

Unleveled



Contents

1. Introduction	4
1.1. Project Overview	4
1.1.1 Target Audience	4
1.2. Technical Description / Implementation	4
1.2.1. Core Loop & Gameplay	5
1.2.2. Development Tools	6
2. Art	7
2.1. Artistic Style	7
2.2. Game World	7
2.3. References	8
3. Game Design	9
3.1. Features	9
3.2. Unique Selling Points	9
3.2. Mechanics	9
3.3. Player	10
3.4. NPC's	10
3.4.1. Neutral Characters	10
3.4.2. Enemies	11
3.4.3. Special Enemies	12
3.4.4. Bosses	13
3.5. Progression and Quests	13
3.6. Items	13
3.8. Puzzles	19
3.9. Player Control and Camera	19
3.10. Key Design References	20
4. Level Design	22
4.1. Game World Design	22
4.2. Dungeon Design	23
4.2.1. Initial Dungeon Design	23
4.2.2. Drunkard's Walk Algorithm	23
4.2.3. Loot and Enemy Generation	24
4.2.4. Safe Zone	26
4.2.5. Boss Zone	27
5. UI	28
5.1. UI Distribution	28
5.1.1. Main Menu:	28

5.1.2. Pause Menu:	29
5.1.3. Character Page:	29
5.1.4. Inventory:	30
5.1.5. Bottom Bars:	30
5.1.6. Recipe Menu:	30
5.2. Concepts	31
5.3. Text	31
6. Sound	32
6.1. Music References	32
6.2. Where to place the sounds	32
7. Save game	32
8. Cut out Features	33
8.1. Art	33
8.2. Quests	33
8.3. NPCs	33
8.4. Items	33
8.5. Dungeon	33
8.6. Tutorial	34
8.7. Extras	34
8.8. Sound	34
8.9. Lore	34
8.10. Save game	35
9. Bibliography	35

1. Introduction

1.1. Project Overview

In **Unleveled**, you control *one character (name to be decided)* that finds itself stranded in a beach. As the player explores the closest village, it finds out that many of its habitants are artificially deeply asleep. The only initially awaken NPC's is a fairy that explains the player that a **series of curses** have been placed in this village, and that even though it (the fairy) has been able to remove it from some people, the remaining ones are far more powerful than expected and can't be removed by *normal* meanings.

The **core** of these **curses** remain in the **basement** of one house (**blue house**), and the player is the only one with enough willpower to investigate it from its inside ([see core loop](#)).

When the player defeats a **new** boss from a **new** layer, a person in the village will be freed from its curse. Once all people have been freed from the village, the **cursing entity** will then become more powerful as it will **present itself** to the player and place a very **big** curse in the entire village. The dungeon also evolves to be more difficult. This **big curse** is removed periodically as the player progresses through the dungeon and its layers. When this curse is removed, a **final battle** will occur, which will lead to the end of the game.

(See [lore](#) and [dungeon](#) in cutout features)

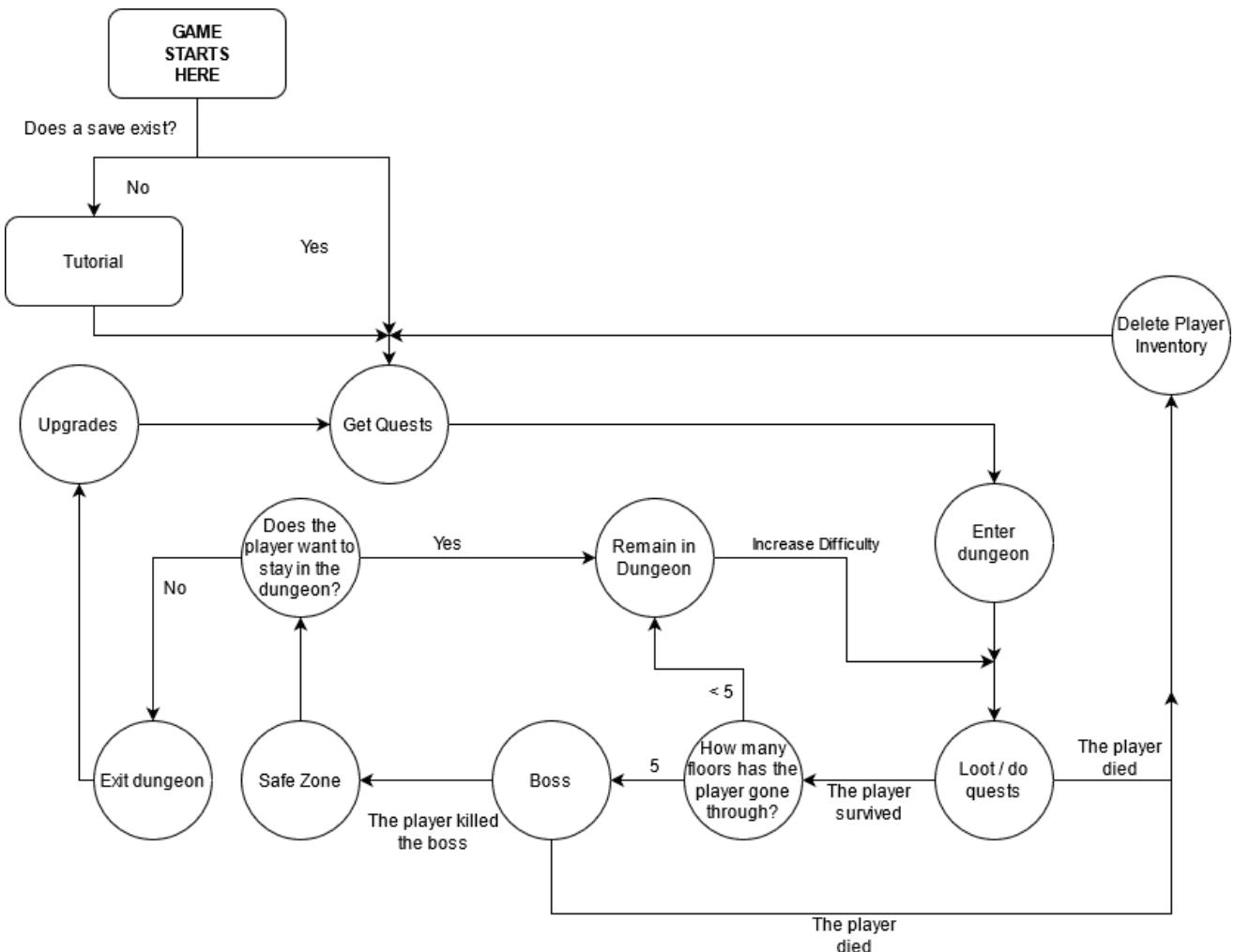
1.1.1 Target Audience

Our main audience for our game is young people that enjoy casual and visually attractive games with an added spice of difficulty. Being a procedurally generated dungeon crawler, our game will most likely attract attention from people that also play or have played other dungeon crawlers and want it to be a bit more difficult as well as rewarding than the other games.

1.2. Technical Description / Implementation

Our game will be a **hardcore dungeon crawler based rpg**, randomly generated levels and rpg-like features that will target people that like procedurally generated games with an added spice of difficulty. The game invites the player to progress through the world by letting it know the perils of the progressively more difficult dungeon as well as its treasures.

1.2.1. Core Loop & Gameplay



The player starts in a crossroads zone where it has two possible ways, up or right. The path above it has two houses, one is the entrance to the tutorial and the other one the exit point.

In the tutorial zone there are mainly three zones, in each one a core mechanic is explained. The first one is about basic combat, the second one is about throwing weapons and the third one covers basic crafting.

The path to the right, on the other hand, leads the player to the main village, where all houses but one are closed (the one that belongs to the player).

Inside the house the player can find a **Crafting Station**, an **Anvil**, a staircase to the basement, a permanent **Chest** to store things as well as the **Save Game** interactable.

