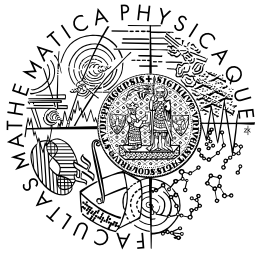


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Tomáš Guth Jarkovský

Adaptive generated encounters in a rogue-like role-playing video game

Department of Software and Computer Science Education

Supervisor of the bachelor thesis: Mgr. Vojtěch Černý

Study programme: Computer Science

Study branch: Computer Graphics and Game Development

Prague 2023

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

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

I wish to greatly thank my supervisor Vojtěch Černý for so many consultations and advice that I could not do without. I would also want to thank all the people in my life whom I have neglected due to my studies.

Title: Adaptive generated encounters in a rogue-like role-playing video game

Author: Tomáš Guth Jarkovský

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Vojtěch Černý, department

Abstract: We want to provide a video game player with a fun, unique, challenging experience. That may not be the case in games involving a lot of possibilities or randomness. This thesis introduces an algorithm for a procedural generation of enemies in a rogue-like RPG game. The algorithm is based on running a series of simulated battles to create an adequately difficult enemy group. We have implemented the algorithm in a custom rogue-like turn-based RPG game and in the experimental part, our approach has shown to be moderately successful. The generated enemies have shown to be neither too difficult nor too easy while providing a reasonable amount of variety and new challenges. The outcome of this thesis may be a step forward in the generation of unique, fun, and balanced enemy encounters in rogue-like RPG games.

Keywords: procedural generation games RPG difficulty

Contents

Introduction

Rogue-like games are games focusing on progress through an increasingly difficult dungeon. The focus is on getting the most effective combination of gear and abilities to progress through the game. Such games also usually involve a variation of an "endless run" mode with constantly increasing difficulty with the goal of getting the furthest. In a rogue-like game, a defeat usually means resetting to back where you started and starting anew.

When playing the game, the player commonly encounters groups of enemies. Fights with such enemies are usually turn-based with multiple characters in both players and the enemy party. Examples of rogue-like games are Darkest dungeon, Slay the Spire and Hades.

Players enjoy rogue-like games for the thrill of a challenge and the aim of getting better at progressing through the game. Each playthrough must be similar enough to allow improvement of skills and knowledge of the game but at the same time variable enough to not get repetitive and stale. The difficulty of a playthrough should gradually increase so that the player's improvement over several playthroughs is possible and can easily be recognized by both the player and the game.

Procedural generation is a process of creating content by an algorithm, rather than manually. Procedural generation allows to save development time and provides the possibility to constantly produce new non-repeating content. Procedural generation has been successfully used for creating massive worlds and thousands of levels but it has not yet been widely used to generate enemy encounters and to influence the game's difficulty. Procedural generation is often used with no or very little regard to the player's previous actions and current progress, such as in games Spore or No Man's sky.

In this thesis, we will design an algorithm for the adaptive generation of enemies (further called the "AGE algorithm"). The AGE algorithm's task is to generate adaptive, unique, fun and adequately difficult enemy encounters for a turn-based rogue-like game. Making the encounters adaptive means that the group of enemies is generated considering the weak and strong sides of the player's party. Making the encounters adequately difficult means making them require some strategy and decision-making while not making them too punishing. Such automation allows to save time and resources on designing levels and enemies and makes it possible to easily design an "endless run" variant of the game.

The main goal of the thesis is to measure whether enemy encounters generated by the AGE algorithm are similarly fun, difficult, and unique as manually designed encounters and whether the player has recognized the lack of human involvement in their design. Such findings may be key in deciding whether a game designer wants to include adaptively generated enemy encounters in their game. A successful outcome may also be a stepping stone towards expanding the amount and variety of procedurally generated content in games.

The thesis is divided into four sections. The first section is about the topic of, role-playing rogue-like games, procedural generation, the current progress in the field, and some related work. The second section is about a custom game that we have designed to test out our approach and about a component system crucial for the algorithm itself. In the second section, we also explore the practicalities of implementing our simulation-based approach into our game. The third section is dedicated to the experimental part, testing players' enjoyment and rating of their playthrough of Deep Crawl with procedurally generated enemies.

1. Background

In this chapter, we need to define several terms which we will be using throughout this work. As this work is focused on procedural generation and the enjoyment and difficulty of video games, the defined terms will be from these areas.

1.1 Procedural content generation

Procedural content generation (further abbreviated as PCG) refers to a process of content creation, where only a limited amount of parameters are set up by a human, and the rest of the work is performed by an algorithm, usually making use of randomness for creating unique pieces of content. [1]

The word "content" refers to any asset that players may encounter with their senses within the game. As stated in a book *Procedural content generation* [1], one of the widely known sources on the subject "In our definition, content is most of what is contained in a game: levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles, characters, etc."

1.2 Video games

Video games are a piece of new media carrying entertainment and artistic value. Video games have been around for more than 50 years [2], but only recently have they been normalized and enjoyed by most of the population in several demographic groups. In the US, a majority of all adults play video games regularly, with the number reaching 80% with the younger population. [3] Further in the text, whenever we refer to "games" we are going to be talking about video games.

Famous video game designer Sid Meirs presents a very fitting definition "Game is a series of meaningful choices", which appropriately describes the amount of variety within the video game genre, which varies from text-based adventure stories across large-scale battles fantasy fights counting hundreds of players to a simulation of presenting the opportunity to experience being a sentient mobile bread.

1.3 Role-playing rogue-like video games

The term role-playing has been a subject to many interpretations and meanings, but for the purpose of this text, we shall define a role-playing game (further abbreviated as RPG) as one, where a player controls one or multiple characters, is expected to make meaningful choices about the progression of their characters and such choices have an impact on the gameplay following such choice. An example of an RPG is World of Warcraft.

