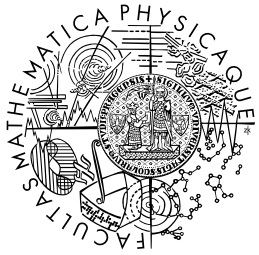

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

GAME DESIGN DOCUMENT

Jiří Filek, Martina Fusková, Shivam Sharma

Dungeon of Chaos

Department of Software and Computer Science Education

Supervisor of the software project: Pavel Ježek

Study programme: Visual Computing and Game Development

Study branch: Computer game development

Prague 2023

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Title: Dungeon of Chaos

Author: Jiří Filek, Martina Fusková, Shivam Sharma

Department: Department of Software and Computer Science Education

Supervisor: Pavel Ježek, Department of Distributed and Dependable Systems

Contents

Introduction

This document is aimed mainly at game designers. However, it may be useful for all developers working on this game, as it explains the features of the game and the ideas behind their design.

The first chapter of this document is concerned with the main features of the game. The reader will find there their overview, descriptions, and explanations. This chapter should provide an overall understanding of the game.

The following chapters then focus on the content of the game and its relation to the features. Each content chapter starts with a description of the content, guidelines for their design, a list of content in the game, and a tutorial for creating new or adjusting the existing content in Unity.

The last two chapters are aimed at sound designers and graphic designers. The sound design chapter is very similar to the content chapters, as it contains guidelines for their creation and tutorial in Unity. The last chapter then provides the user with visual guidelines describing the general visual style, and color palette, as well as a tutorial for creating VFX in Unity.

The software documentation is located in a separate file and it provides information about the implementation of features and content. The chapters of this document are quite similar to the chapters of the software documentation for simpler navigation between the two documents. However, both documents are written in a way that they can be read without reading the other one.

1. Game Overview

Dungeon of Chaos is a 2D dungeon crawler game developed in Unity using C#. The player controls a character that moves through dungeons fighting enemies and trying to get stronger. The game focuses on three gameplay mechanics — combat, character progression through statistics, and exploration. The combat is the core mechanic and the remaining two are supplementing it. These mechanics create the core loop of our game as can be seen in figure 1.1.

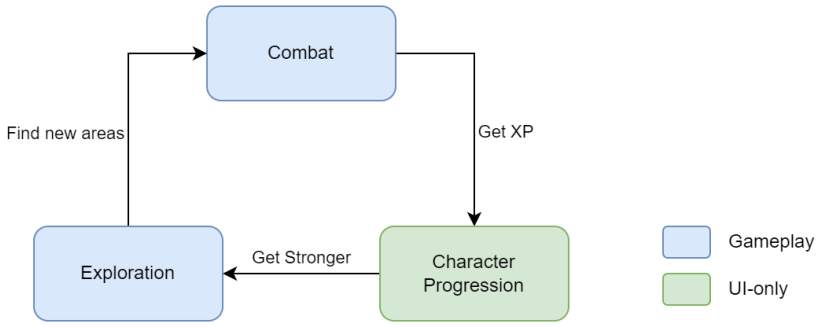

Figure 1.1: Core Loop

The goal of the game is to finish all dungeons. In order to fulfill the goal, the player needs to explore new areas and fight the enemies they will encounter. Killing enemies gives the player experience and the player can strengthen their character. Once the character is strong enough, the player can continue exploring.

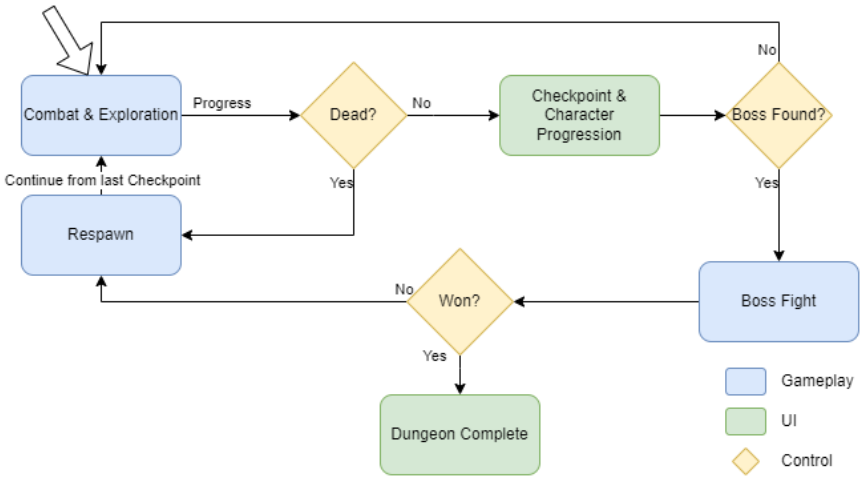

Figure 1.2: Dungeon's Flow

Figure 1.2 shows the flow within one dungeon in more detail. We can see that the player's goal in the dungeon is to find the boss and defeat him in order to

proceed to the next dungeon. Fulfilling this goal still requires the same steps — explore, fight, and get stronger. The player's character can die during the fighting phase. Dying results in respawning at the last visited checkpoint and deleting all progress that happened afterward.

The game has a dark fantasy setting which is reflected in the overall audio-visual part of the game and the content. The enemies (such as orcs, skeletons, or demons), as well as dungeon objects (such as chests, torches, or traps), are all crafted to fit in the medieval fantasy style. The dungeons have very low ambient light and are lit mostly by torches, in order to emphasize the dark atmosphere, as we can see in figure 1.3.

Figure 1.3: Game Screenshot

1.1 Exploration

The player needs to explore the dungeon in order to progress. They must find the boss room and since the dungeon is non-linear, as is shown in figure 1.4, that's impossible to do without exploring. Moreover, the player needs to search for checkpoints to save their progress and level up if possible.

Another important incentive for exploration is power creeping which is also a key part of our game. It is quite probable that the player might die quite often, thus exploring other areas to get stronger, will be useful. Moreover, quite often the player can be rewarded with loot when exploring.

More information about the design of dungeons and dungeon objects, as well as a tutorial for generating new dungeons can be found in chapter 6.

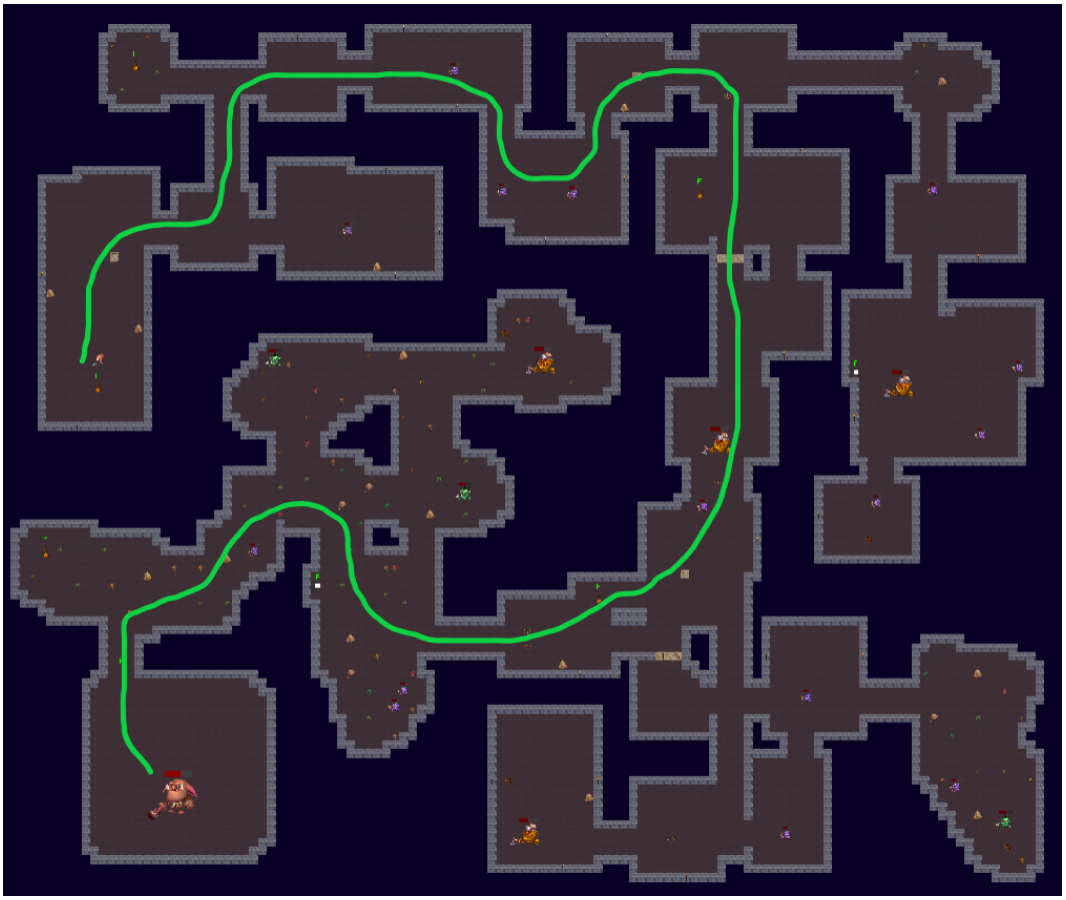

Figure 1.4: Dungeon's Layout

1.2 Combat

The biggest emphasis is placed on combat. We were trying to create an experience that requires skill from the player but at the same time keeps the action. It should be easy to pick up but quite hard to master.

We wanted to achieve skillfulness by giving the player more options on what to do during combat and giving him at least some time to decide and react, as opposed to combats that require fast reflexes and speed from the player.