The basement contains an entrance to the **Dungeon**, where the player can progress through the game. See [dungeon design](#) for an explanation of each of its elements.

1.2.2. Development Tools

For the development of this project, we needed to choose the specific technologies that we would use in terms of development. Based on experience and convenience, we decided that Unreal Engine, Unity, DXT (DirectX Toolkit) and a custom renderer were the options to consider.

First we considered using Unity because of how easy it is to use, but we quickly discarded it because it doesn't allow us to embed C++ code. Both the DirectX toolkit and the custom renderer would've made the development very exciting but far more painful in terms of systems development and debugging because of the amount of things we would've been required to do, which is why we discarded both of them.

In the end, we decided to go with Unreal Engine in its 4.24.3 version because of how used all the members of the group are to it, the fact that it uses C++, and the amount of features that it includes out of the box in terms of systems, even if we don't have full control over them.

We decided to develop our game for Microsoft Windows because all the members of the group have worked on Windows applications and we thought that starting the development on a new platform might have introduced unexpected problems that might have taken us more time than expected.

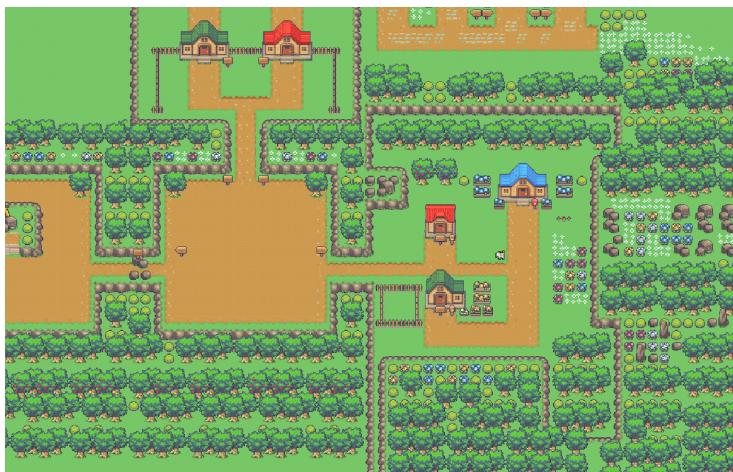
2. Art

2.1. Artistic Style

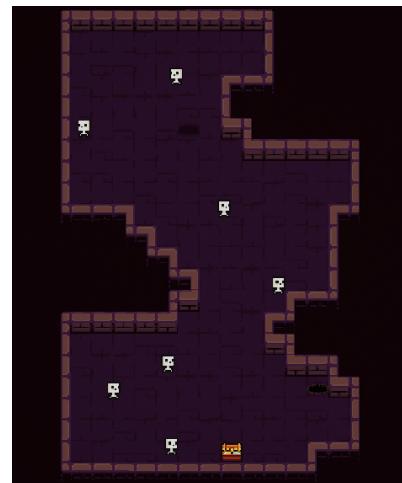
The game has a 16x16 pixel art style with minimum details. It uses bright and colourful colors for places with no danger and darker colors to more aggressive zones. This style was chosen because we want to relate to older generations and create the feeling of nostalgia to the player. Another reason why we chose this artistic style is because it is easier for us, who are not artists, to create some pieces of game art for our game.

2.2. Game World

We intended to create an **Overworld** that encourages the player to explore it, with different small zones that unlock different things.



Overworld



Example of Dungeon floor

We (internally) divided the game into two subworlds: the **Overworld** and the **Dungeon**. Depending on the current state of the game, different music will be played in the Overworld (see [project overview](#)) (exploring music, danger music) (see [sound](#) in cut out features). In the **Dungeon**, different music (see [sound](#) in cut out features) will be played and different colors will be used, depending on its **depth** (see [dungeon](#) in cut out features)

2.3. References

- [Moonlighter](#): Serving as one of the main references for our game, we share many of the features that Moonlighter includes, the main one being a rogue-like RPG with procedural dungeon generation.



- [Enter the Gungeon](#): Following the dungeon crawler path, Enter the Gungeon served as a baseline for how arcade we wanted the game to be in terms of amount of enemies as well as being *run-based* or not.



- [Stardew Valley](#): Mostly regarding art style.



3. Game Design

3.1. Features

- Procedurally generated dungeon crawler
- Questing system where the player must accomplish a series of objectives to progress through the game (see [quests](#) in cut out features).
- Crafting System which allows the player to create items with materials
- Player progression through rpg-like features
 - Equipment
 - Special skills
- Village / Overworld where the player can interact with non hostile NPCs (see [NPCs](#) in cut out features).
- Harder enemies as the player progresses through the dungeon

3.2. Unique Selling Points

- Not entirely run-based: the player shall use its own equipment to gather more things in the dungeon.
- Dying removes all the items from the player's inventory, including everything equipped.
- Layered dungeon depth: when the player reaches a certain level of one layer, it will transition to another one, which will have different traits such as different enemies, different environments and different objects (see [dungeon](#) in cut out features).

3.2. Mechanics

Aggressive:

- Weapon Attack
 - Pokes with the weapon in a range and causes damage to enemies.
- Weapon Throw
 - Throw the weapon, dealing damage to all enemies that get hit..
- Skill (Rune) system
 - There are three types of runes or skills
 - Fireball: Projectile which inflicts damage to enemies
 - Add Stats: For a period of time, the player gains some stats temporarily.
 - Time Stats: For a couple of seconds the player gains stats permanently.

Passive

- Equipment
 - The player can equip items to gain skills and stats.
- Crafting
 - Allows the player to craft items to improve their equipment and progress throughout the game.
 - In order to craft something the player shall first get its **recipe**.

3.3. Player

- Progress through the dungeon to acquire loot and do quests (see [quests](#) in cut out features).
- Acquire enough materials to upgrade equipment.
 - Materials are used to craft weapons and equipment.
- Actions:
 - Use items: certain items can be used to receive something (potions gives health, gear gives stats).
 - Attack with weapon: Pokes with the weapon in a range and causes damage to enemies.
 - Throw weapon: Unequips the weapon and throws it forward at a certain distance.
 - Attack with skills (Runes): Some items have skills (Runes) which can throw fireballs to enemies.
 - Store items in its inventory.
 - Craft objects: The player can craft items in the crafting table if it has the materials needed. These items shall be used to further progress in the dungeon which will make it progressively harder.

3.4. NPC's

- As the dungeon progresses (levels), enemies get harder by having more health points and dealing more damage (see [dungeon](#) in cut out features).
- Chance to drop items from a pool of objects on death so that the player can craft better items and therefore dive deeper in the dungeon and progress further. (see [extras](#) in cut out features):
 - Materials - Used by the player for crafting better items or complete quests
 - Recipes - New items for the player to craft
 - Gold - Currency used in the Overworld shop

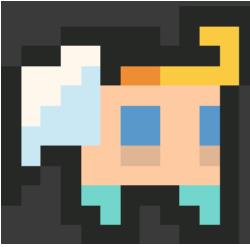
By having the **enemies** also drop objects we add another way of getting objects to improve the player's gameplay (see [dungeon](#) in cut out features).

- Each npc has a special role in the overworld, which affects the way they interact with the player. While some npc's might only give you quests, others might only sell items (see [NPCs](#) in cut out features).

3.4.1. Neutral Characters

As we mentioned before, there are different types of NPCs:

- Vendor NPCs - These ones act like a shop for the player where he can buy or sell anything that he currently has.
- Quest NPCs - These ones will hand in quests for the player to complete. Some of them will not have the quest right away because the player may need to complete some previous quests before being eligible to get the next one.
- Information NPCs - These ones will just have some dialog for the player to read. Some of them will have important information that could help the player in some way, ranging from where to find some special loot or a key way to kill a certain enemy.
- Item giver NPCs - These one will just give the player some items along with some sort dialog.