In an RPG, the player is expected to create a connection to their characters and to gain enjoyment from experiencing the result of their choices usually in a form of victory or further progress. In most RPGs, the progress and state of one's character are characterized by their equipment, or by the state of their skills and available abilities.

The term rogue-like game refers to a type of game, in which player repeatedly encounters similar scenarios and is expected to improve upon every iteration, either in their expertise in mastering the game's rules and logic or by improving their character to a point in which the player is able to progress further into the game.

An example of a rogue-like role-playing game is Darkest Dungeon, in which the player controls a roster of fantasy-inspired characters with different skills and possible gear options, and repeatedly ventures into dungeons, crypts, and similar locations to hunt enemies, gather better gear and loot, and to gather experience for their characters, so they may take on harder challenges and dungeons later on.

1.4 PCG in video games

In video games, PCG is commonly used to generate content which would be too time-consuming and money-consuming if it were done manually by a game designer or a 3D artist. The most common use case is generating landscape and nature. The randomness and chaos usually following PCG are perfect for automating the creation of nature and of landscapes.

This case can be most clearly seen in the game Minecraft, in which the entire world is procedurally generated, merely following predetermined constraints with the aim to create a pseudo-realistic world made entirely of blocks. The second example is the game No man's sky, in which the player is a space explorer with the ability and task to explore plenty of procedurally generated planets, al-

together with generated biomes, floras, and faunas, reaching hundreds of millions of possible combinations.

In this work, we are going to focus on the procedural generation of enemies and enemy encounters for a rogue-like role-playing game. Such a task brings a new challenge due to the fact that the generated content must not only be acceptable and unique but also adequately difficult and enjoyable to interact with - in this case, enjoyable to defeat.

2. Related work

Not much academic work has been done in the field of PCG of enemies and using PCG to balance the difficulty of a game.

There have been experiments related to balancing the difficulty of a game using PCG, but the specific research[4] used procedural generation for determining the number and formations of swarms of enemies, not the design and parameters of the enemies themselves.

There has been a great deal of work related to the PCG of maps and levels and generating quests, but not a lot regarding the generation of enemies.

One of the first games to ever use PCG was the game Rogue [5], involving PCG in generating game levels following a set of rules. A secondary major breakthrough in the history of PCG in games is Minecraft [6], which uses procedural noise to generate the entirety of the game world including large mountain ranges and deep cave systems below ground, following a complex set of rules and possible interactions between generated objects and biomes.

There have, however, been related breakthroughs in the commercial video gaming industry. In the game, No Man's Sky, the great majority of the explorable game universe is procedurally generated, including planets and their fauna and flora [7]. A very similar thing can be seen to a lesser extent in an older game from 2008, Spore [8]. However, in both these games, the PCG of enemies is used mostly for visuals, models, and behavior, rather than their difficulty and the player's ability to defeat them, so for the purpose of this thesis, we will not consider that as a PCG of enemies.

Matouš Kozma [9] has written a thesis regarding the PCG of enemy encounters in a similar style of games, although with a very different approach. His approach consisted of precalculating a difficulty matrix, with columns and rows being indexed by whole creature groups and each cell describing the difficulty of the encounter of the two groups. When trying to assess the difficulty of an encounter, his algorithm tries to find the most similar encounter and estimates the difficulty accordingly.

This approach brings several advantages and disadvantages. For one, trying to assess the similarity between two groups requires a reliable comparative function. Further on, calculating such a difficulty matrix may require a lot of computation beforehand and any future change to the game's mechanics requires

the new calculation of the whole matrix. With the used approach, the calculated difficulty matrix might also take up a lot of space, and such required space grows exponentially with more content and features in the game.

One of the key advantages of that approach is not require a lot of calculations during the player's game. A secondary advantage is that the accuracy of the difficulty matrix can be incrementally improved and even fitted to a specific player to adequately reflect their skill and the already resolved encounters.

In 2021, a research paper [10] described using Parallel Evolutionary Algorithms (PEA) to procedurally generate enemies for a game in the action-adventure genre, a genre closely tied to the rogue-like genre. In that piece of work and within the PEA, they used a difficulty function to approximate and estimate a group's difficulty and then put players against sets of differently difficult groups.

Building a difficulty function bears similar advantages and disadvantages as the difficulty matrix approach. It does not require additional processing power during the game, but it is also potentially very hard to design correctly and has to be redesigned or redone with every change to the mechanics of the game and adding of new content.

3. Problem definition and analysis

In this section, we shall define several terms related to the problem of PCG of enemies in role-playing rogue-like games

3.1 Definitions

3.1.1 Enjoyment and flow

The enjoyment of players in a game has been described using the idea of a flow, a persistent cycle of new experiences, rewards, and the sense of moving forward and having meaning. It was found that people with certain skills sometimes gain a positive experience from simply performing a difficult task requiring their certain skill, improving the said skill in the process. [11]

On the understanding of flow, researcher Mihalyi Csikszentmihalyi [12] formed eight elements of flow as an useful tool and indicators to measure flow in a task. The said elements are:

- A challenging but tractable task to be completed
- One is fully immersed in the task, no other concerns intrude
- One feels fully in control One has complete freedom to concentrate on the task
- The task has clear unambiguous goals
- One receives immediate feedback on actions
- One becomes less conscious of the passage of time
- Sense of identity lessens but is afterward reinforced

These elements have been iterated on and modified over the years and still serve as a solid background for measuring flow. These elements are the basis of our survey for the experimental part of this work.