Telegraphing the attacks is done by showing an indicator of the area of effect in the case of melee attacks or by simple animations in the case of ranged attacks (such as a slowly appearing projectile). An example of a melee indicator is shown in figure 1.5. This way is far more readable in 2D where it is hard to show both the direction and type of the attack by rotating the weapon. Therefore, we are showing the direction by weapon rotation (both the player and the enemies can attack in all directions) and the type of attack by the indicator or the simple animation (this is shown only for the enemies).

In order to ensure the action and also put a bigger emphasis on the decisions, all hits are meaningful as they deal quite a lot of damage. Moreover, the player is

Figure 1.5: Attack Indicator

discouraged from attacking, dashing, or using spells randomly, as all these actions cost either mana or stamina which are quite valuable resources.

In order to make it more interesting, there are various types of melee, ranged, and special attacks. The most basic enemies can do one of those attacks, the stronger enemies have more of them. Enemies and their attacks will be described in more detail in chapter 5.

1.3 Character Progression

The last part of the core loop is the character progression through statistics, as is typical for the dungeon crawler genre. In our game, the player can progress in two systems — stats (such as damage, hp, mana...) and skills (such as a new type of attack, spell, or a passive skill).

It is quite typical to include also some kind of a gear system, however, due to the smaller scope of our game, we have decided not to. Therefore, the only loot the player can collect are consumables that immediately replenish hp, mana, or stamina, experience points, or items that can be used for resetting points invested into skills.

The stats system is common for both the player's character and enemies. Though, unlike the player's character, enemies cannot level up. The stats system will be described in more detail in chapter 3.

The skill system is used solely by the player's character, however, some of the skills inside of the system can be used by the enemies as well. The system is described in chapter 4.

2. Player's Character

The player controls only one character whose visuals and basic actions are set from the beginning. The only way to modify the character is through the character progression system, in which the player can set the values of the stats and select special actions (skills).

2.1 Actions

The player's character can do the following actions in the game (outside of UI screens). Some of the actions require mana or stamina to be used.

- Movement - the character can move in all directions using `WSAD`. It doesn't cost anything.
- Dash - the character moves faster in the selected direction. The dash can be further upgraded in the skill system to have an additional effect such as dealing damage to enemies or healing the player. The dash is used by pressing the `SPACE` key and the direction is selected based on the last direction the character was moving in. Using the dash costs stamina and in the case of the upgraded versions it might also cost mana.
- Basic attack - the character can attack in all directions, the direction is set by the mouse cursor, and the attack is used by pressing the `LEFT MOUSE` key. It is a basic melee attack whose damage depends on the stats of the character, all other parameters such as the speed or reach of the attack are fixed. The basic attack costs stamina.
- Secondary attack - it is a more advanced version of the basic attack that needs to be learned in the skill system. Secondary attacks differ in values such as speed, reach, stamina cost, or damage. A secondary attack is used by pressing the `RIGHT MOUSE` key.
- Skills - the character can do also other special actions, called skills, that are learned in the skill system. Most of the skills are supposed to be used in combat, as they are usually special types of attacks (such as ranged or AoE), combat buffs or debuffs, or healing abilities. They cost stamina, mana, or both. They are mapped to keys `Q`, `E`, `1`, `2`, and `3`.
- Interact - the character can interact with some of the objects such as checkpoints or chests by pressing the `F` key.

2.2 Character Progression

If the player gains enough XP they can level up their character. XPs are gained through killing enemies and exploration of the dungeon. Each new level gives the player 2 stat points and 1 skill point.

Stat points can be invested into primary stats (Strength, Intelligence, Constitution, Endurance, Wisdom). The player can redistribute them anytime they want (while resting at checkpoint), but the values of the primary stats can't go below 10 or above 30. The stat system is further explained in chapter 3.

Skill points can be invested into learning new skills. There are passive skills whose effects are applied automatically and they provide bonuses to the character's stats or add a new stat to the character. Active skills can be actively used by the player and they give their character new actions.

Each new skill or skill upgrade costs 1 skill point but there might be also other requirements (such as the level of the character). The skill system is explained in more detail in chapter 4.

3. Stats

There are two types of stats - primary stats which can be leveled up and secondary stats which are calculated from primary stats and can be further improved by skills.

3.1 Primary

Primary stats have a minimum value of 0 and a maximal value of 30. The player's character starts with all stats on level 10 which is an "average" value. Enemies then have the primary stat below or above average based on their type (physical or magical) and their supposed power (basic, elite, or boss enemy).

There are the following primary stats:

- Strength
- Intelligence
- Constitution
- Endurance
- Wisdom

The player's character can level them by investing stat points. Each stat level costs 1 stat point. Stat points are obtained only by leveling up.

3.2 Secondary

The secondary stats are calculated from the primary stats and they can be further affected by both passive and active skills. The secondary stats can be divided into two categories, regenerable stats, regular stats, and tertiary stats.

There are the following secondary stats:

- Physical damage
- Spell power
- Max HP
- Max Stamina

- Max Mana
- Armor
- Stamina regeneration

3.2.1 Regenerable Stats

The regenerable stats are HP, Stamina, and Mana. Regenerable stats can be recovered up to their maximal value which is determined by the secondary stat. All regenerable stats are calculated as follows:
$RegenerableStat = 100 * exp((PrimaryStat - 10) * 0.07) + (PrimaryStat - 10) * 3$.
All regenerable stats are recovered after resting at the checkpoint.

Max HP is derived from Constitution. During the game, it can be recovered by healing spells or health essences collected from dead enemies. When the player's character reaches zero HP, they die and they are respawned at the last visited checkpoint (unless they have the second breath skill unlocked).

Max Stamina is derived from Endurance. It is consumed by using the dash and all physical-based attacks (basic, and secondary attack) and skills. Since it is basically impossible to fight enemies without stamina or run away from them, the stamina is regenerated automatically. The regenerated amount is determined by the tertiary stat Stamina regeneration.

Max Mana is derived from Wisdom. It is consumed only by using spells, therefore it is not regenerated automatically. The only way to regenerate mana is by resting at checkpoints or collecting mana essences from dead enemies.

3.2.2 Regular Secondary Stats

There are two regular secondary stats, Physical Damage, and Spell Power. Physical Damage is calculated as $PhysicalDamage = 2.5 * Strength$ and Spell Power as $SpellPower = 2.75 * Intelligence$.

Physical damage affects the damage dealt by the basic attack, the secondary attack, and the skills related to physical damage. It is calculated from the strength primary stat.

Spell power affects the values of all spells, such as the healed amount by the healing spell, or the damage healed by a fireball. It is calculated from the intelligence primary stat.

3.2.3 Tertiary Stats

There are two tertiary stats, Armor and Stamina Regeneration. Both these stats have a unique formula.

Stamina Regeneration is derived from Endurance. Its value is increased only every 3 levels of the Endurance stat and it is calculated as follows: $StaminaRegen = Mathf.Floor(Endurance/3) * 6$. Stamina regeneration can be further increased by passive skills.

Armor is derived from Constitution. It is consumed by receiving damage and similarly to regenerable stats, it is recovered to its value determined by the stat after resting at the checkpoint. However, the value can be further replenished beyond that by using skills. It is calculated as follows: $Armor = (Constitution - 10)/2 * 5$

3.3 Stats Derived from Skills

There is a list of stats that are invisible to the player as they cannot be leveled directly using stat points. However, they can be improved by unlocking corresponding passive skills. Namely, it is the following stats:

- Skill cooldown decrease - a value that determines by how much the cooldown of all skills is decreased
- XP income multiplier - a multiplier by which each gained XPs are multiplied
- Lives - the number of times the player's character gets resurrected
- Visibility range - the range of the light around the player's character

3.4 Enemy Stat Modifiers

Each enemy has their primary stats assigned and they are the same as the character's primary stat. Those stats are then further modified by the power type of the enemy (basic, elite, and boss) and the level of the enemy. The secondary stats are then calculated from the modified stats in the same way as the character's stats with the exception of HP.

The power type gives a flat bonus to all the base stats. Basic gives 0, elite gives 2 and the boss gives 4. Each level then gives a 0.75 bonus to each stat.

As mentioned before, the final secondary stats are then calculated from these modified stats. The same formulas as for the character are used. The only

exception is HP which is further multiplied based on the power type of the enemy. Elite enemies have the multiplier set to 1.5, and boss enemies to 3.

3.5 Setting up Stats in Unity

All values for enemies and the formulas that are used for calculating modified primary stats and secondary stats are saved in a spreadsheet where it is easier to do the balancing. The values then need to be copied over to Unity to corresponding `ScriptableObject` which are described in the following subsections.

3.5.1 Stats

Each unit has its own `ScriptableObject` with the stats, where the designer can set the primary stats, movement speed, stamina regeneration, chase distance leveling data, formulas for calculating secondary stats, and enemy modifiers. Both character and the enemies are using the same `ScriptableObject`, however, some of the fields are optional depending on whether it is the character or the enemy.

As we can see from figure 3.1, for the player's character the chase distance and the stats modifiers are optional. The rest of the values are set to their values.

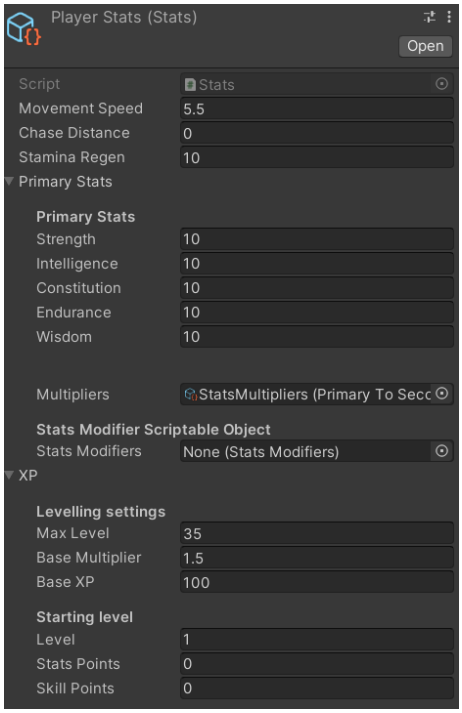

Figure 3.1: Player Stats

For enemies on the other hand, most of the leveling data are optional as they are not leveling the same way the character does. The only field from leveling data

that needs to be set is level. Thus, if we want to have two enemies of the same type with different levels, we need to create two separate `ScriptableObject` and then assign those correctly. We can see an example of enemy stats from figure 3.2.