Current NPCs			
Image	Name	Type	Description
	Fairy	Item giver	Gives the player a Knife after telling him that she is there to help him progress through his house basement. (which is the actual dungeon)

3.4.2. Enemies

We wanted to have variety in our enemies so we decided to create different behaviours to create diversity and more interesting dungeons and combat.

There are different types of enemies:

- **Melee:** This type tries to get close to the player to cause damage.
- **Range:** Range enemies have a list of patterns that it has to make. A pattern establishes how the enemy is going to shoot projectiles.

Image	Name	Type	Description	Stats
	Follow	Melee	Will follow the player and damage him when close enough.	Health: 10 Damage: 5 Movement Speed: 75
	Fast Follow	Melee	Will follow the player at a faster rate than the Follow enemy, but has lower health.	Health: 1 Damage: 3 Movement Speed: 245
	Rush	Melee	Goes towards the player until is close enough and then charges a rush attack to rapidly close any position gaps	Health: 10 Damage: 5 Movement Speed: 125

	Range	Ranged	When the player is close enough, will launch a projectile towards the player.	Health: 15 Damage: 2 x1 attack Movement Speed: 100 Projectile Movement Speed: 400
---	-------	--------	---	--

3.4.3. Special Enemies

We have some special enemies that are more complex than the rest. For this special enemies we added a new type that uses weapons:

- **Weapon** (Melee and Range) : This enemy carries a weapon that is used to hit the player if it is close enough and has a chance of throwing it if the player is in a certain range.

The multirange enemy is just a ranged enemy with different patterns.

Image	Name	Type	Description	Stats
	Armored	Weapon Melee	Capable of using any weapon he finds. This enemy will attack the player with the current weapon he has equipped and sometimes he can throw the weapon to the player.	Health: 20 Damage: Based on current weapon Movement Speed: 100
	Multi Range	Ranged	When the player is close enough, will launch multiple projectiles towards all 4 axis	Health: 15 Damage: 3 x4 attacks Movement Speed: 100 Projectile Movement Speed: 400

Behaviour component

- The Behaviour Component manages how the AI is going to react to the player actions and deliver a credible response.
- Types of AI:
 - Wander: Wanders around a specific area.
 - All enemies wander if the player is not in range.
 - Follow: Follow a specific target.
 - Rush: Rushes to the target.
 - Range: Fires projectiles following a pattern or following a target.
 - Weapon: Picks up a weapon and attacks the player when it is close. There is a chance to throw the weapon if the player is in a certain range.

3.4.4. Bosses

For the boss, we wanted him to have different behaviours and stages, so we added a behaviour list that decides which behaviour to use with the health it has. This boss can recreate any behavior of any enemy explained before.

Image	Name	Description	Stats
	Apple King	Capable of using any behaviour. The behaviours are set in a list.	Health: 200 Damage: 4 Movement Speed: 100

3.5. Progression and Quests

- The player does not have stats per-se, equipment is what gives it progression, and progressing in equipment will affect the capabilities of the player in terms of output damage and resistance to it.
- As the player progresses through the levels, the enemies will become harder: faster, more damage and harder to kill (see [dungeon](#) in cut out features).
- Quests that progressively challenge the player, *forcing* him to go to harder levels as the game progresses (see [quests](#) in cut out features):
- Different types:
 - Retrieve an object that only appears within a level
 - Kill a certain amount of a given enemy
 - Defeat an specific boss
 - Explore an area within a level
 - Hand in a given object to another NPC
 - Find certain NPCs in the dungeon and save them to unlock different features in the Overworld

3.6. Items

- Each equipable piece has **at least** 4 rarity categories, which are common, rare, legendary or unique (**fixed spawns for unique items**, a quest needs to be completed to upgrade them (see [items](#) in cut out features)). Every equipable item contributes to the player stats in a different way depending on its type:
 - Weapon increase damage output
 - Armor equipment decrease damage input and/or increase movement speed
- 4 types of **weapons**:
 - Rapier (fast attacking, low damage one handed sword),

- Sword and shield (allow some protection with medium damage),
- Two handed sword (slow attacking very high damage)
- Bow (medium damage, ranged attacks).
- **Armor equipment** (see [items](#) in cut out features):
 - boots: increases movement speed.
 - legs: increases armor (medium).
 - thorax: increases armor (high).
 - arms: increases armor (low).
 - head: increases armor (medium).
- Each piece of equipment has a probability of spawning with **slots**, which will allow the player to place **runes** that slightly enhance the stats. probability increases with rarity (common items have 0%), up to **2** slots per item (see [items](#) in cut out features).
- Each piece of equipment can be **upgraded** up to three times to increase the stats that the player received when equipping it. (blacksmith (see [NPCs](#) in cut out features)).
- The player can effectively improve a piece of equipment by placing a **rune** in **slots** (if it has it, slots are based on probability) (see [items](#) in cut out features).
- Projectile type: throws a projectile which damages the enemies.
- Objects required for certain quests such as a letter for another NPC or some materials that the blacksmith needs. (see [quests](#) in cut out features).
- Crafting materials (to create equipment, cosmetics).
- Potions and throwables (see [items](#) in cut out features).

Consumables		
Common	Potion	Heals the player for 2
Rare	Potion	Heals the player for 4
Legendary	Potion	Heals the player for 8

Materials		
Common	Stone	Used for crafting
Common	Blue Shards	Used for crafting
Rare	Silver Ingot	Used for crafting
Rare	Red Gem	Used for crafting
Rare	Green Gem	Used for crafting
Legendary	Blue Gem	Used for crafting
Legendary	Gold Ingot	Used for crafting

Recipes		
Common	Wood Spear	Stick x1 Knife x1 Stone x5
Rare	Archer helmet	Knife x1 Ginger Helmet x1
Rare	Knight Helmet	Purple Helmet x1 Red Gem x2 Green Gem x2 Silver Ingot x2
Rare	Pirate Helmet	Knife x1 Ginger Helmet x1
Rare	Princess Helmet	Pumpkin Helmet x1 Blue Shards x5
Rare	Silver Boots	Basic Boots x1 Silver Ingot x10
Rare	Rusty Sword	Knife x1 Stick x1 Silver Ingot x5
Rare	Sword	Rust Sword x1 Blue Shards x10
Rare	Silver Sword	Sword x1 Silver Ingot x10
Legendary	Doctor Helmet	Ginger Helmet x1 Common Potion x10 Rare Potion x5 Legendary Potion x2
Legendary	Gold Helmet	Knight Helmet x1 Gold Ingot x10
Legendary	Mage Helmet	Basic Mage Helmet x1 Red Gem x5 Green Gem x5 Blue Gem x3
Legendary	Gold Boots	Silver Boots x1 Gold Ingot x10
Legendary	Gold Sword	Silver Sword x1 Gold Ingot x10

Weapons		
Common	Stick	Stats: Attack: 1 Range: 0.50 Attack rate: 4.0 Skill: None
Common	Knife	Stats: Attack: 2 Range: 0.75 Attack rate: 4.0 Skill: None
Common	Wood Spear	Stats: Attack: 3 Range: 3.00 Attack rate: 1.0 Skill: None
Common	Rusty Sword	Stats: Attack: 3 Range: 1.00 Attack rate: 4.0 Skill: None
Common	Sword	Stats: Attack: 5 Range: 1.00 Attack rate: 4.0 Skill: None
Rare	Silver Sword	Stats: Attack: 6 Range: 3.00 Attack rate: 4.0 Skill: None
Legendary	Gold Sword	Stats: Attack: 10 Range: 2.00 Attack rate: 4.0 Skill: Fireball