3.1.2 Difficulty

The question of difficulty is an integral question of flow. To achieve desired flow and enjoyment, we present the player with a series of meaningful decisions and must provide feedback on those decisions.

Video game designer Jesper Juul[13] brings a cohesive and understandable description of what a game is, out of which we can derive a comprehensive understanding of difficulty: "A game is a rule-based formal system with a variable and quantifiable outcome, where different outcomes are assigned different values, the player exerts effort in order to influence the outcome, the player feels attached to the outcome, and the consequences of the activity are optional and negotiable."

With such a definition, difficulty in video games is a difficulty of efforts to bring the desired or the best outcome.

We can also divide difficulty into several categories depending on which type of skills are required in improving the efforts towards the desired outcome.

Blogger and video game designer Rhys Frampton defines a taxonomy of difficulty in his blog[14], dividing difficulty into three categories.

Comprehensive difficulty describes the amount of knowledge and ability to connect such knowledge to form logical conclusions and solve relevant knowledge-based puzzles. It tests not only the informational capacity but also the ability to form logical connections between pieces of information. Such difficulty is most often tested in puzzle and detective games.

Executive difficulty describes that a desired and best outcome requires physical activity that is either, fast, precise, or somewhat another way physically challenging. In the area of video games that may most often mean quickly and precisely moving the computer mouse our having quick reflexes and hand-eye coordination. Such difficulty is most often tested in first-person shooter video games, such as the Call of Duty series.

The last difficulty described is **strategic difficulty** and the related ability to quickly evaluate a large number of information and make fast tactical choices, usually requiring to react to new inputs and changes in both the rules or the situations the game presents. It also usually requires thinking several steps forward and picturing and preparing for different possible scenarios. This difficulty is most often found in strategy games such as StarCraft or the Total War series.

Besides the presented taxonomy, in story-driven games, a difficulty may be

present in a hard ethical or emotional choice in the game's story. For such a decision to bear meaning and contribute to the flow, each such decision should carry a resolution. In a dating simulator game, it may be either understanding how to be affluential in social situations, or it may be simply understanding the game's mechanics.

In rogue-like games, in particular, the common randomness aspect requires you to come up with new strategies on the spot and the player always needs to adapt to new circumstances. Even within a familiar environment with understandable rules, the best rogue-like games are designed to constantly generate new challenges. using the taxonomy described above, rogue-like games contain both comprehensive and strategic difficulty.

In a lot of rogue-like games, there is also an aspect of permanent death. Permanent death means no possibility for a player to save their progress and rewind if they happen to fail. Failing and dying in a rogue-like often result in a significant loss of progress or even having to completely start the game from start. Such a high-rich rich-reward system may seem to deter a lot of players from rogue-like games, but it is suggested that it may also bring a new meaning and more impact to players' actions and the impact of defeat. [15]

3.1.3 Originality

Whenever working with a procedural generation, one must implement a way to make sure the generated content does not get repetitive too fast and to provide new original content with enough variety so that the repetitiveness can not be easily recognized.

A key component in PCG is therefore an element of required randomness. Achieving true randomness has always been a struggle in the computer and mathematics world in general, but aside from cryptography and other particular uses, semi-random generators built-in most current computers are sufficient enough to seem random to human observers.

In PCG, the goal is usually not only to generate random numbers but usually to generate a random noise - two-dimensional pattern, out of which content and assets can be generated. As described in State of the Art in procedural noise functions: "It is a random and unstructured pattern, and is useful wherever there is a need for a source of extensive detail that is nevertheless lacking in evident structure." [16] Noise is most often used in generating maps and terrain.

3.1.4 Believability

At the moment, there is not a lot of hard evidence that players appreciate hand-crafted-looking content more than procedurally generated. However, we trust that it is safe to assume that content must seem as if crafted with intent and intended experience in mind instead of randomly thrown together without any underlying rules of consistency and realism.

For this reason, it can be expected to tighten the range of possibly generated outcomes with its own author-imposed rules to follow the desired design intentions. As an example, with randomly generated maps, the designer might impose restrictions for the maps to follow basic notions of geography.

3.2 Problem Analysis

The problem to be tackled brings several subproblems that desire to be addressed

3.2.1 Difficulty

One of the major subproblems is generating enemy encounters to be appropriately difficult. The required difficulty is hard to pinpoint exactly, s we lack the tools to measure and quantify ”difficulty”.

If we were trying to quantify a difficulty of a game, an enemy encounter, or basically any situation, one simple solution may simply be to count the number of outcomes that result in a victory. However, these solution quickly becomes inadequate, because simply the amount of possible positive outcomes hardly defines the actual difficulty player has in reaching them.

The previous approach could be modified and improved by being able to calculate the best decision at every step. That way we could define a measurement of difficulty as a number of mistakes one can make while still being able to reach the desired outcome. This seems like a bit more nuanced way, but it requires us to be able to compute every possible decision sequence in a game or specific encounter, and that can very easily not be possible with even simple games. There may simply be too many possibilities to calculate. This approach is also completely unusable in games that are not turn-based because the amount of real-time decisions increases very fast and it even becomes unclear what can be considered a decision in such a calculation.

The third approach I’d like to mention is not as accurate but may prove as

the best one for our use case. The best way to measure difficulty is to let players play the game and measure how often the player wins. We can also improve this method by then asking them how it felt.

3.2.2 Difficulty increments

In any used algorithm, we are going to have to increase and decrease the difficulty of an encounter in both big and small increments to reach our desired encounter difficulty of an enemy encounter.

A large increment or decrement of difficulty could be adding or removing an entire creature from the enemy encounter. This may run into constraints of a game, especially with an upper bound of possible enemies present, and with a large number of enemies, the expected difficulty increases may get diminishing results.