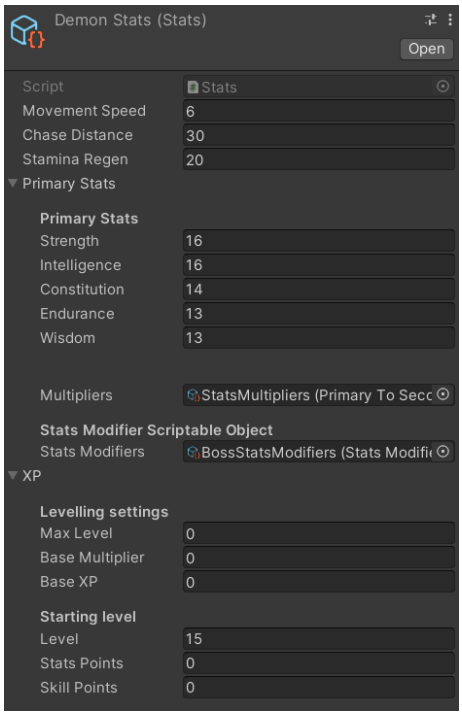

Figure 3.2: Enemy Stats

3.5.2 Stats Multipliers

The `ScriptableObject` holds the multipliers and exponents used for calculating the secondary stats from the primary ones. We have only one `ScriptableObject` for stats multipliers as they are common for the character as well as for the enemies.

However, it would be possible to have several different objects based on the unit type, power type, or for instance dungeon. The correct object would then need to be assigned in the corresponding Stats object.

The `ScriptableObject` is depicted in figure 3.3.

3.5.3 Enemy modifiers

There is one `ScriptableObject` per each power type, as the `ScriptableObject` holds only the flat bonus to all primary stats and the hp multiplier applied to the secondary stat. Similarly to the Stats Multiplier, this

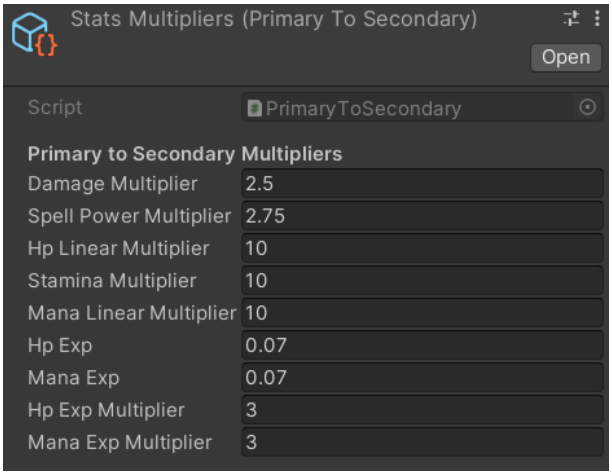

Figure 3.3: Stat Multipliers

could be easily extended if we needed more power types or wanted to have unique stats modifiers for some concrete enemies (such as the final boss).

We can see an example of enemy modifiers `ScriptableObject` in figure 3.4.

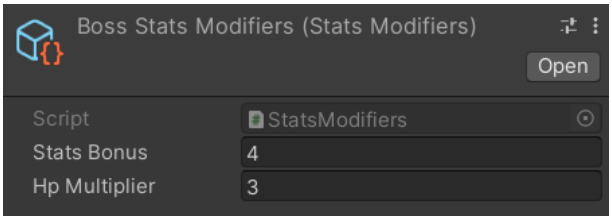

Figure 3.4: Enemy Modifiers

4. Skill System

The skills are organized in a tier-based structure where the tier determines the character level needed for unlocking the skill. Some skills might have also other requirements for unlocking or leveling up, namely a value of one or more primary stats, and other skills. Further levels of skills increase the values but they might also add some more advanced behavior to the skill.

All skills and skill levels cost the same, namely 1 skill point. Invested skill points can be redeemed by using the reset book which resets all skills. The number of reset books the player is limited as they are acquired by exploration and they don't respawn by resting at a checkpoint.

4.1 Skill Types

The skills are divided into two types - active and passive, based on whether they require activation by the player or not. Active skills need to be placed into a skill slot in order to be later used in the game. Each skill slot is bound to a unique keyboard key. Passive skills on the other hand don't require activation or any input by the player.

4.1.1 Active Skills

Active skills need to be activated by the player and they require the player's input in order for their effect to take place. They can be used both inside and outside of combat. Their usage, similar to other actions of the character, costs stamina, mana, or in the case of more advanced skills both. All active skills also have a cooldown.

They are further divided into basic active skills, dash skills, and secondary attacks. There are five skill slots for basic active skills, one slot for a secondary attack, and one slot for the dash. Aside from having different slots, these three types of active skills behave the same.

Active skills could be also divided based on the stat with which they are scaling into physical skills and spells. Physical skills scale with the physical damage and they usually require stamina. Spells scale with spell power and they cost mana. However, they might also be skills that have more parameters, some scaling with physical damage, and some with spell power. These skills usually cost both stamina and mana.

4.1.2 Passive Skills

Passive skills don't require any activation or input from the player, as they give buffs to the character's attributes. In our case, they give buffs only to attributes that cannot be upgraded directly through the stat system.

The values of passive skills scale with the level of the character. Most of them use the following formula for scaling: $finalValue = baseValue * (1 + level * 0.2)$

Currently, there are the following passive skills in the game:

Name	Description	Tier	Max level	Values
Shielded	Increases armor of the character	2	3	10; 30; 50
Tireless	Increases stamina regeneration; decreases stamina cost	8	2	10; 15
Decrease Cooldown	Decreases cooldown of all skills	11	1	5 %
Light	Increases the range of the light around the character	8	1	10
Second Breath	The character resurrects with some part of his health	13	1	30 %
Quick Learner	Increases the amount of received XP	5	2	0.1; 0.25

4.2 Character Sheet UI

All the skills are visualized in a Character Sheet which can be accessed by resting at a checkpoint. The Character Sheet has 2 tabs, the first tab shows the stats and leveling info of the character, and skills can be found on the second tab.

Figure 4.1: Character Sheet: Skills

We can see the character sheet in figure 4.1. Each column represents one tier,

and the number above the column represents the level required for unlocking the first level of the skill. The numbers below each skill then depict the current and maximal level of the skill.

All skills need to be placed in the UI, in order for the player to be able to unlock them and use them in the game. Adding new skills to the UI will be explained in the following section.

4.3 Skill Composition

Firstly, we will start by explaining the composition of the skill as each skill has more data layers that need to be correctly connected. This structure has some UX drawbacks, however, it allows the designers to easily create a high amount of different skills with a limited amount of effects. Moreover, adding new effects is very simple and it requires only basic programming knowledge.

The composition of the skills differs between active and passive skills, with passive skills being easier. They have the following structure:

- Skill Info - wrapper around skills
  - Skill - skill data (such as name, icon, and description) and the actual functionality of the skill
    - Skill Effect - part of the behavior of the skill, can contain other effects

4.3.1 Skill Info

The first layer is the `Skill Info` layer which serves as a wrapper of the skills. It holds the list of the skills (levels of the skill), unlocking requirements, the current level of the skills, and whether the skill is unlocked.

Each skill type (active, passive, secondary, and dash) has its own `SkillInfo` class and a `ScriptableObject` of the corresponding type needs to be created. However, the structure within the `ScriptableObject` is the same for all types.

The structure of the `ScriptableObject` is shown in figure 4.2. The list of skills holds all the levels of the skill, it is important that the skills are correctly ordered. The list of skill requirements then corresponds to the list of skills. Only the first level is obligatory, as the others can be calculated from the first one (unless you want to have some special requirements).

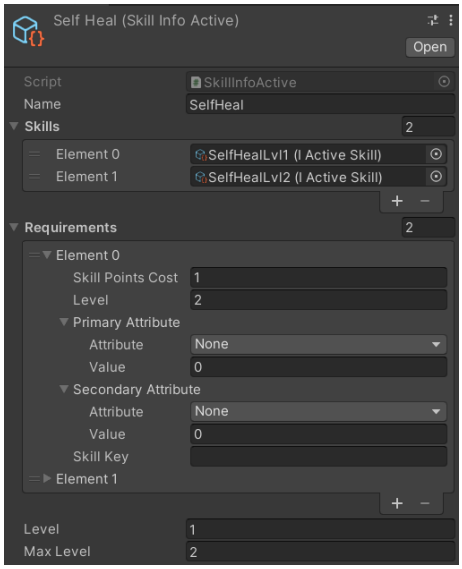

Figure 4.2: Skill Info

4.3.2 Skill

The next layer is the `Skill` itself. The behavior and data of the `Skill` object differ based on the type, thus again each skill type has its own `Skill` class. Opposite to `Skill Infos`, the structure of `ScriptableObjects` of `Skill` of different types differs significantly. The only thing they have in common is the `Skill Data` which holds the name, description, and icon of the skill.