Gear

Common	Ginger Helmet	Stats: HP: 0 Armor: 5 Attack: 0 Movement Speed: 0 Skill: None
Common	Pumpkin Helmet	Stats: HP: 2 Armor: 3 Attack: 0 Movement Speed: 0 Skill: None
Common	Purple Helmet	Stats: HP: 0 Armor: 5 Attack: 0 Movement Speed: 0 Skill: None
Common	Basic Mage Helmet	Stats: HP: 1 Armor: 1 Attack: 0 Movement Speed: 0 Skill: None
Common	Basic Boots	Stats: HP: 0 Armor: 0 Attack: 0 Movement Speed: 25 Skill: None
Rare	Archer Helmet	Stats: HP: 0 Armor: 5 Attack: 2 Movement Speed: 0 Skill: None
Rare	Knight Helmet	Stats: HP: 0 Armor: 10 Attack: 0 Movement Speed: 0 Skill: None

Rare	Pirate Helmet	Stats: HP: 5 Armor: 0 Attack: 2 Movement Speed: 0 Skill: None
Rare	Princess Helmet	Stats: HP: 5 Armor: 5 Attack: 0 Movement Speed: 0 Skill: None
Rare	Silver Boots	Stats: HP: 0 Armor: 0 Attack: 0 Movement Speed: 50 Skill: None
Legendary	Doctor Helmet	Stats: HP: 0 Armor: 7 Attack: 0 Movement Speed: 0 Skill: Health per second
Legendary	Gold Helmet	Stats: HP: 0 Armor: 10 Attack: 0 Movement Speed: 0 Skill: Increases Attack by 5
Legendary	Mage Helmet	Stats: HP: 4 Armor: 4 Attack: 0 Movement Speed: 0 Skill: Fireball
Legendary	Gold Boots	Stats: HP: 0 Armor: 0 Attack: 0 Movement Speed: 75 Skill: None

3.8. Puzzles

- At the start of the game, the player will have its weapon of choice (rapier, sword and shield, two handed sword or bow), and the rest of the basic weapons can be found through the overworld, it will have small hints that will lead to their position. **
- **Leveled Sword:** extremely powerful sword only achievable by progressing through the game. (direct reference to [The Legend of Zelda: Breath of the Wild](#)) (see [extras](#) in cut out features)
 - When trying to retrieve the sword, the player will suffer damage for as long as it is trying to extract it from the stone.
 - It will require the player to have **at least** 100 health.
 - This is the only item in the game that can get its stats improved by killing enemies as well as runes and blacksmith.
 - This is the only item that will not be destroyed if the player dies while having it in his inventory / equipped.



Master Sword

3.9. Player Control and Camera

- **[W A S D]** movement in keyboard, **[Left Thumbstick]** with a gamepad.
- **[Left Mouse Button]** attacking with keyboard, **[attack]** with a gamepad.
- **[B]** inventory in keyboard, **[inventory]** with a gamepad.
- **[E]** to interact with everything, **[interact]** with a gamepad.
- **[Space bar]** to throw the current weapon, **[throw]** with a gamepad.
- **[Left Mouse Button]** on an inventory item to use it, **[use]** with a gamepad.
- **[Right Mouse Button]** on an inventory item with an open chest to store it, **[store item]** with a gamepad.

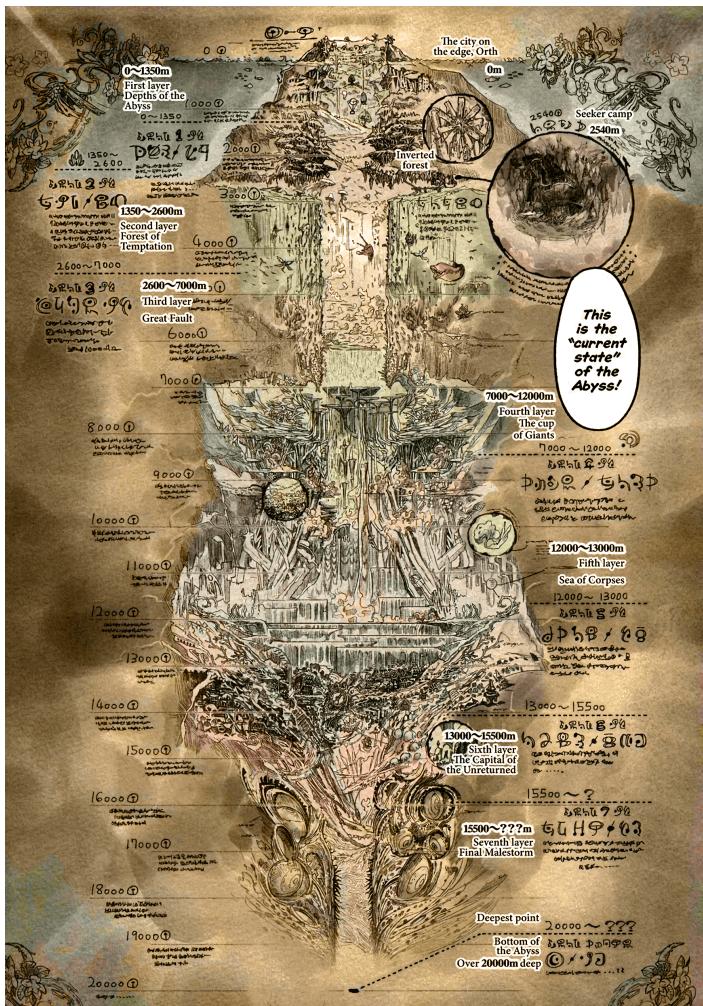
See [gamepad](#) in cut out features

The camera trails the player instead following it, meaning that whenever the player moves, the camera will shortly follow the player with a smooth movement.

3.10. Key Design References

- [Made in Abyss](#)

Initially, we wanted the dungeon to have a very deep meaning both in lore and gameplay, where each of the dungeon layers would have a separate theme and different traits with it, but was discarded for this prototype because of the complexity it would've added to the dungeon generation (see [dungeon](#) in cut out features).



In Made in Abyss, each layer of the map has different traits, including different objects and enemies, and the deeper the point the user is at the abyss, the harder it is ([Full resolution image of the abyss here](#)). The abyss served as an architectonical figure into building the city in its circular crater ([full resolution image of the city and the](#)



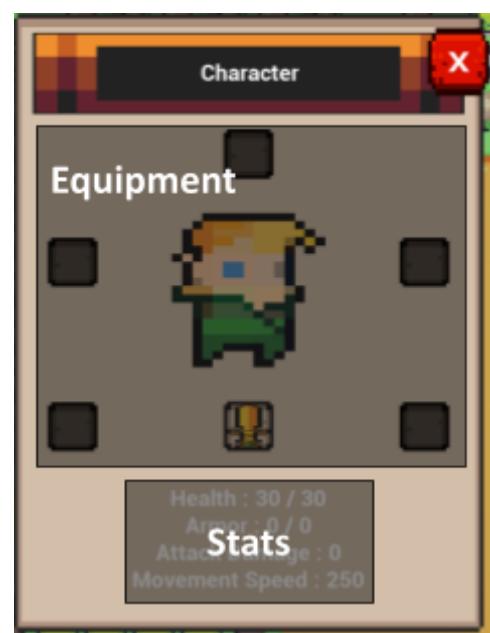
[abyss here](#))

- [Escape from Tarkov](#)

Although nothing alike in terms of genre, our game shares some mechanics with the FPS, the main one being that the player needs to equip itself with things it has gathered from the dungeon and save things in its house for further crafting. In Escape From Tarkov, the player has a series of slots in which a piece of equipment can be placed. The decision of having something in it or not after entering any of its maps is up to the player. We also gave the player a **permanent** inventory in which it can store things as it progresses through the game.