A medium increment or decrement of difficulty is by adding or removing a part of a creature's equipment or skill. In a fantasy setting RPG game, this could mean adding a layer of armor, a new weapon, or a new active or passive ability an enemy could use. This seems promising but may result in a generated creature with either too few or too many such pieces of equipment or abilities, which may turn out hard to keep track of the player and may look out of place.

The slightest increments or decrements of difficulties in an encounter can be changes in values of already present equipment or abilities, tuning it by a small amount to reach even tiny desirable changes. The easiest implementation of this is changing the amount of health an enemy can lose before being defeated. However, such straightforward changes may result in a loss of enjoyment and flow, due to encounters simply taking way too short or too long.

In contrast, the AGE algorithm we use in this thesis doesn't pre-compute any encounters but relies on fast real-time simulations of combat encounters to calculate the encounter's difficulty. This saves space because the pre-computed difficulty matrix could be getting very large with more complex encounters, and this approach allows for balance changes and additions to the game without requiring computing a whole new difficulty matrix.

Matouš Kozma's approach, mentioned in the previous chapter, does bring several advantages to the PCG of enemies. Firstly, it doesn't require a lot of processing power. The AGE algorithm requires several sets of simulations, possibly spanning even into hundreds and that simply may be too much for some computers and may cause visible lag during the sections the simulation takes place

in the background. His approach also allows for incremental updates of the difficult matrix over time, allowing further balances reflecting the player's individual playstyle and strengths. However, we see a great hindrance in having to renew and recalculate the difficulty matrix with any new balance changes to the game, which is why we have decided to go with our own approach.

4. The Game

4.1 Existing vs. custom game

A major question for the thesis was whether to tailor the algorithm to a specific existing game or whether to create our own to test and experiment on it. Considering tailoring it to an existing game would very likely require advanced programming skills and would very likely bring licensing issues, We have decided to create a game from scratch, allowing for a much larger variety of options and design paths and giving both creative and designer freedom. The created custom game is however very clearly inspired and derives from existing known games and the designed algorithm is easily imagined and implemented into similar-style existing games.

4.1.1 Existing game

The game to test the AGE algorithm was from the start planned to be a turn-based rogue-like game. A major inspiration for coming up with this problem was the game Darkest Dungeon.

Darkest Dungeon is a turn-based role-playing rogue-like game in which you control a party of adventurers, which you repeatedly send out in groups of four to missions into catacombs and similar locations to hunt down enemies, gather supplies, and fulfill similar tasks. The chosen heroes all wield different skills and equipment, and a major part of gameplay is deciding on the correct combinations of skills and equipment to use to maximize the chance to win while being aware of possibly losing it in case of a failed mission together with the party. Besides this gameplay aspect, the player makes use of the gathered supplies, money, and experience of their heroes to improve their hamlet, which serves as a camp of operations in between missions.

The missions themselves consist of traversing a dungeon full of rooms interconnected with pathways and tunnels. The Player's party is always either battling an encountered group of enemies, or exploring the dungeon, gathering loot, solving puzzles, and finding new equipment and consumable items.

The battles are turn-based, with different characters having different speeds and therefore acting in an order according to it. The battle also brings focus to positioning, because both the battling parties stand in a line one behind another and different skills have different reaches. For example, a stab with a sword may

only be able to reach the first two characters in an opposing group, so the game requires strategic decisions around positioning as well.

4.1.2 Custom game

Creating a custom game allowed us to focus on the key aspects of the gameplay that we find most important to our topic, which is repeated combat encounters in a dungeon with time in between to rest, heal, and improve the player's characters to face subsequently tougher enemies.

In both existing and our custom game, the battles are four vs. four with positioning having to be taken into account. In both games, the fight is going to be turn-based with a system to determine when each character acts.

In the existing game, the player is rewarded with consumable items, new equipment, and treasure for each fight, while in our custom game, we are going to simplify this system to better suit our goal a to be more straightforward.

In the existing game, the time in between battles is spent exploring the dungeon, finding loot and solving puzzles, and occasionally having the opportunity to heal and improve the player's characters' abilities. This we are going to replace with a time for improvements and healing in between each battle and we are going to make use of this downtime to generate a new enemy encounter.

4.2 Game overview

For the purpose of this thesis, we have created a lightweight turn-based role-playing rogue-like game for testing the AGE algorithm. The custom game is called Deep Crawl. The game is set within a giant humanoid body with the main protagonists implied being microorganisms fighting against different microorganisms, defending the body. The theme and setting of the game are very simplistic to allow us to focus on the battles while also providing at least some significance and meaning.

4.2.1 Game goal and structure

In Deep Crawl, the player controls a party of four characters, subsequently facing up to four enemy encounters during the game. After each battle, the player visits a room with a campfire, giving their party time to rest and prepare for the next combat. While in this phase, players may heal or revive their characters

and may purchase upgrades for them, improving their gear and abilities before the following combat.

4.2.2 Gameplay

The main gameplay of Deep Crawl lies in carefully selecting and targeting abilities to successfully eliminate the enemy group in combat before they eliminate the player's group. In battles, the player decides whom to target, what abilities to use, and even whether heal their own creatures or deal more damage instead.

The second aspect of the gameplay is in the campfire room, in which player needs to strategize around allocating resources to heal and upgrade their characters. Each creature has a large array of possible upgrades that allow for tactical choices to create a coherent and strong group for the following combats.