We will start with the passive skills as they are the simplest in terms of their structure. The `ScriptableObject` contains only the `SkillData` and then the value by which a given stat should be increased. The only tricky part is specifying the stat, as this is done by creating a unique script for each stat and then creating a `ScriptableObject` of the type of the script. However, as we can see, the script is very simple:

```
[CreateAssetMenu(menuName =
   "SO/Skills/PassiveSkills/Armor")]
public class IncreaseArmor : IncreaseStat
{
    protected override void ChangeStat(Stats stats, float
       val)
    {
        stats.SetArmor(val);
    }
}
```

Listing 4.1: Example of Increase Stat script

Figure 4.3 shows a simple passive skill and its parameters. We can see the

`Skill Data` that are common to all skills. Other than that, the skill has only `amount` parameter representing the value of the skill.

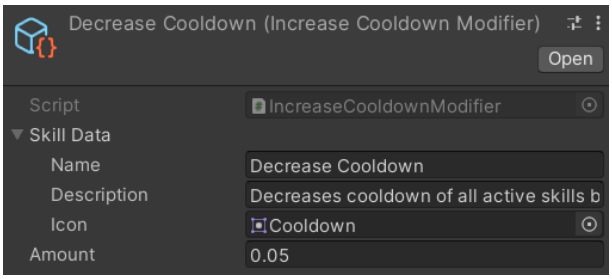

Figure 4.3: Passive Skill

The next are basic active skills for which we need to create a `ScriptableObject` of the type `IActiveSkill`. We need to define the skill data, cooldown, mana cost, stamina cost, and the list of effects. When the skill is used, it consumes the specified amount of mana and stamina and immediately executes all the effects in the list. Effects are described in more detail in the following subsection.

Figure 4.4 shows an example of `IActiveSkill`. Compared to passive skill, it has three more parameters, that are common to all active skills – cooldown, mana cost, and stamina cost. It has also a list of effects that should be applied once the skill is used.

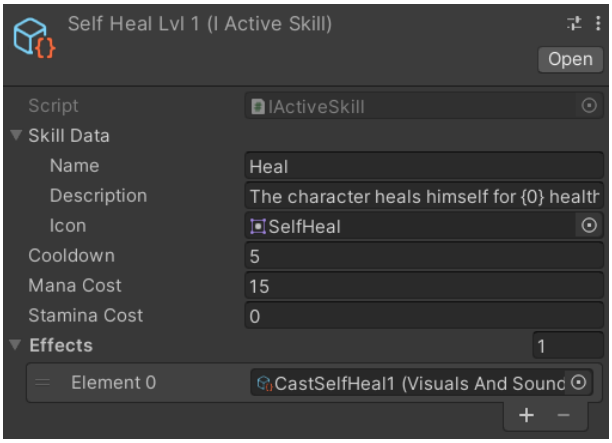

Figure 4.4: Active Skill

For dash skills, we need to create a `ScriptableObject` of the type `IDashSkill`. Aside from the active skill fields, it also contains parameters for the dash, such as the behavior of the dash, speed of the dash, and color of the dash (trail). In the behavior of the dash we can implement changes in the physics, such as a go-through dash, and also when the effects should be applied. Regular dash applies the effects on collision, and positive effect dash (used by

Healing Dash) when the character starts dashing. We can see an example of the `ScriptableObject` in figure 4.5.

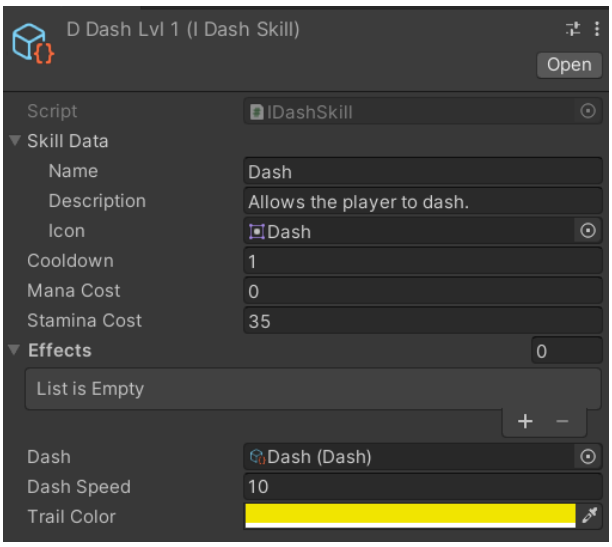

Figure 4.5: Dash

Secondary attacks are created using `ISecondaryAttack` `ScriptableObject`. Similarly to dash, it contains active skill fields and then fields specific for a secondary attack. Namely, the game object containing the implementation of the attack, and attack configuration. The game object is just a dummy game object with an attack component that can be then attached to the character. An attack configuration is a `ScriptableObject` holding all parameters related to an attack, such as range, damage, animation duration, and SFX. Attack configurations will be described in more detail in 5.2. An example of the `ISecondaryAttack` `ScriptableObject` is shown in figure 4.6.

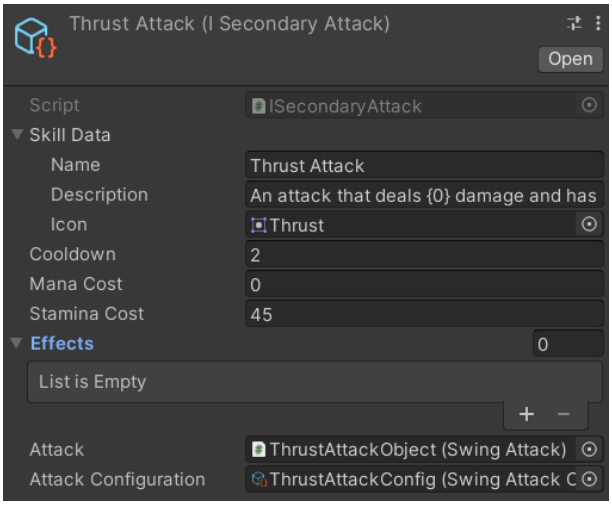

Figure 4.6: Secondary Attack

4.3.3 Skill Effects

`Skill Effects` are `ScriptableObjects` that implement parts of the behavior of the skills. All `ISkillEffect` have parameters related to targeting, the rest of the fields depend on the type of behavior. The idea behind this is that we can create different effects easily by simply changing the target.

We have three main types of `ISkillEffect` - stat effects, immediate effects, and VFX. They are described in the following sections.

Stat Effects

The `Stat Effects` are effects that increase or decrease the stats of their target/s. There are further divided into:

- One-time temporal effects
- Repeated temporal effects
- Regenerable stat buff
- Weapon Effect

One-time temporal effects are buffs and debuffs that are applied once and their effect holds for some duration. An example of this effect is an attack boost which increases the physical damage of the target for X seconds. Its `ScriptableObject` is shown in figure 4.7.

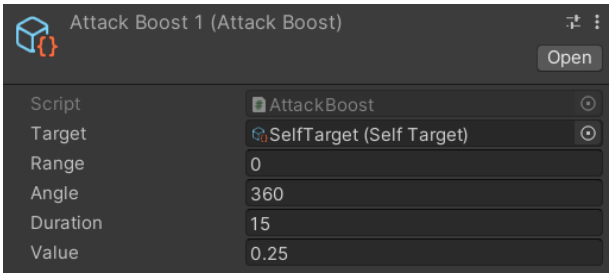

Figure 4.7: One Time Temporal Effect

Repeated temporal effects hold for X seconds and their effect is repeated every Y. An example of such an effect is regeneration or burn damage. We can see the `ScriptableObject` of burn effect in figure 4.8.

Regenerable stat buffs are one-time effects applied to regenerable stats. An example of this type of effect is the healing skill.

Weapon effects are temporal effects that are applied to the weapon of the target. An example of this effect is the flaming sword skill, whose `ScriptableObject` is shown in figure 4.9.

All stat effects are structured in the following way:

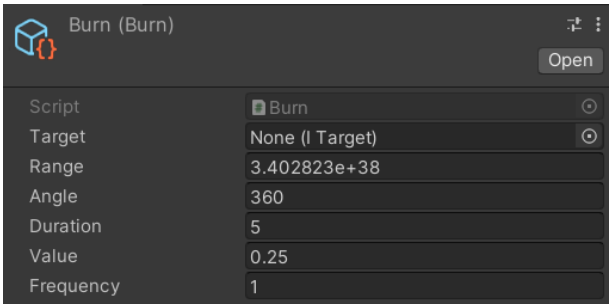

Figure 4.8: Repeated Effect

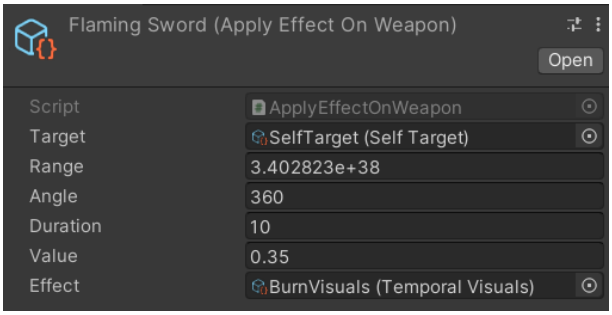

Figure 4.9: Weapon Effect

- Cast spell - an effect holding the visual and sound effects of casting a spell
  - Effect VFX - the temporal VFX of the effect
    - Stat Effect behavior - the skill effect holding the actual behavior, it can be of one of the types above

Immediate Effects

Immediate effects are applied straight away and they don't have any duration. These are examples of this type of effect:

- Deal Damage
- Push Back
- Projectile Skill

The deal damage effect, as the name suggests, deals immediate damage to the target. An example of its `ScriptableObject` is shown in figure 4.10.

The push-back effect applies force to the target units in the direction from the caster. As we can see in figure 4.11 the only parameter it has, besides targeting, is the force value.

The projectile effect creates a projectile at the caster's location and the projectile is then fired in the target's location. On impact, the projectile applies the specified effects. All the parameters are shown in figure 4.12.