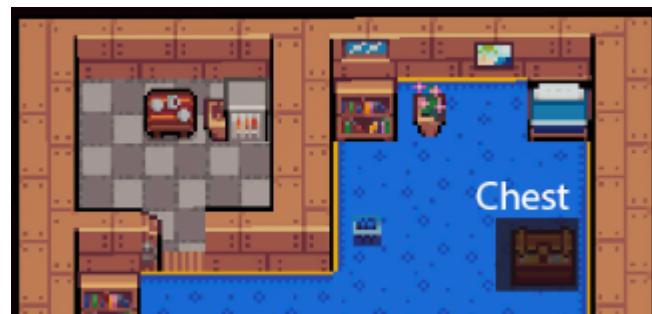
Player slots in **Escape from Tarkov**



Character Page in **Unleveled**



Inventory in **Escape from Tarkov**



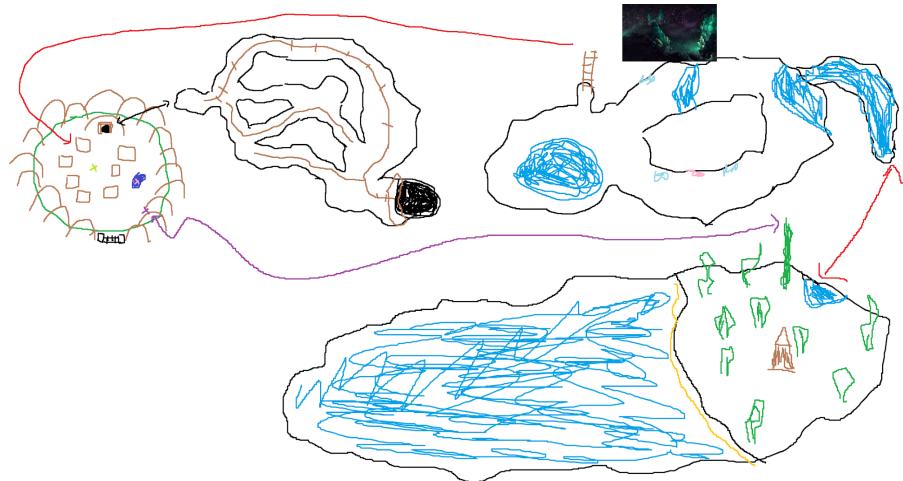
Chest in the house



4. Level Design

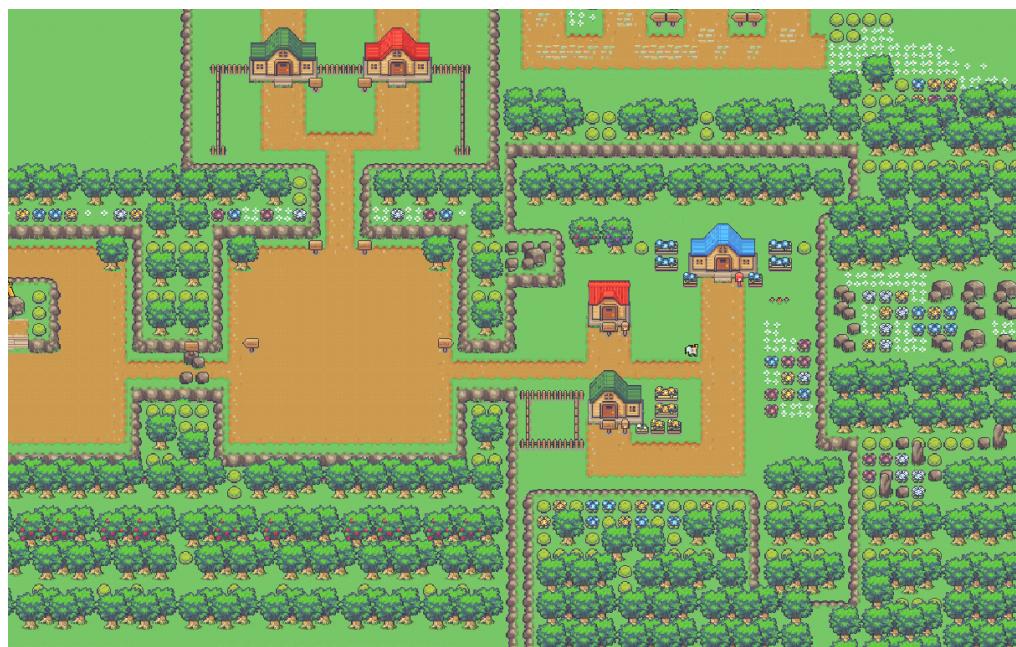
4.1. Game World Design

Certain parts of the overworld are tightly connected to the **Dungeon**, serving as **shortcuts** to different **layers** (see [dungeon](#) in cut out features).



An early prototype of the dungeon layering

The mountain covered circle represents the village (**Overworld**), and each one of the drawn places represents a different dungeon from a different layer, and each one of them has 5 separate floors. As the player progresses through the game, shortcuts can be unlocked from the village to intermediate points between the layers (see [dungeon](#) in cut out features).



Screenshot of our actual Overworld

4.2. Dungeon Design

4.2.1. Initial Dungeon Design

Initially, each of the rooms would have been pre-made and would have small areas in which enemies and/or loot could appear, and these rooms would be connected with pathways (see [game world design](#)).



Initial Concept of a single dungeon room

4.2.2. Drunkard's Walk Algorithm

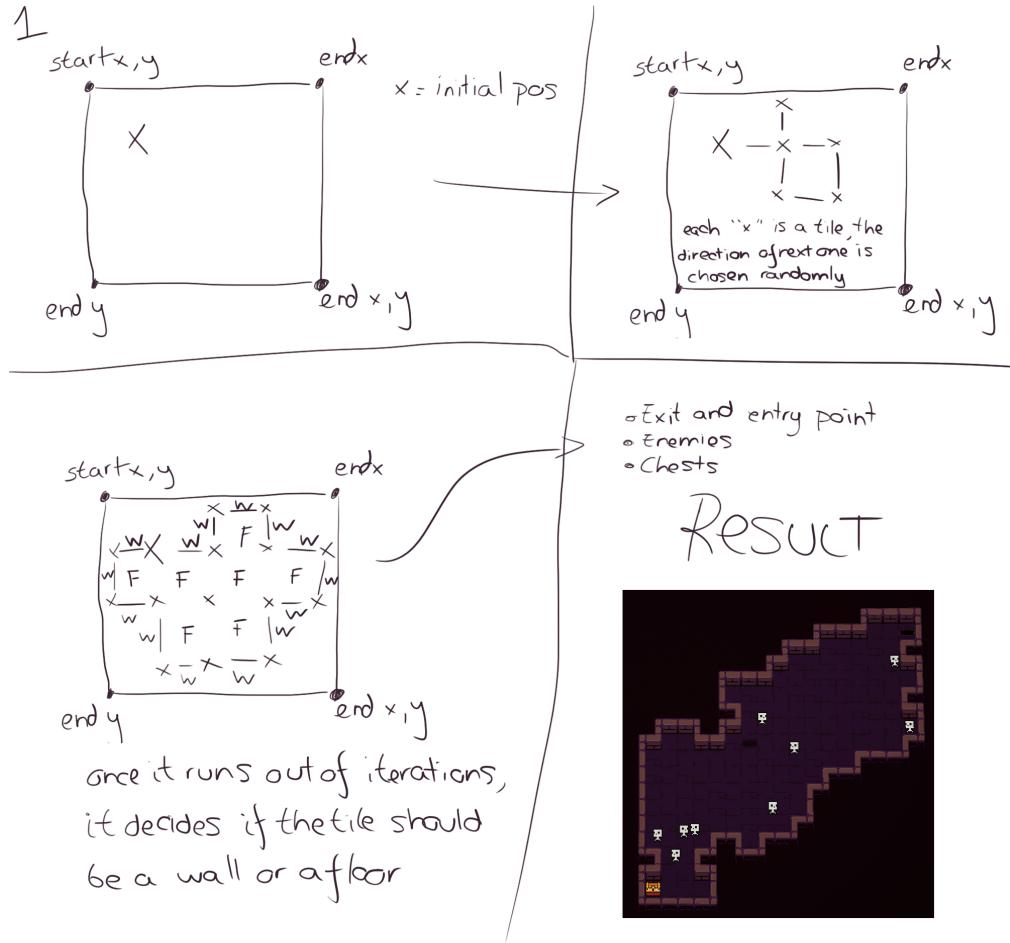
The initial dungeon design was discarded due to the possible complications that making the pathways could cause and was instead changed to a single-floor single-room algorithm but with a much wider result spectrum, where the room themselves would be the random factor. In the end we decided to have a version of the [Drunkard's Walk](#) algorithm implemented. Our algorithm's procedure consists of:

- Having a pre-sized grid and placing an **entity** in a given position (in our case this position is the middle of the grid).
- Each iteration (up to a maximum of X, in our case depends on the dungeon size and difficulty), the **entity** walks towards a random direction, and if the tile under that given position hasn't been marked, it is now marked.
- When the algorithm runs out of iterations, all marked tiles shall now get processed. In this process, the tile will be evaluated to be either a **wall** or a **floor**.
- After the edge cases have been evaluated, the entry and exit point are set up randomly with a minimum distance between them.