4.3 Game-specific definitions

From now on, all definitions from now on will refer to the entities and concepts in Deep Crawl. Whenever we would want to speak more broadly about those words, we will make it clear

- *Creature.* An entity consisting of a name and components such as speed, health, armor, etc...
- *Creature group.* Group of exactly four creatures.
- *Enemy.* A creature with the purpose of either being killed by or for defeating the player's creature group
- *Enemy group.* A creature group of enemies
- *Defeat.* A group is defeated in this game when all of its creatures' are killed
- *Kill.* A Creature is killed when its health count reaches zero.
- *Component.* Part of a creature describing any of their attributes besides name - health, armor, speed, and any others. All creatures have several components.
- *Health.* A component that describes the amount of damage it takes for the creature to be killed.
- *Campfire.* A game phase, in which the player chooses upgrades for their creatures, may heal the wounded ones and revive those killed.

- *Upgrade.* An action that permanently improves a component within a creature
- *Downgrade.* An action that permanently worsens a component within a creature
- *Combat encounter.* A situation consisting of two creature groups trying to defeat each other over the course of several rounds.
- *Ability.* An action usable during a creature's turn. During its turn, a creature can only play one ability. Each creature has exactly two abilities - a basic attack and a special ability.
- *Basic attack.* An action possible by having a PhysicalWeapon component, which all creatures in Deep Crawl contain. It Deals physical damage but is modifiable by additional components.
- *Special ability.* A second action that each Creature in Deep Crawl possesses, usually provides a special attack, a spell, or healing. It is generally a more modifiable and specific ability.
- *Creature's turn.* When the creature is given the possibility to use one of its abilities.
- *Round.* A period of time in which every creature on the battlefield gets to have a turn. The order of creatures is dictated by their speed component. Creatures that are dead or stunned don't take their turn.
- *Speed.* A component deciding the order in which the creatures take their turns. The higher the value in the speed component, the earlier a creature takes its turn. With high enough speed, a creature may play twice in a round.

4.4 Components

Creatures in Deep Crawl are comprised of a name and a number of component objects carrying information about the creature, its attributes, or its behavior.

4.4.1 Component types

- *Health.* Keeps track of creatures' health and changing it when receiving damage or healing.

- *Speed.* Determines the order in which creatures get their turn.
- *PhysicalWeapon.* Describes the basic attack ability, which deals physical damage, and which all creatures possess.
- *LongWeapon.* Provides the basic attack with a range to reach all enemies.
- *ShortWeapon.* Provides the basic attack with a range to reach only the first row of opponents
- *Armor.* Reduces all incoming physical damage by a fixed amount.
- *PowerStrike.* Adds physical damage to a special attack. Restricts the reach of special attack to short range.
- *PoisonBlast.* Provides the special attack with the ability to inflict poison that lasts several rounds.
- *Poison.* Deals elemental damage to its wielder at the end of a round.
- *AmplifyPoison.* Enhances basic attack to amplify the potency and duration of poison on a targeted creature.
- *ShieldBash.* Adds physical damage, short range and the ability to inflict stun.
- *Stun.* Makes the creature unable to perform actions.
- *Claws.* Provides the special attack to deal physical damage at any range and deals two times more damage to enemies bearing Clawed component. Inflicts Clawed component to enemy hit.
- *Clawed.* Makes the holder receive 2x damage from Claws special attacks.
- *FirstStrike.* Makes the bearer do more physical damage when they are at full health.
- *Anger.* Makes the bearer do more damage when below 50% health.
- *FieryWeapons.* Adds fire damage to all physical attacks.
- *HealingWave.* Makes the special attack able to target allies and heals the target.
- *ElementalResistance.* Reduces poison or fire damage dealt to the wielder.

4.4.2 Timed Components

Some of the components are timed components, which means that they persist only for a limited number of rounds before destroying themselves. At the end of each turn, all components receive a query of a Tick type and reduce their timer by one, some even performing an action and sending a query by themselves. The timed components include:

- *Stun.* A creature in possession of the Stun component cannot act.
- *Clawed.* Attacks with Claws deal double Claws damage.
- *Poison.* Deals periodic elemental damage.

4.4.3 Upgradable Components

Some of the components are upgradable, having the possibility to increase or decrease their effectiveness. All Components have their own means of upgrading, their own costs for upgrades, and their own upgrade limits.

Some upgrades have an upper limit for upgrades, being unable to get upgraded beyond a certain point, while others are able to be upgraded indefinitely. This has been done for game balance reasons.

Almost all upgrades have a lower bound for downgrading. If you try to downgrade beyond this point, the component gets removed instead. An exception is *Health*, *Speed* and *PhysicalWeapon*, which are required for combat and therefore cannot be removed.

- *Health.* Every upgrade adds or subtracts 10 health points and costs 2 points.
- *Speed.* Every upgrade adds or subtracts 1 speed and costs 2 points.
- *PhysicalWeapon.* Every upgrade adds or subtracts 5 physical damage and costs 2 points.
- *Armor.* Every upgrade adds or subtracts 3 points of physical damage negation. Costs 2 points. Gets removed when below 3 armor.
- *PowerStrike.* Every upgrade adds or subtracts 10 physical damage and costs 2 points. Gets removed when below 20 damage.
- *PoisonBlast.* Has three levels of upgrades. The second level increases damage. The third level increases duration to infinite. Costs 2 points. Gets removed if downgraded below the first level.

- *ShieldBash*. Has three levels of upgrades. The second level increases damage. The third level increases both duration and damage. Costs 2 points. Gets removed if downgraded below the first level.
- *Claws*. Every upgrade adds or subtracts 5 physical damage and costs 2 points. Gets removed when below 10 damage.
- *FirstStrike*. Every upgrade increases or decreases the damage modifier by 20%. Costs 1 point. Gets removed when below 20%.
- *Anger*. Every upgrade increases or decreases the damage modifier by 10%. Costs 1 point. Gets removed when below 20%.
- *FieryWeapons*. Every upgrade adds or subtracts 5 fire damage and costs 2 points. Gets removed when below 5 damage.
- *HealingWave*. Every upgrade adds or subtracts 10 health restoration power and costs 2 points. Gets removed when below 20 power.
- *ElementalResistance*. Every upgrade increases or decreases the resistance by 20%. Costs 1 point. Gets removed when below 20%.

