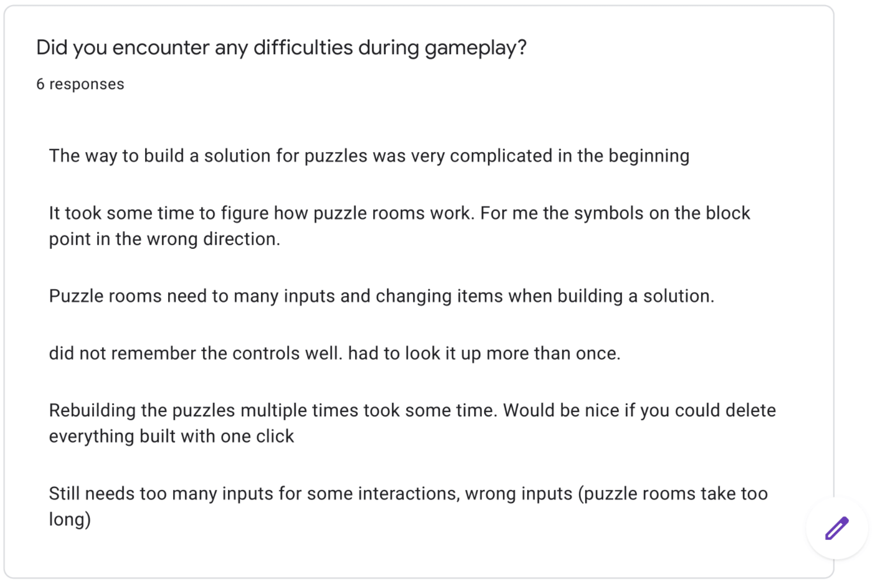
Puzzle rooms need to many inputs and changing items when building a solution.

did not remember the controls well. had to look it up more than once.

Rebuilding the puzzles multiple times took some time. Would be nice if you could delete everything built with one click

Still needs too many inputs for some interactions, wrong inputs (puzzle rooms take too long)

Figure C.3: Questionnaire Page 3



Did you encounter any difficulties during gameplay?

6 responses

The way to build a solution for puzzles was very complicated in the beginning

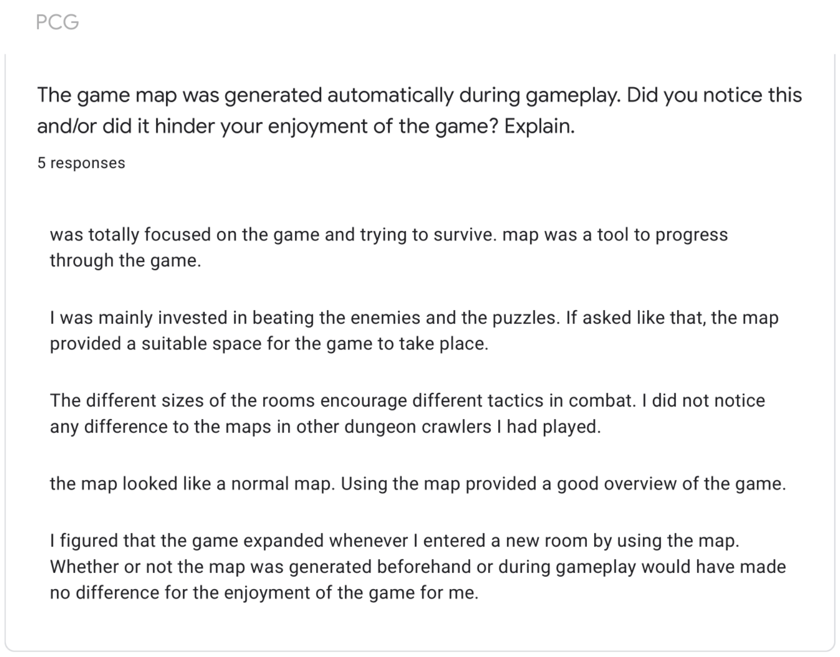
It took some time to figure how puzzle rooms work. For me the symbols on the block point in the wrong direction.

Puzzle rooms need to many inputs and changing items when building a solution.

did not remember the controls well. had to look it up more than once.

Rebuilding the puzzles multiple times took some time. Would be nice if you could delete everything built with one click

Still needs too many inputs for some interactions, wrong inputs (puzzle rooms take too long)



PCG

The game map was generated automatically during gameplay. Did you notice this and/or did it hinder your enjoyment of the game? Explain.

5 responses

was totally focused on the game and trying to survive. map was a tool to progress through the game.

I was mainly invested in beating the enemies and the puzzles. If asked like that, the map provided a suitable space for the game to take place.

The different sizes of the rooms encourage different tactics in combat. I did not notice any difference to the maps in other dungeon crawlers I had played.

the map looked like a normal map. Using the map provided a good overview of the game.

I figured that the game expanded whenever I entered a new room by using the map. Whether or not the map was generated beforehand or during gameplay would have made no difference for the enjoyment of the game for me.

Figure C.4: Questionnaire Page 3