Entry point (left) and exit point (right)

- Then the dungeon is divided into different area types, which are used to set up closed zones where enemies and loot can be spawned.
- Spawn enemies and loot in those areas and once that is done mark the dungeon as finished.



Simplified / visual explanation of the algorithm

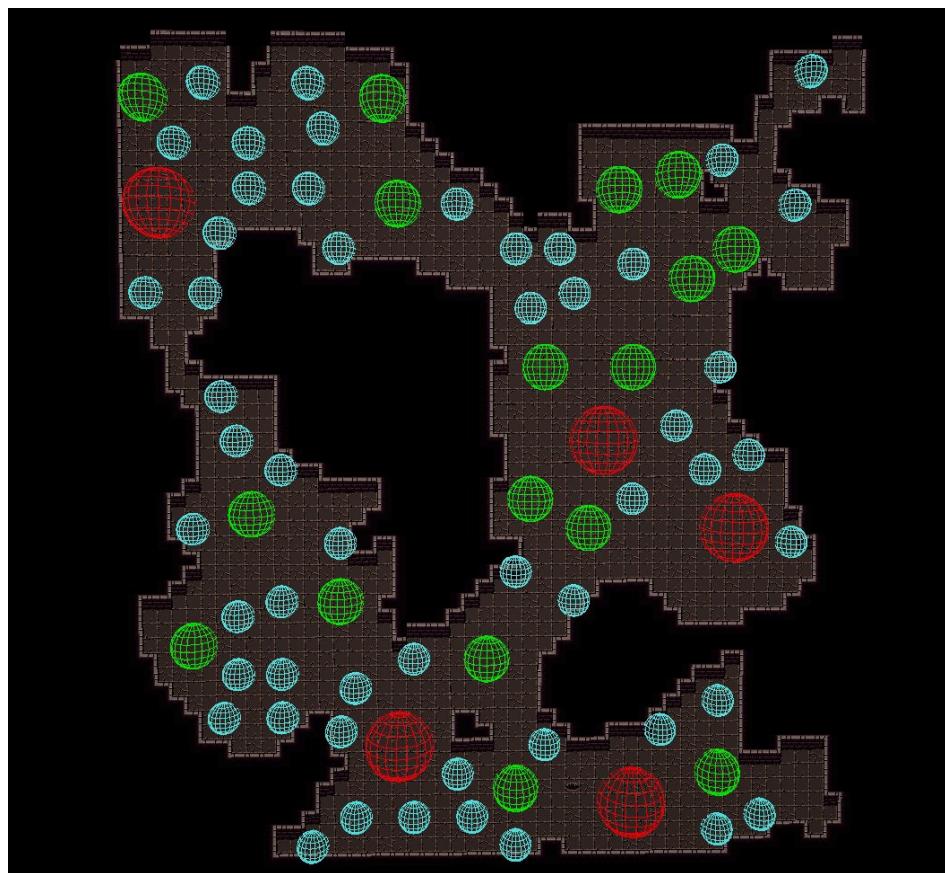
With this implementation, we achieved much more versatile and extravagant layouts at the cost of having to specifically consider a lot of edge cases.

The dungeon was initially going to be divided by several layers each of one would consist of several floors with common traits such as visual appearance and a variety of enemies, but this idea was discarded due to added complexity to the generator. Some layers were going to have an intermediate level that would've allowed the player to skip entire layers (from start to layer 3, from layer 1 to layer 3....), but this idea was also discarded because of the aforementioned possible added complexity (see [dungeon](#) in cut out features).

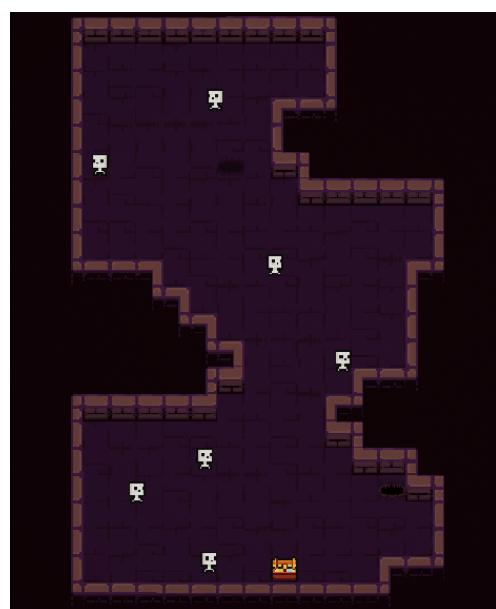
4.2.3. Loot and Enemy Generation

Once the dungeon has been fully generated, all its innards must be generated, which include everything related to things the player can gather from it. To do this, we used a method that consists on the following points:

- For each dungeon size (small, medium and big), there is a set of variables that increase both with the aforementioned and with an logarithmic curve which is bound to how many dungeons the player has explored throughout the same run. This set of variables determines the maximum amount of chests.
- Each dungeon is then divided into smaller **areas**, and it is in these areas that the loot can be spawned, as well as enemies.



An early example on how these areas divide the dungeon, **red** for big areas,
green for medium ones and **cyan** for small ones



An up to date example of loot and enemies

- Each chest rolls a number between 2 and 6 to determine the amount of objects it will have inside it, and depending on the smaller area it is within, the odds of a rarer object increase (the bigger the area the bigger the chance). Once this has been calculated, an object with the given rarity is requested to the **item manager** (holds a reference to every item class and is used to request items from it) and then it is placed inside the chest.