5. AGE approach

Our approach for generating adequately difficult enemies is based on preparing a base enemy group and then incrementally tuning its difficulty while running multiple sets (further called series) of simulated encounters, each time changing the setup until reaching the desired percentage of won battles (further called win rate).

In further text, *simulated encounter* is an encounter in between the player's group and the enemy group, in which both sides are controlled by a simple decision-making algorithm, trying to emulate the player's behavior.

5.1 Simulations

In between each modification of the enemy group, a series of simulated encounters is run and the results are evaluated.

5.1.1 Evaluation

When evaluating a result of a simulated encounter, we have decided to look only and the fact of which side has won or lost. A more detailed approach could be evaluating how much health the winning side has lost and/or how many creatures have died on their side. We have decided to not implement such an approach.

Depending on the game the approach may be used for, a more complex evaluation of the results of combat can be used, taking into consideration abilities used, loot gained, and even taking into consideration not only the result but also the progress of the encounter.

5.1.2 Simulation decision making

For the purpose of simulation, there must be a way to adequately mimic and simulate the player's decision-making process and pick abilities accordingly. In theory, there is a vast amount of approaches in the field of artificial intelligence as to how to approach such a decision-making system.

Our custom game provides full transparency regarding the outcome of each character's actions and even does not have any chance-based abilities, so it would be possible to calculate an optimal strategy for each encounter. However, such

an approach would completely negate the purpose of the series of simulations, so we do not engage in this approach.

In our custom game and for the experimental part, we employ a simple decision-making system. Half the time, it picks the best course of action for a character depending on simple metrics - how much damage it deals to an opponent or heals an ally, considering all the possible combinations of abilities to use and opponents to target. The remaining half the time, it picks a random ability for a random possible target.

In future work, the decision-making system in simulations could be more personally tailored to fit the player's playstyle to reflect their likely actions and decision better, to generate even more precise adequately difficult enemies.

5.1.3 Enemy generation process

In the following text, the word *modification* means either upgrading, downgrading or adding a component.

A base group of enemies is generated and a series of simulated combats is performed. At the end of this series, the win rate is calculated and compared to the desired target win rate, in the experiment being 40% - 60%. If the target win rate was not hit, the group is modified. The amount of modification is determined by a calculated amount of modification points $U = Abs(targetWinrate - achievedWinrate)/3$. If the evaluated win rate was below the target, the group is going to be upgraded, and vice versa with downgrading.

After computing the number of modification points, a creature is randomly chosen from the group and a random modification is chosen from all the possible ones to apply to the creature and its components. When a component is chosen to be downgraded beyond its minimal point, it is removed instead. When a component is chosen to be upgraded beyond the maximal possible upgrade, the modification fails and a new possible modification is chosen.

Each modification has a previously assigned cost of modification points and the process is repeated until modifications are no longer possible. The addition of the system of modification points is implemented to speed up the generation process and lower the number of required series of simulations.

Each generated creature is restricted to only being able to contain six upgradable components. This is to prevent information overload for the players having to figure out the enemy group's strengths and weaknesses. It is also

not possible to remove *health* and *speed* components because of their necessity to make the *creature* function and take turns.

5.1.4 Simulation effieceny

When running simulations of combat, the efficiency comes into question, because all the simulations have to be run during the player's gameplay, taking up computational power. For this reason, the series of simulations are run during the campfire phase, when players are choosing upgrades for their party.

However, generating the enemy group during the campfire phase while the player is choosing upgrades means that the enemies are suited for the previous version of the player's group. This could be avoided by waiting for the player to pick their upgrades and then making the player wait for the series of simulations to finish before starting the following level.

This slight flaw has been deemed acceptable as it doesn't invalidate the generation results and may even be interpreted and accepted as always giving the player a slight upper hand and a feeling of outsmarting the game, making the player always a few steps ahead.

6. Experiment parameters

In this chapter, we describe the methods and parameters used for testing the algorithm and approach presented in the previous chapters. First, we describe the goals of the experimental part and we describe the specific parameter of the testing versions of the custom game.

6.1 Goals

The testing should fulfill or deny the expectations of the presented algorithm and component-based approach in PCG. Specifically, we want to focus on the following targets

- The generated enemies should be beatable
- The generated enemies should be adequately difficult throughout the game
- The generated enemies should present new challenges
- The generated enemies should not overload the player with information
- The player should enjoy the game
- The player should not be able to recognize procedurally generated enemies from human-designed ones.

6.2 Control group

We have created an algorithm for generating enemies and want to compare it with an experience with human-designed counterparts. For the purpose of the experiment, we have prepared two slightly altered versions of the game, one with procedurally generated enemies, and a second with encounters hand-designed by the author, both containing four enemy encounters to progress through.

6.3 Experiment

In both variants of the game, the first encounter was human-designed by the author, a this is for the purpose of introducing the player to the game, showing the mechanics and controls, and making the player ready for the harder encounters.

This decision was not necessary for the experiment but allowed us to gather more data as opposed to presenting the player with three encounters and a written tutorial. Such an approach would very likely take a similar time, provide less data, and may even deter some testers from finishing the experiment due to having to read through a lot of text.

After finishing each encounter, the player was awarded 10 upgrade points and was moved to the campfire phase, being given an opportunity to upgrade their creatures.