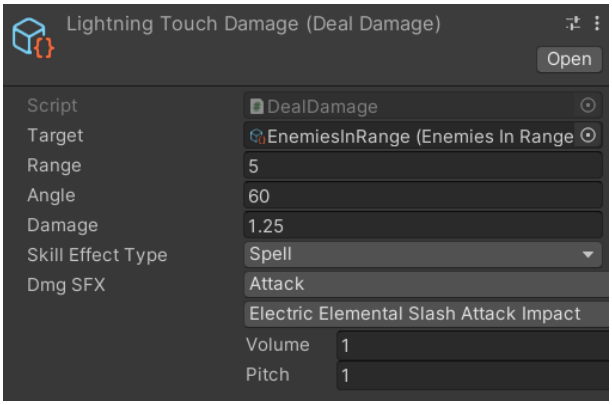

Figure 4.10: Deal Damage Effect

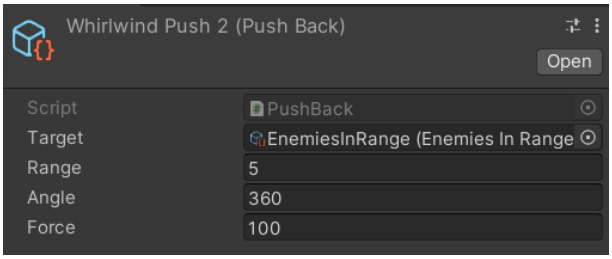

Figure 4.11: Push Effect

VFX

We can use skill effects also for displaying the VFX of the skills and playing the SFX. Their `ScriptableObjects` hold the data common for all skill effects and in addition also the prefab with the corresponding VFX. There are three types of skill effects used this way:

- Visuals and Sounds
- Temporal Visuals
- AoE Visual

Visuals and Sounds are used for the casting VFX of the skills and for the skills with immediate effects with no duration. The VFX of this type usually has a quite short duration as the other effects are applied only after the visual effect ends. All parameters of the visual and sounds effect are shown in figure 4.13.

Temporal Visuals are used for temporal stat effects and weapon effects as they have the same duration. VFX of this type are either looping or their duration is long enough to fit with the stat effect. Parameters of temporal visuals are shown in figure 4.14.

AoE Visuals are used for skills that apply some effect in a range. The VFX takes the range of the skill as a parameter to visualize the AoE correctly. An example of `ScriptableObject` is shown in figure 4.15.

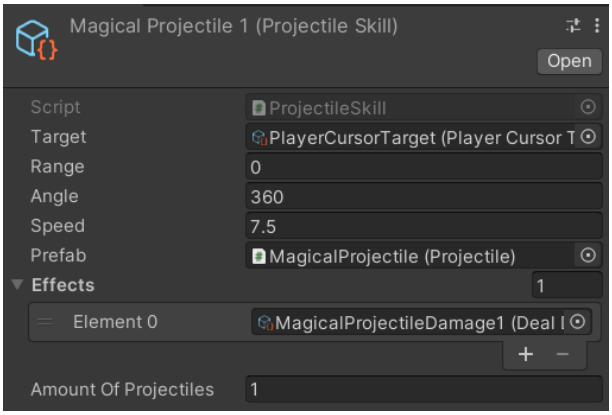

Figure 4.12: Projectile Effect

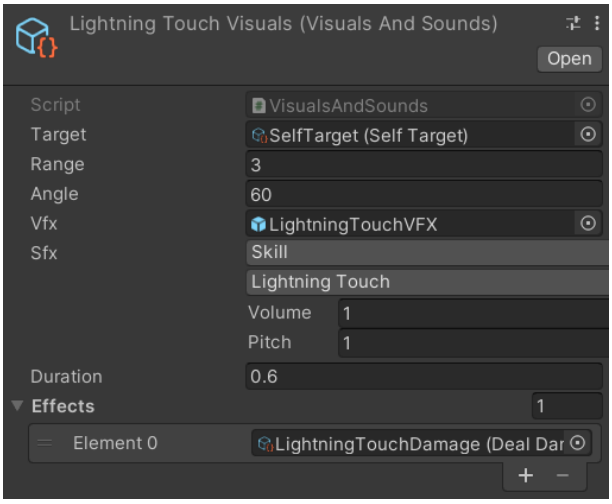

Figure 4.13: Visuals and Sounds

Target

The targeting information contains the `ScriptableObject` with the target, range, and angle. The range and angle are set to 0 and 360 respectively by default. They need to be reset only in case we are using the AoE target. Currently, there are the following targets:

- Self - applies the effect on the caster of the skill
- PlayerCursor - if the caster is an enemy, it applies the effect on the player's character. Otherwise, the effect targets the player's cursor. It is mainly used for ranged abilities.
- AlliesInRange/EnemiesInRange - the effect is applied on all allies/enemies of the caster that are in range. We can also specify the angle if we want to have an effect that is applied in a cone.

The target can be also specified by the parent effect. In that case, we leave

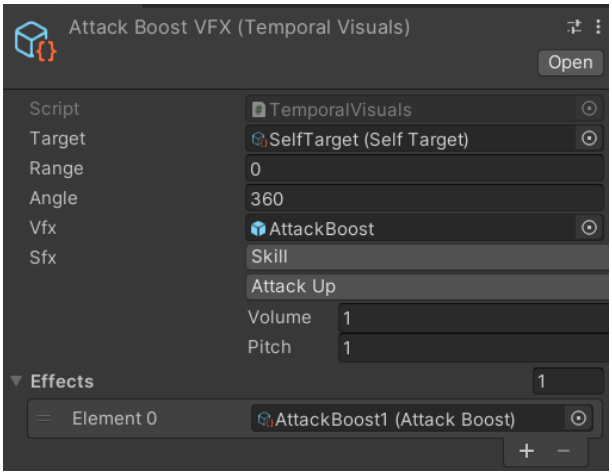

Figure 4.14: Temporal Visuals

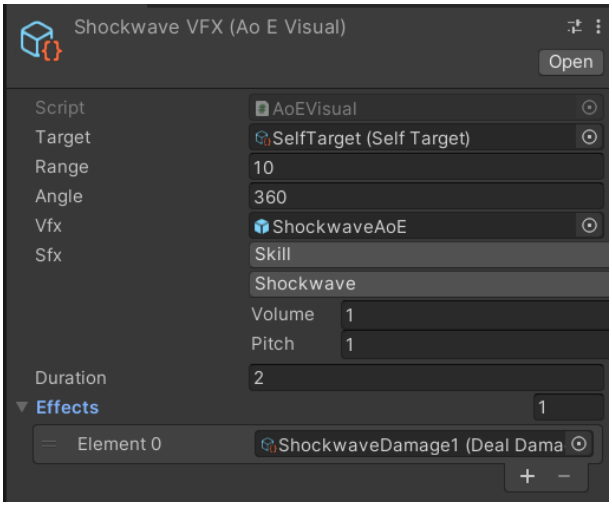

Figure 4.15: AoE Visuals

the targeting data empty. An example of this usage is with ranged skills with which we want the target of the effect to be the unit the projectile collided with.

4.4 Skills Creation Tutorial

In this section, we will explain how to create new skills in Unity. We have already described the different types of `Scriptable Objects` that need to be created and their parameters in the previous section. Thus in this section, we will focus on other objects that need to be created and provide a step-by-step guide for creating new skills.

4.4.1 Skill VFX

We slightly touched the VFX of the skills when we were explaining the skill effects of type VFX. In this subsection, we will explain how to create the game object holding the visual effect.

Based on the type of visual effect we need, we create an empty game object with a corresponding script attached to it. For general VFX (Visuals and Sounds) we use the `VisualEffects` script, for temporal visuals we use the `TemporalVisualEffect`, and lastly, for the AoE visuals, we use the `AoEVisualEffect`. Another option is to simply duplicate the game object of the corresponding type and simply adjust the visuals.

General VFX

The general VFX are the simplest, as the script doesn't require any parameters. The game object with the script only needs a child object with a particle system attached to it. The duration of the VFX is determined by the skill effect and it is better to set up the particle system in the same way.

Temporal Visuals

Temporal visuals have the same structure as the general VFx, however, there are more settings available. Moreover, the child game object with a particle system is not obligatory, compared to the general visuals. They can work in one of three ways:

- Looping particles
- Change material of target (unit or weapon)
- Both of the above

In the first case, we set up the VFX in the same as in the case of general visuals. However, the particle system has to be looping in order to have the desired visual effect. This setting is used with the regeneration skill.

In the second case, we don't have the child game object, and we only use the parameters of the temporal visuals script. We need to set three parameters:

- Apply on Weapon - should the effect change the material of the weapon
- Default material - a reference to the default material (`Sprite-Lit-Default`)
- Effect material - a reference to the new material we created using the `Allin1Shader`, see more details in 9

In the third case, we simply combine the two above. We set up both the materials and the particle system. No other special settings are needed.

AoE Visuals

AoE visuals are also using the child game object with an attached particle system. Moreover, the particle system needs to be referenced in the parameters of the script, as it is adjusted by code (the size of the AoE as well as duration).

All the data are sent over from the skill effect, thus there is no additional setting needed in the game object.

4.4.2 Projectiles

Projectiles, similarly to visual effects, also need a standalone game object with the `Projectile` script attached to it.

Aside from the script, it also needs components related to the visuals of the projectile, namely `SpriteRendered` and optionally also `AllIn1Shader`. It also needs components related to physics, namely `CircleCollider2D` and `Rigidbody2D`.

In the inspector of the script, we then need to set up the SFX of the projectile, and also the delay parameter. The delay determines how long it will take to cast and shoot the projectile.