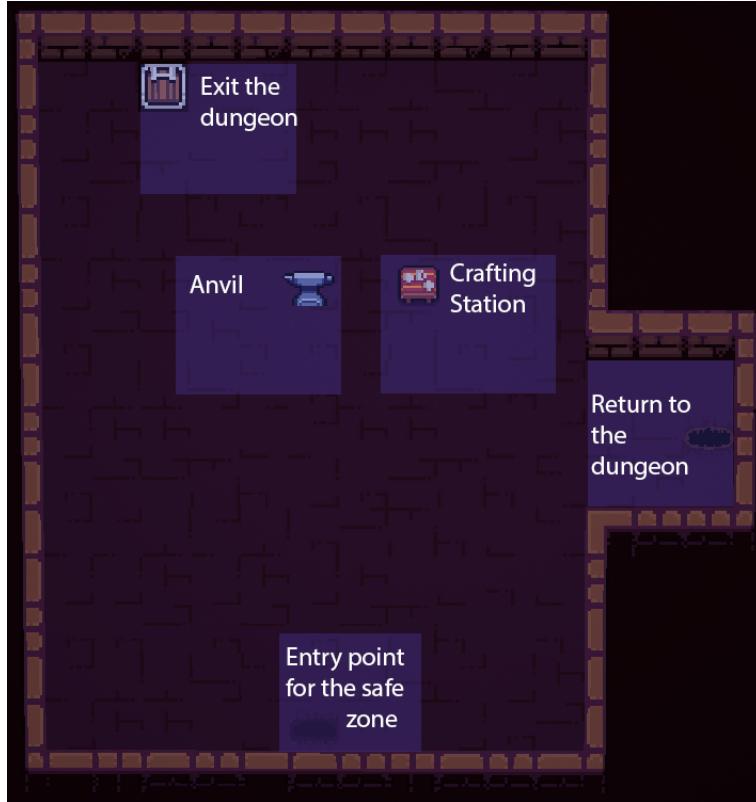
Example of a chest inside the dungeon

- For the enemy spawning, we created a **pool manager** takes care of all the enemies that need to be placed in the dungeon, and the way it works consists on the following points:
 - At the start of the generation, the pool size of each enemy is **requested for change**.
 - After this **request**, the pool manager will then decide if it should start allocating more objects or leave it as it is, which depends on the current size (requested size) and the **actual size** (amount of allocated objects).
 - If the current size is bigger than the actual size, then the pool will begin to add items to the pool until its size is filled (one item per tick, to avoid game stalling). While the pools are being allocated, the dungeon is being generated, which leaves plenty of time for the pools to be allocated.
 - Once the tiling generation has finished, the generator **requests** items from the pools and **places** them if they are valid. Repeats this step until all enemies are spawned or the dungeon has run out of budget.
 - Each enemy has a certain budget cost that will be subtracted from the total dungeon budget. The dungeon budget increases with dungeon size and the number of consecutive dungeons the player has gone through.
 - Finally, when the dungeon is being **reseted**, all items are returned to the pool and **re-initialized**.

4.2.4. Safe Zone

Every 5 dungeons and after a **boss zone**, instead of teleporting the player to the next dungeon, it will be teleported to the **safe zone**, where the following actions can be made:

- Continue to the next floor: **Return to the dungeon**.
- Exit the dungeon: **Exit the dungeon**.
- Repair armor: interacting with the anvil will restore the players' armor to its full.
- Craft new objects: using the crafting station with known recipes and materials to create new objects



Safe zone layout

When the player exits the dungeon, all internal variables as well as enemy pools are reseted.

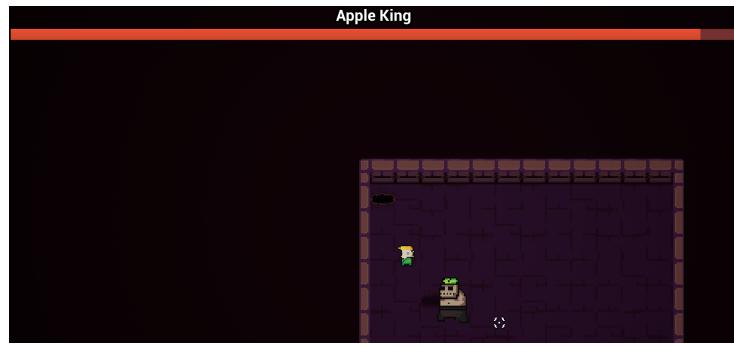
4.2.5. Boss Zone

The final room in that floor always consists of a **boss zone**, where the dungeon generation basically follows the same process but changes once the tiling has finished. Once the tiling has finished, a single enemy is spawned (a boss).

When the player attacks it, it will show a health bar on top, and the boss' behaviour will change to be **aggressive** (read more about enemy behaviour [here](#)).



A **roaming** boss, wait for the player to attack it.



An **aggressive** boss, and its health bar on top



When the boss is defeated, a **timed chest** (after this time the chest will get destroyed) will appear with **exclusive loot**.

Also, if the player does **not** want to fight the boss, it has the option to **run** to the exit instead.



5. UI

5.1. UI Distribution

5.1.1. Main Menu:



We wanted to give the player a brief view of what the main level (Overworld) looks like from the moment it launches it, which is why we decided to have the actual buttons on the side, with the title on top and a moving background.



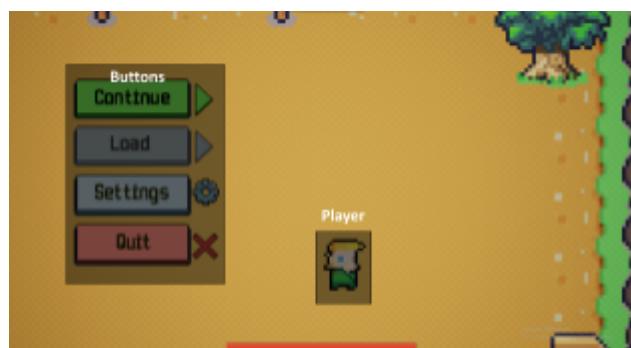
New Game: starts a new game

Continue (**only available if a save exists**): loads the saved game state.

Settings: **

Quit: Exits the application

5.1.2. Pause Menu:



Having the same reasoning that the Main Menu has, we basically tried to keep everything from the main scene while integrating everything we needed or thought was necessary for the Pause Menu. When the player pauses the game, the camera is zoomed towards it.



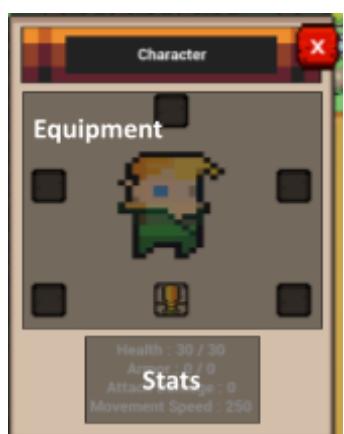
Continue: resumes the game

Load (only active if a save exists): allows the player to quickly load a previous game state.

Settings: **

Quit: Exits the application.

5.1.3. Character Page:



Equipment: Displays everything that the player has currently equipped

Stats: Displays the current stats of the player. For HP and Armor: Current / Max

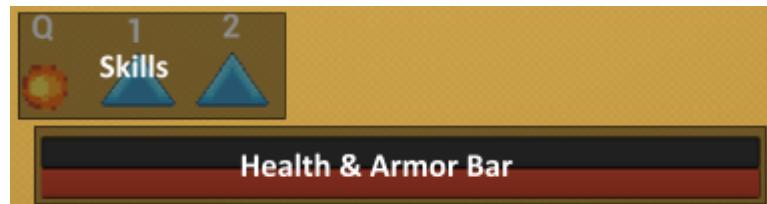
Whenever a **gear** type item is equipped, its sprite will be displayed in its assigned slot in the equipment section, and the provided gear stats will be added into the Stats section.

5.1.4. Inventory:



Allows the player to store a limited amount of things. Each of the slot consists of a button that is only active when something is in it. **Using** an item has different effects depending on the item type.

5.1.5. Bottom Bars:



Being at the bottom of the screen, both the health and armor bars needed to have distinctive colors, which is why we opted to use red and light grey (health and armor, respectively). The skills bar, on the other hand, consists of descriptive icons** that shall give a brief visual description of it.

5.1.6. Recipe Menu:



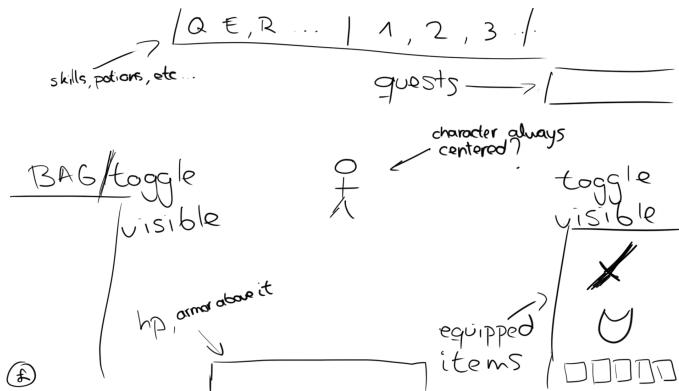
Because we decided to make the crafting workbench (limiting the crafting to this specific object) instead of allowing the player to craft everywhere, we opted for a more minimalistic interface, where:

Recipe list: list of all the learned recipes

Ingredients: once a recipe has been selected, the necessary ingredients to craft it will be displayed here.

Craft Button: if the player has all the necessary items to craft that object, this button will be active and **red**, otherwise it will be inactive and **grey**.