In the meantime, a series of simulations were run, each series consisting of 30 simulated battles, running and modifying the party every 30 simulated battles until the desired win rate of 40-60% was hit. After the simulations were finished, the next encounter was possible to be started.

6.3.1 Data gathering

All the testers have been asked to fill out a survey, which was also used to assign them to real or control groups before presenting them with a link to their respective variant of the game.

The survey has been focused on their flow, their perception of difficulty, and their gaming experience and habits. In the survey, we used the EGameFlow questionnaire to self-report flow in video games [17]. The whole survey can be found in the Attachments.

7. Experiment Results

In this chapter, we will present the results gathered from the experiment and testing, the achieved or not achieved goals, and a general overview of the efficiency of our algorithm.

7.1 Result analysis

The entire results can be seen attached, while here we present the differences between the test and control groups in sections of the survey. In the experimental part, 23 people took part in the experiment and filled out the form to the end. Unfortunately, 2 of these have been sent an older version of the survey, missing two questions. Their results have been counted in.

Concentration

Regarding the concentration questions, there are not a lot of important differences. There are slightly higher scores in the test group regarding general focus on the game. Other than that, an important result for our thesis is a similar score regarding the question ”The game tasks are adequate for me” because it gets close to the topic of difficulty.

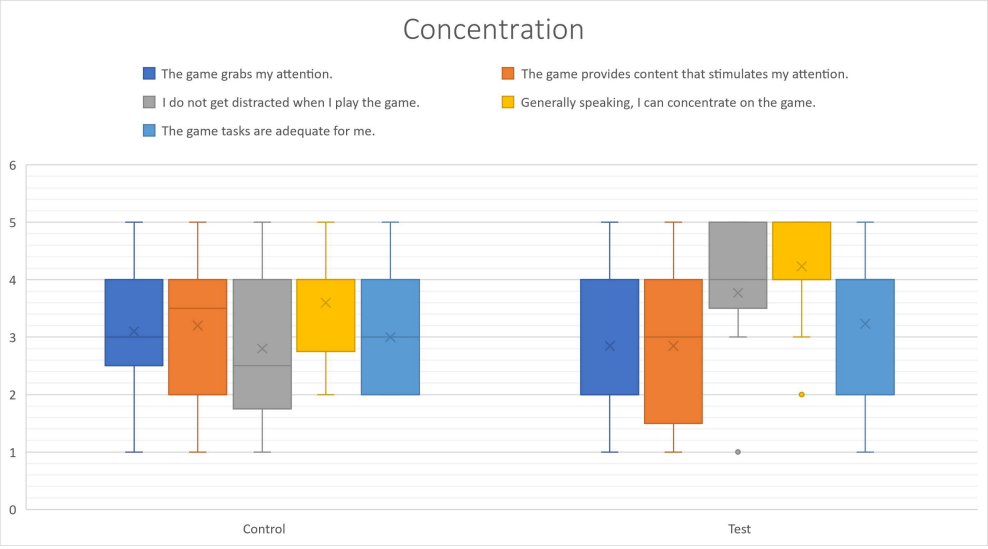


Figure 7.1: Results of the survey subsection: Concentration

Goal clarity

Regarding goal clarity, there are very few differences in answers, only slightly less predictability of the game for the test group.

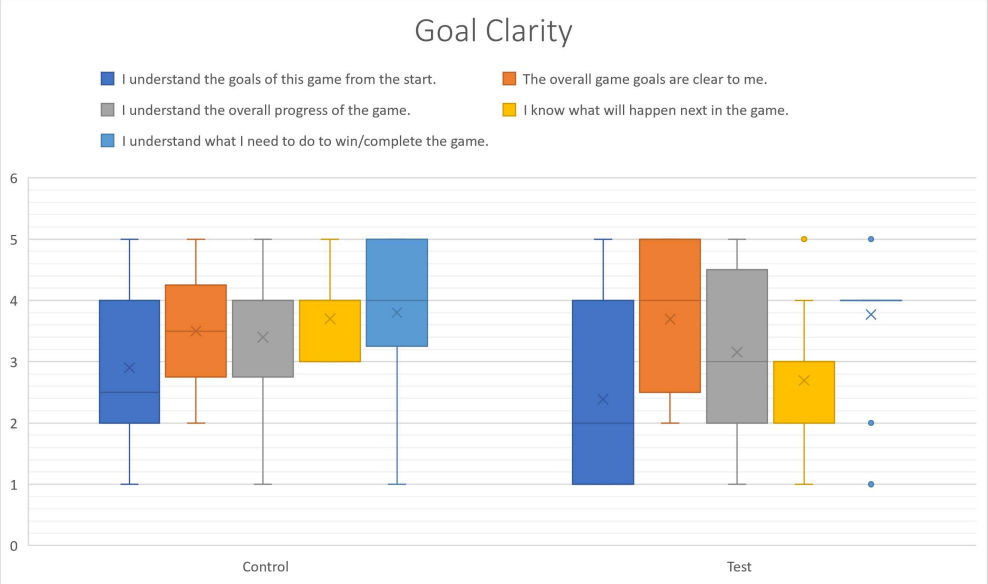


Figure 7.2: Results of the survey subsection: Goal clarity

Feedback

There are no significant differences in feedback, with only a slightly larger spread of answers in the test group.