4.4.3 Step by Step

Active Skills

1. Write script for the behavior if needed, it is possible to simply reuse the effects that have been already implemented
2. Create corresponding skill info - either active, dash, or secondary attack
3. Create a skill object for each level of the skill
4. For each level of the skill create the skill effects needed
5. Create or reuse needed VFX, materials, or projectiles
6. Connect all the objects
7. Add the skill info to the corresponding list in the Skill System prefab
8. Create a new skill button of the corresponding type in the Character Sheet UI prefab

- Place it in the correct tier
- Set correctly the index of the skill in the list in the Skill System prefab

9. Remove all saved data in order to test the new skill

Passive Skills

1. In case of passive skills increasing some stat, you can simply write a new class extending the `IncreaseStat` class and just override the `ChangeStat(Stats stats, float val` function. Otherwise, create a script extending the `IPassiveSkill` class.
2. Create a `ScriptableObject` of `SkillInfoPassive` type
3. Create a `ScriptableObject` of type of the class you have written in the first step for each level of the skill you want
4. Attach the skills to the skill info object
5. Add the skill info to the passive skills list in the Skill System prefab
6. Create a new skill button of type `SkillButtonPassive` in the Character Sheet UI prefab
    - Place it in the correct tier
    - Set correctly the index of the skill in the list in the Skill System prefab
7. Remove all saved data in order to test the new skill

5. Enemies

In this chapter, we will describe the desired properties and behaviors of enemies, types of enemies and enemy attacks. We will also provide a list of the content that is currently in the game, content that can be easily created, guidelines for designing new enemies, and lastly a tutorial for creating new attacks and enemies in Unity.

5.1 Properties and Behaviors

In this section, we will provide an overview of the implemented behaviors and properties of the enemies. Some of the behaviors (mainly attacks) and properties (mainly enemy types) will be described in more detail in the following sections.

5.1.1 Properties

All enemies have the following properties:

- Power type
- Action type
- Stats

Power type determines the overall strength of the enemy, the loot the enemy is going to drop, as well as the amount and type of actions the enemy can do. We differentiate three power types — basic, elite, and boss. The individual power types are described in more detail in 5.3.

5.1.2 Behaviors

- Movement – all enemies can move in all directions
- Attacks – all enemies have at least one melee or ranged attack, they are described more in 5.2
- Skills – more powerful enemies have in addition to the basic attacks also sum skills, such as healing, summoning, or more advanced attacks
- AI behavior
  - All enemies are able to navigate through the environment

  - When they see the character or are close enough, they start to chase him
  - Enemies are able to choose between actions based on their cost and damage output

5.2 Attacks and Skills

We differentiate between three types of actions the enemy can do:

- Melee attack
- Ranged attack
- Special ability

All enemies can do at least one action. More powerful enemies can do more actions that also might differ in their type.

All actions have an indicator or an animation that should tell the player what action is the enemy going to do and what location is the enemy targeting.

5.2.1 Melee Attacks

All melee attacks have an area of effect as an indicator that is shown on the ground. If the character is in the area during the attack, they get hit. The size and shape of the area depend on the weapon and type of attack the enemy is using.

Melee attacks also have a technical animation showing the movement of the weapon. The movement starts at the starting point and moves alongside or towards the indicator.

Currently, there are the following melee attacks:

- Swing
- Thrust
- Smash
- Stomp

Swing Attack

A swing attack is an attack with quite a low range and it is considered to be the most basic attack as most of its parameters are average. Its indicator is a sector of a circle, whose angle, and radius can be configured for each enemy.

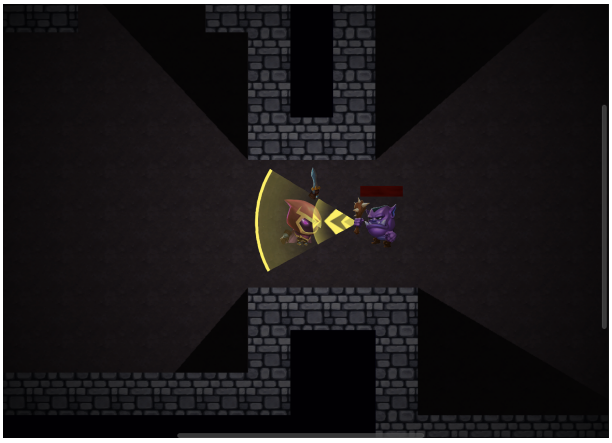

Figure 5.1: Swing Attack

Thrust

A thrust is a fast attack with a larger range but the width of the AoE is way lower than in the case of the swing attack. Its indicator is also a sector of a circle.

Smash

A smash attack is a slow attack with a medium to low range, a small AoE, and high damage. Its indicator is a circle located in the range of the enemy. The enemy moves the weapon to a starting position that is approximately 90 degrees from the target position, and then quickly moves the weapon toward the target.

Figure 5.2: Smash Attack

Stomp

The enemy strongly hits the ground causing large damage in a small area of the stomp and lower damage in a larger area. The AoE is indicated by two concentric circles of different sizes.

Figure 5.3: Stomp Attack

5.2.2 Ranged Attacks

Ranged attacks are indicated mainly by a technical animation of the projectile or multiple projectiles appearing (in case of magical ranged attacks) or by a movement of the hand and weapon (in case of throwing weapons).

There are no other indicators as projectiles always behave the same — they are directly targeting the player's character.

Ranged attacks can differ in cadency of shooting, amount of projectiles, and in the projectiles themselves. Projectiles can either apply some effects on the target they hit (such as poison), cause an explosion on impact, or follow its target.

Special Abilities

Under special abilities are all other actions the enemies can do besides the basic attacks that were described above. Actions, such as special attacks (dash), summoning minions, or healing allies belong there.

Currently, there are the following special abilities implemented for the enemies:

- Dash — the enemy dashes towards the player's character and deals damage on impact. The dash is indicated by an arrow-shaped indicator.
- Summon — the enemy summons several minions of predetermined type next to him. The minions then behave as regular enemies.
- Heal — the enemy heals himself and all allies that are in range.

The rest of the enemies are using either melee or ranged attacks. The more powerful enemies have these attacks with higher stats or they apply or have some additional effect.

Special abilities of enemies can reuse parts of the skill system of the character, therefore it would be easy to create additional content.

5.2.3 Creating New Attacks

In this subsection, we will explain how you can create new attacks in Unity, what objects need to be created, what values need to be set, and what those values mean.

Steps to create a new Attack:

1. An attack `IAttack` can be of the type - `MeleeAttack`, `RangedAttack` or special attacks `SummonAttack`, `HealAttack`.
2. Add the respective `IAttack` component as per the requirements to the `GameObject` *Arm* (There can be more than one `IAttack` per enemy).
3. Create the respective `ScriptableObject` `AttackConfiguration` of each `IAttack`.
4. Set the values of the variables exposed in the `ScriptableObject` and then assign the `ScriptableObject` to the `IAttack` component.

5.3 Enemy Types

We distinguish between three enemy types — basic, elite, and boss enemies. The types differ in their stats, as each power type has its own stat modifier. More powerful enemies can also do more than one action and quite often they can do either special abilities or more advanced versions of basic attacks.

5.3.1 Basic

Basic enemies are the weakest type of enemies. They can only do one action, either melee or ranged attack. They are also the most often ones found in the dungeon.

Currently, there are the following basic enemies found in the game:

- Orc Fighter – melee enemy with swing attack found in Orc Camp dungeon
- Skeleton Fighter – melee enemy with short ranged swing attack found in Tombs of the Undead dungeon
- Poltergeist – melee enemy with short-ranged magical attack found in Tombs of the Undead dungeon

- Imp – melee enemy with short ranged higher damage attack found in the Demon Pits dungeon
- Stone Elemental – melee enemy with smash attack found in the Demon Pits dungeon

5.3.2 Elite

Elite enemies have better statistics than basic enemies. Mainly they can do more different actions (usually two), either from the same type of actions or even from different ones. They are slightly rarer than basic enemies and they are also more visually unique.

There are the following elite enemies in the game:

- Orc Chieftain – melee enemy with swing and smash attacks found in the Orc Camp dungeon
- Orc Shaman – has both a melee (swing) and a ranged attack (regular projectile), and is found in the Orc Camp dungeon
- Skeleton Warrior – melee enemy with swing and thrust attack found in the Tombs of the Undead dungeon
- Wraith – melee enemy with swing and thrust attack that applies poison found in the Tombs of the Undead
- Electric Elemental – has both a melee (swing) and ranged attack, and is found in the Demon Pits dungeon
- Dark Elemental – fast ranged enemy, that has also a special attack (dash), and is found in the Demon Pits dungeon

5.3.3 Boss

Bosses appear only once in a dungeon and for each dungeon they are unique. They have a room that is designed specifically for them (usually a large room without obstacles, so the boss can move around freely). They can do multiple actions of different types. They also have way more health than all other enemies.

There are the following boss enemies in the game:

- Giant – the first boss located in the Orc Camp dungeon, the boss is strictly melee as it has a swing, stomp, and smash attack

- Lich King – the second boss located in the Tombs of the Undead dungeon, the boss has a ranged attack with a follow projectile, he can summon skeletons, and heal himself and all skeletons in range
- Demon – the last boss located in the Demon Pits dungeon, he has two melee attacks (swing and stomp) that apply the burn effect and a ranged attack that uses the following projectile

5.4 Enemy Creation

In this section, we provide a step-by-step guide for creating new enemies in Unity.