5.2. Concepts

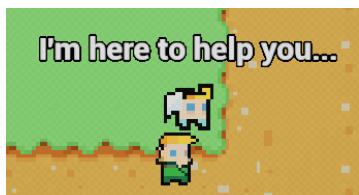


Initially, we planned to have the following items in the UI:

- An equipment section where everything that the player has equipped can be seen. **Semi transparent and not removable.**
- An HP and Armor section where initially there would be an HP bar and an Armor bar on top of it.
- An inventory section (its visibility can be toggled).
- A top bar which would have quick access to skills and usable items such as potions. **
- A quests bar which would display the title as well as a small text regarding the currently active quest(s). **

Out of this initial UI concept, we removed what we thought was **not** achievable or didn't make sense for this prototype, which includes the quests bar **, the hotkey bar **, and the equipment section (Character Page).

5.3. Text



Dialogs



Damage text



Interacting



Boss chest

6. Sound

6.1. Music References

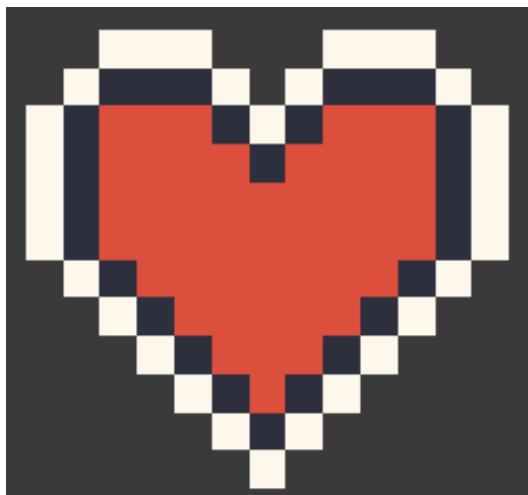
- Stardew Valley - Calm and relaxing music for the **Overworld** to help the player relax after some dungeon diving.
- Moonlighter - Music for the dungeon. Used to emphasise the danger theme within the dungeon.

6.2. Where to place the sounds

Here is a list of where we want to introduce sound in the game and why (see [sound](#) in cut out features):

- Menu buttons: click sound to give feedback when selected
- When an enemy or the player is hit: We want to give auditive feedback to the player when he is hit.
- When the player interacts with things: An exclamation sound to tell the player that it has interacted with something.
- When the player moves: Satisfying footstep when walking.
- Background music for the overworld / dungeon levels: Tense music to set the player mood.
- Background music for boss battles: Really active music to give more intensity to the battle.
- When the player completes a quest: A rewarding sound that satisfies the player when a quest is completed.
- NPC sounds simulation - speaking (Animal crossing): To give the player the feeling that it is talking to someone.

7. Save game



- Interactable entity. When the player interacts with the object, which is located in the player's house, it saves the current game.
- It stores the current **player's position**, the **player's inventory**, **equipped items**, all the items inside the **player's house chest** and the current **player's stats**.
- After saving the game then a save game slot is created allowing both menu buttons, continue from main menu & load from pause menu, to be pressed.
- When the player loads a previous game save everything is then restored to what the save game slot has stored.
- In the future we want to introduce here also the current quests that have been completed so that they don't become infinite. (see [save game](#) in cut out features)

8. Cut out Features

8.1. Art

- Most of the **art** we used in our project belongs to non copyrighted pages and does **not** necessarily compliment each other. This includes both the visual and sound aspects of the game. See [bibliography](#) for more.
- Designing all the **sounds** as well as the **music** for the game was definitely out of scope.

8.2. Quests

- For this prototype, we considered that making a global **quest** manager would have been far too labor-intensive both in terms of time and complexity.

8.3. NPCs

- **NPCs** did not make it for this prototype due to the lack of cohesive lore it would've required to create. We also planned to have a single **shop** where the player could spend and gain some **in game currency** (gold) by buying or selling some materials or gear, respectively, but was also discarded when we decided that dropping objects from the enemies was not a possibility.

8.4. Items

- Having unique items initially required us to create the **Questing Manager**, which was discarded for this prototype.
- In the **tutorial**, we intended to show the player the **4 types** of available **weapons** and show how they work to finally let the player choose one of those, which would be selected as its starting weapon.
- 5 slots were planned for **armor equipment** but were reduced to 2 due to the amount of recipes we would've needed to make to craft all the parts .
- **Equipment upgrades** were discarded for this prototype because we needed planned them to be within the NPC's.
- We planned to create customizable **runes slots** for each weapon but we did not have time to implement it.
- Even though we had a basic **random equipment generator**, we quickly realised that the player could cheat its way into its probabilities and force a better item to be spawned.
- Being virtually the same as **throwing** the **weapons**, we thought that having throwables did not make much sense if the player can directly throw its weapon.

8.5. Dungeon

- Making a layered-based dungeon generation system required a lot of art manipulation (**which is fair to say that none of us knows how to do that properly**), a very good understanding on writing a cohesive and coherent lore and game state management, all

of which we considered was out of scope for this period of time. Shortcuts were also discarded for the same reasons.

- The dungeon progression was closely related to making a layered-based dungeon mostly in terms of different enemies and objects that shall be dropped exclusively on certain areas, which was discarded for the same reasons. The current dungeon progression depends on the amount of dungeons the player has generated in the same *run*.

8.6. Tutorial

- We initially intended for the **tutorial** to be at the start of the game and then leave a training area in which those mechanics could be explained again. In the end we created a small area at the top of the player starting position which explains three basic mechanics, attacking, throwing weapons and crafting. Letting the player choose its starting weapon was also discarded due to the initial complexity of creating 4 different weapons.

8.7. Extras

- We moved the hotkey bar to the bottom and changed the equipment section to a window visibility of which can be toggled.
- Adding another way of acquiring loot was planned, and it would've mainly been drops when an enemy dies.
- For this prototype, everything was made exclusively for **mouse and keyboard**, **gamepad controls** were excluded mainly for any possible UI control problems.
- Having the **Leveled Sword** implemented was not one of our priorities and because of this, its implementation got excluded from this prototype.

8.8. Sound

- We want to have sound across the entire game but, for this prototype, we only had time to introduce some **SFX** to the buttons from the main menu and the pause menu.
- The rest of the **SFX** and **OST** will be introduced at a later point of the development but the idea is to have at least the ones mentioned [here](#).

8.9. Lore

- Very closely related to the [dungeon](#), the main setup of the game requires the player to progress through it, defeating its bosses and advancing through the different layers to remove the curses that the village habitants have. Considering that we did not have the time to create the layering system for the dungeon, we considered that adding any kind of lore at this point of the project was out of scope for this period of time.

8.10. Save game

- As of now the player is only capable of saving its current position, inventory, stored items at the house chest, equipped items and current stats but we also want to store the current state of the quests so that when the player completes a quest it actually becomes finished and therefore can't be completed again.
- We haven't included this in our save game because in the actual prototype we don't have any quests so we thought that it wasn't needed at this point in time.
- Another thing that we want to store is the current state of some specific NPCs such as the blacksmith, the first time you encounter that NPCs he will be inside the dungeon but after rescuing him he will then appear at the village. This is one thing that needs to be stored in the save game.
- In our prototype we don't have the actual blacksmith or any relevant NPC that needs this kind of system implemented and save alongside the rest of the save data so we thought that it wasn't needed at this point in time.

9. Bibliography

- Unreal Engine: <https://www.unrealengine.com>
- Dungeon tilesets & enemies
 - <https://pixel-poem.itch.io/dungeon-assetpuck>
 - <https://0x72.itch.io/dungeontileset-ii>
- UI
 - <https://0x72.itch.io/dungeonui>
- Characters
 - <https://0x72.itch.io/pixeldudesmaker>
 - <https://superdark.itch.io/enchanted-forest-characters>
 - <https://superdark.itch.io/16x16-free-npc-pack>