Steps to create a new Enemy:

1. Duplicate an already existing enemy `prefab`.
2. Rename the *Root* of the `prefab`.
3. Replace the sprite of the `GameObject` *Enemy* to match the new enemy's body.
4. Replace the sprite of the `GameObject` *Asset* to match the new enemy's weapon hand.
5. Enable the `SpriteRenderer` of the `GameObject` *Arm*. This is only meant to aid the weapon placement and should be disabled after the final step.
6. Adjust the `Rotation, Position` of the `GameObject` *Asset* as well as the `Rotation` of the `GameObject` *Arm* until it seems correct with respect to the body.
7. Adjust the `Z` value of the `Rotation` of the *Asset* and note down the values when it aligns with the Y-Axis (`UprightAngle`) and when it aligns with the *Arm* sprite (`ArmAlignAngle`). Copy these values in the exposed variables of the `Weapon` component.
8. Adjust the position of the `GameObject` *Trail*. Copy-paste its `Position` value in the `WeaponTipOffset` variable of the `Weapon` component.
9. Create an `AnimatorController` by the same name as the *Root* and place it in a folder by the same name Assets/Animations.
10. Replace the `AnimatorController` of the `GameObject` *Enemy* with the newly created one.

11. Create two sprite sheet animations - $Idle, Walk$ in the same folder and assign them to the `AnimatorController`.

12. Add / Modify the `IAttack` components corresponding to the type of attacks used by the enemy.

13. Create new `Scriptable Objects` for `Stats` for each level of the enemy that is going to be used.

14. Modify the values of the variables exposed in the `Enemy` component such as `Stats`, `Movement`, `Effects`.

Currently, each enemy has unique sound effects for ambient, attacks, take damage and death. These SFX can be set either directly in the enemy, or in the actions (attacks) he is using. How to create new sound effects is described in 8.

Moreover, new enemies should be added to the spreadsheet that is used for balancing. Values in the spreadsheet should always correspond to the values that are in the game.

6. Dungeon

Each dungeon is thematically different, the theme is determined by the enemies that are in the dungeon. Moreover, each dungeon has its own unique boss.

The dungeons should be also slightly visually different, this is provided by the decorations that are used, and also how light or dark the dungeon is.

6.1 Dungeon Structure

The dungeons have a structure of a spanning tree with a few extra paths from the starting point to the boss room. There are also several dead ends that can be used for grinding.

All dungeons, with the exception of the Tutorial Dungeon, have a very similar size, as the size can be preset in the dungeon generator.

Rooms that can be found in the dungeon differ both in size, shape, and purpose. Namely, we are talking about the following:

- Boss room – a large room with one entry point used to accommodate the boss. Its size and shape are hardcoded in the generator so it looks the same across all dungeons. The entry point to the boss room has unique visuals and players need to interact with it by pressing the F key in order to enter. Therefore it is impossible to enter the boss's room by accident.
- Caves – longish irregularly shaped rooms that are usually quite big compared to the other rooms
- Checkpoint room – small squarish rooms with the checkpoint object
- Regular room – medium-sized squarish rooms with enemies inside

With our current level design, we are attempting to have dungeons containing both caves and rooms. We are also trying to avoid a high amount of very long corridors. There should be multiple possible paths from the starting point to the boss room. The shortest path shouldn't be too short and straightforward, as we want to promote exploration.

6.2 Minimap

The player does not see the whole dungeon, and overall his visibility is quite limited. However, the player can use the help of minimap, which shows the

visited checkpoints and their immediate area. The player can open the minimap by pressing the M key, afterward, he sees the following:

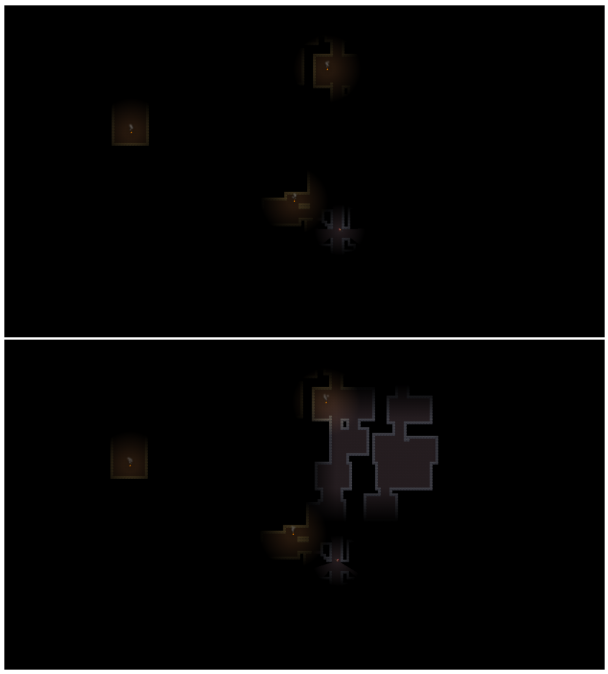

Figure 6.1: Minimap

Moreover, the player can also collect map fragments. Map fragments are objects placed around the dungeon, that reveal a larger part of the dungeon on the minimap.

The player can see only the layout of the dungeon and the placement of checkpoints on the map, enemies and other dungeon objects are not visible. However, even with this information, it should be easier for the player to navigate toward the boss rooms or to areas where they can grind, based on their needs at the moment.

6.3 Dungeon Objects

Dungeon objects are supposed to make the exploration more interesting and rewarding for the player. By dungeon objects we mean objects the player can directly interact with, there are also lots of decorations that will be described in the next section.

The most important dungeon object is the checkpoint. Visiting the checkpoint saves the current progress and shows the player a UI screen where they can level up their character and learn and assign skills. Moreover, resting at a checkpoint respawns all enemies and replenishes the health, mana, stamina, and armor of the player's character. This mechanic thus allows the player to grind if they need

to. Usually, there are 4 or 5 checkpoints located throughout the dungeon. One checkpoint is always at the starting point and another one is near the entrance to the boss room.

Figure 6.2: Checkpoint

There are also dungeon objects that deal damage to the player when they step into them. Namely, trap and lava lake. We can set the damage individually for each trap or lava lake. With traps, we can also set the delay and reload time.

The trap is triggered when someone steps into it and there is a small delay before it deals damage. Moreover, there is also a slight delay in triggering it again. Thus the player can either walk around it, use the dash to get over it, or bait the trap and then walk over it before it gets triggered again.

Figure 6.3: Traps

The lava lakes, on the other hand, start dealing damage immediately after the player's character steps into them and they deal continuous damage every second while the character is inside them. They are located in the last boss room.

Exploration is supposed to be an important part of the game. Therefore, it was important to add some mechanics that would reward the player for it. One of the objects are the map fragments that we already mentioned in the previous section. Next are the chests, which give the player XP which can be otherwise

obtained only by killing enemies. The last reward object is a reset book, which allows the player to reset invested skill points. All these objects are not respawned when the player rests at a checkpoint.

Figure 6.4: Dungeon Objects

Crystals give the player mana and an HP boost. The size of the bonus depends on the dungeon and on the size of the crystal. They are respawned when the player rests at a checkpoint.

Boxes are obstacles that can be destroyed by both basic attacks and spells. Destroyed boxes are respawned when the player rests at a checkpoint.

Fog separates the boss room from the rest of the dungeon. It prevents the boss from escaping the room as well as the player from wandering in by an accident. In order to enter the boss room the player needs to actively interact with the fog. Once the player is in the boss room, they cannot return back to the dungeon.

6.4 Dungeon Visuals

The dungeon is drawn on a grid with tiles that are approximatelly twice the size of the player's character. There are several layers of tilemaps in which are the dungeons generated, namely:

- Ground
- Walls – layer containing the walls of the dungeon, it has a collider
- Decorations – layer used for decorations the player cannot interact with, such as moss, mushrooms, or plants
- Rocks – layer containing objects that cast shadows and blocks path
- Torches

All these layers are common for all dungeons as well as the assets for ground, walls, rocks, torches, and basic decorations (mushrooms, plants, and moss).

There are also decorations that are specific to certain dungeons. These decorations have to be placed manually after the dungeon is generated. They should be drawn into the Decorations tilemap layer.

In the case of the Tombs of the Undead dungeon, the specific decorations are bones and tombstones. Bones can be used also in other dungeons, but they are used in this dungeon the most.

The light of the dungeon is provided by four types of sources — the character, the torches, the checkpoints, and general ambient light. The color, range, and intensity of the torches and checkpoints can be set in the corresponding objects in the `Light 2D` component.

The ambient light can be adjusted for each dungeon separately in `Ambient Light` game object, again in `Light 2D` component. Currently, the intensity of the ambient light is slightly decreasing with each dungeon.

6.5 Dungeon Generation

The layout of the dungeon as well as basic decorations are procedurally generated. The algorithm outputs all the tilemaps in separate layers so they can be further manually edited.

However, since the number of tiles for the whole dungeon is really high, we would recommend doing only minor changes, mainly in the decorations or torches layers.

These are the steps for generating a new dungeon:

1. Go to the `Dungeon Generation` scene and hit play
2. Press `G` key to generate a dungeon until you are happy with the result
3. Press `E` key to save the dungeon into tilemaps
4. Drag and drop the object with tilemaps into prefabs
5. Exit the play mode and start again with step 1 to generate another dungeon

6.6 Dungeon Customization

The generator generates only the tilemaps and dungeon decorations. All enemies and dungeon objects need to be placed into the dungeon manually. In order to ensure lucidity in the scene hierarchy, all objects should be placed under their own corresponding empty `GameObject`.

We are trying to have a variety of encounters. Usually, at the beginning of the dungeon, there are one on one encounters against basic enemies. Further in the dungeon, we are adding more enemies and we are increasing their power type

and level. There should not be more than 3 enemies near each other. Enemies should not be placed near checkpoints.

There are usually 3 to 4 checkpoints in the dungeon. One checkpoint is always located at the starting position and another one nearby the boss room. The remaining checkpoints are usually placed along the path to the boss.

The position of Map Fragments matters as they reveal the dungeon around their position. Usually, there are 2 or 3 map fragments in the dungeon.

The damage of each trap can be set in the `Spikes` object under a component of the same name. The number and placement of the traps depend on the dungeon. Usually, they are placed in corridors and entrances to rooms.

Crystals are very similar to traps – they are placed under their object and have adjustable values. They are usually placed in front of boss room to provide a boost before the final fight.

All objects that should cast shadows, such as chests or boxes, should be placed under the Shadow Generator `GameObject`. Chests can be further customized in terms of the loot they are supposed to drop, the amount, and the values.

When all objects are placed, you need to bake shadows and assign IDs. The shadows are baked in the Shadow Generator `GameObject` using the corresponding button. You can assign the IDs in the top navigation bar, under the `Dungeon` tab.

In each dungeon, you can also adjust the intensity and color of the ambient light. This can be done in the Ambient Light `GameObject`. Currently, we decrease the intensity with each dungeon.

7. UI

In this chapter, we will just quickly go over the UI screens that are in the game and show their layout and components.

Main Menu is the first scene the player can see. From the Main Menu, they can either start a new game or load an existing one. They can also open sound settings and controls from this scene.

In the Load Game scene, the player can load an existing game or start a new one by selecting an empty slot. There is no Save Game counterpart as the game is saved automatically at checkpoints.

The Settings can be opened either in the Main Menu or anytime in the game by pressing `ESC` key. The player can adjust the sound settings there and also see the controls. However, the controls are not customizable.

The Game Won scene appears when the player successfully finishes the last dungeon. It allows the player to navigate back to the Main Menu.

The in-game UI is always visible during the game and it gives the player information about his character and about the enemies. In the top-left corner are the health bar, mana bar, stamina bar, and icons for status effects (such as attack up). In the middle bottom are shown the skills the player has equipped and their cooldowns. Above the skills is the XP bar and armor bar. The armor bar is visible only when the character has more than 0 armor.

The last part of UI is the Character Sheet which allows the player to level up their character, increase their stats and learn and equip new skills. We have described this UI screen in more detail in 4.2.

8. Sounds

In this chapter, we will describe the approach to SFX and soundtrack and their creation both inside and outside of Unity.

8.1 SFX

All SFX are held in the Sound Manager `Game Object`. The sound effects are divided into several categories based on their type. Each category then has its own pool of audio sources. Therefore the sounds are not directional.

We are using the following categories of sounds:

- Looping – all sounds that are supposed to loop need to be placed in this category, as sounds from this category are handled differently.
- Enemy Ambients – enemy ambients start playing when the player's character is close enough and stop playing during fights. As ambients, we usually use various footsteps, grunts... There are unique for each enemy.
- Attack
- Skill
- UI
- Items – in this category are mainly item pickup sounds
- Deaths
- Take Damage

8.1.1 Sound Creation

1. Find the desired sound or sounds on pages with SFX assets (freesound, zapsplat...)
2. Compulsory edits
    - Trim the beginning of the audio clip so the sound starts playing immediately
    - Trim the end of the audio clip so the sound does not play for too long
    - Normalize the loudness of the sound to -30 LUFs

3. Optional edits
    - Mix more sounds together
    - Change the pitch of the sound
    - Change the speed of the sound so it fits animation timing

8.1.2 SFX in Unity

1. Add the asset to the correct folder in the project
2. Add the SFX to the corresponding category in the prefab of the Sound Manager object
3. Add the SFX to the corresponding category in the `SoundCategories` script
4. Adjust the volume and pitch of the sound in the prefab of the Sound Manager object
5. Bind the SFX to its event (such as using a skill or an attack)
    - Add `SerializeField` Sound Settings to corresponding `GameObject` or `ScriptableObject`
    - Select the correct Sound Setting in the `SerializeField`
    - Call the `PlaySound` function at a correct time in the code
    - Optionally, there can be added functionality for changing the volume of the sound based on the distance

8.2 Soundtrack

We distinguish between two types of soundtracks – regular soundtrack and boss soundtrack. The boss soundtrack is unique for each boss and it starts playing once the player's character enters the corresponding boss room. Otherwise, there are no differences between the two soundtrack types.

8.2.1 Soundtrack Creation in AIVA

The soundtracks were created using AIVA which is an AI music generator. You can either use the editor in the browser or download their free application. For non-commercial purposes, the usage of AIVA is completely free of charge.

In AIVA you can use their built-in templates, generation profiles, or existing compositions, to influence the outcome. You can also create your own generation profiles or edit the existing ones.

After you select the template or generation profile, you can choose the key signature and duration of the composition. The majority of our soundtracks are in Ab major or its relative key F minor. The only exception is in A minor.

Generation Profiles

A generation profile is a set of settings for musical properties (such as dynamic range, tempo range, harmony, or chords) and instruments (such as bass or percussion).

We have created two generation profiles for the purpose of our soundtracks – Epic Boss Fight and Dark Epic Orchestra. As the name suggests, the Epic Boss Fight is used exclusively for the boss soundtracks. For the rest of the soundtracks, we used the Dark Epic Orchestra profile or the general Fantasy template.

8.2.2 Adding Soundtrack to Unity

Once the soundtracks are downloaded from AIVA, they need to be edited to have a normalized loudness. For soundtracks, we used -40 LUFs, as they are going to be played all the time and we don't want them to overpower the SFX.

When this is done, they need to be uploaded into the `Assets/Sounds/Music` folder. Afterward, the assets need to be dropped into the `Soundtrack` prefab, into a component of the same name.

Boss soundtracks belong to the `BossMusic` list, and the rest of the soundtracks to the `Music` list. No other setting is needed.

9. Visual Guidelines

All assets are visually done in 2D. They are supposed to have a dark fantasy mood with some glowing elements for contrast. Moreover, the assets should look somewhat smooth and clean.

9.1 Character and Enemies

All character and enemy assets need to be split into two parts:

- Head + body + legs + one arm
- Arm + weapon

This split is necessary for technical animations related to attacks and the general movement of the weapon. Otherwise, it would not be possible to move with the weapon in all directions.

Character and enemies should be looking slightly in one direction – all in the same. Thus we will be able to simply flip them in Unity when needed.

The size and amount of details of the enemies should differ based on their power type, with bosses being the biggest and most polished enemies.

Only walking animations are needed for character and enemies, as the rest of them is done as technical animations or are substituted by VFX.

9.2 VFX

Visual effects are done either by Unity `ParticleSystem` or by `AllIn1Shader`. We are using them mainly to visualize skills and their effects. VFX is also used instead of death animations and for polishing the environment (dust particles, destroyed boxes...).

The only disadvantage of the `AllIn1Shader` is that it replaces the `Sprite-Lit` material that works well with the shadows of the game. However, it allows us to easily do effects such as glow, various distortions, outlines, wave movements, etc... More information about the shader and its use cases can be found in its documentation.

10. Playtesting

Throughout the development cycle, the game was playtested several times both internally and externally. In this chapter, we will describe how the game can be easily tested internally, and the methodology we used for the external playtest.

10.1 Internal Playtest

In the early stages of the development, we were using internal playtests to test out the fun aspect of the core mechanics. In the early prototype, we focused mainly on combat and the mechanic of telegraphed attacks. These playtests helped us to fine-tune the number of enemies in one encounter, the sizes of the rooms, and initial values.

Later, we used the internal playtests for QA and balancing of the game. The game was QAd also externally. For QA, we made use of cheats, that are available only in the editor. There are two cheats available:

- `C` – goes to the next level
- `L` – gives the player 1000 XP

The cheats are implemented in the `Update()` function of the `GameController` script. Further cheats should be added there as well.

10.2 External Playtest

Aside from the external QA, we conducted one qualitative external playtest near the end of the development. For the playtest we selected 6 of our friends who are at least mid-core gamers. This requirement was necessary since the game is targeted at mid-core to hard-core audience, and players experienced with the souls-like genre. Therefore it would not be enjoyable for more casual players.

The main goal of the playtest was to determine the understandability of mechanics and UI and the difficulty of the whole game. Our secondary goal was to find bugs and overall areas where the game is lacking and can be further improved.

The playtesters were instructed to play the game as long as they enjoy it and narrate issues they encounter during the game. They weren't provided with any other information to simulate the experience of real players. During the session, the playtesters were silently observed by one of the developers. After the end of the session, the playtesters were questioned regarding the tutorials, the difficulty

of the game, the enjoyability of skills, and the intuitiveness of core mechanics such as stamina, loot, attack indicators, and skills.

The average length of a session was 1 hour and the majority of the players managed to beat the first two dungeons (Tutorial and Orc Camp) during that time. The game was rated as difficult but quite enjoyable. The game was played longer and enjoyed more by players who played games from the souls-like genre.

Based on the feedback from the external playtest and feedback given to us by the supervisor of this project we did the following changes:

- Improving the tutorials
  - Attack indicators
  - Actions costing stamina
  - Checkpoints respawning enemies
  - Boss room
- Improving UI/UX of the Character Sheet
  - Removing primary stats
  - Making the level-up button more prominent
  - Adding permanent instructions regarding unlocking skills
  - Adding level to the skill screen
  - Adding tooltips to stats
  - Adding stat used for scaling to skill description
- Redesign of the stamina mechanics
  - Decreasing costs of actions
  - Decreasing speed of stamina regeneration
  - Adding delay to regeneration after using an action
- Decreasing difficulty of the game
  - Adding weaker enemy at the beginning of the tutorial dungeon
  - Decreasing speed of enemy attacks
  - Decreasing damage of enemy attacks
- Traps deal damage to enemies
- Decreasing cast time of spells

- Visualizing better the armor
- Highlighting the current position on minimap
- Minor bug fixing