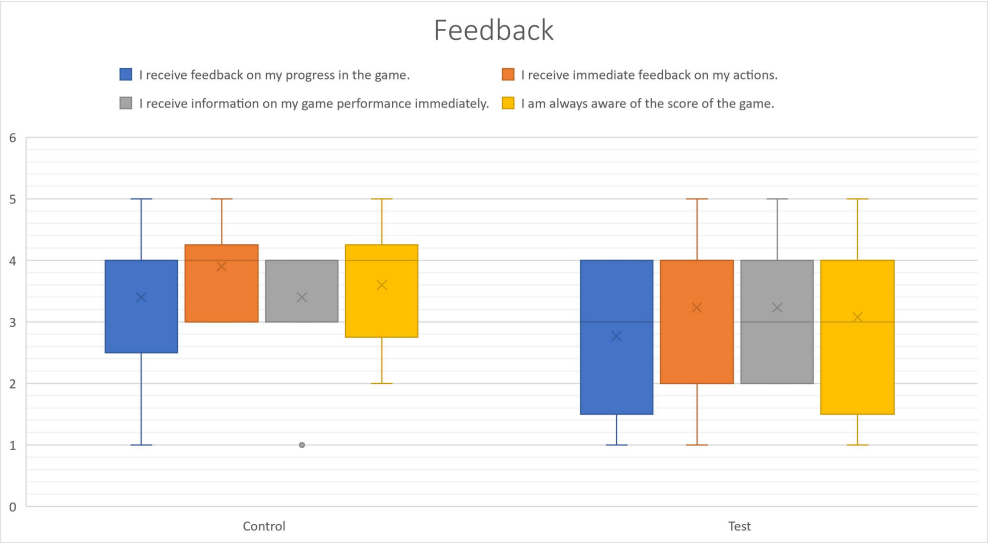


Figure 7.3: Results of the survey subsection: Feedback

Challenge

This part of the survey shows similar scores regarding the challenge the game provided, which is a key topic for our research and goal. There are noticeably high scores in regarding the improvement of players' skills and their motivation for it, but as expected, there is a low score for tailoring the experience differently for different players.

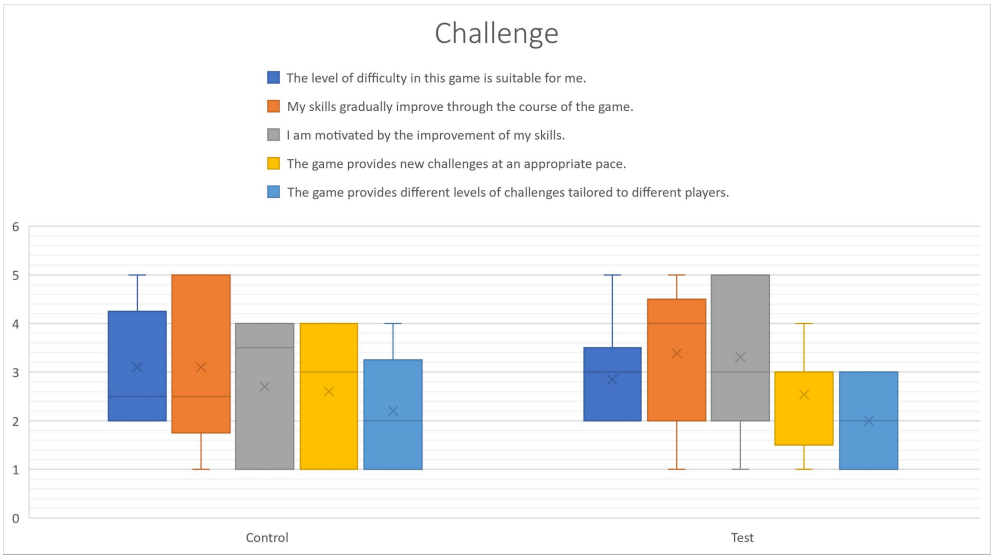


Figure 7.4: Results of the survey subsection: Challenge

Autonomy

There are very similar scores regarding the play's autonomy, with the test group rating on average only slightly higher.

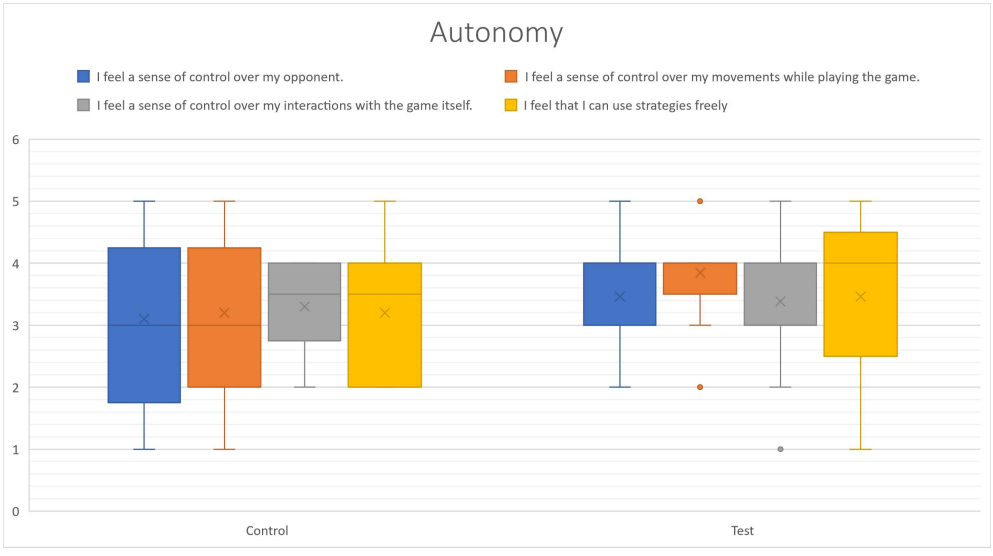


Figure 7.5: Results of the survey subsection: Autonomy

Immersion

There are some non-trivial differences in scores regarding immersion, but considering this was not at all the focus of this thesis, we don't derive a great deal from these results.

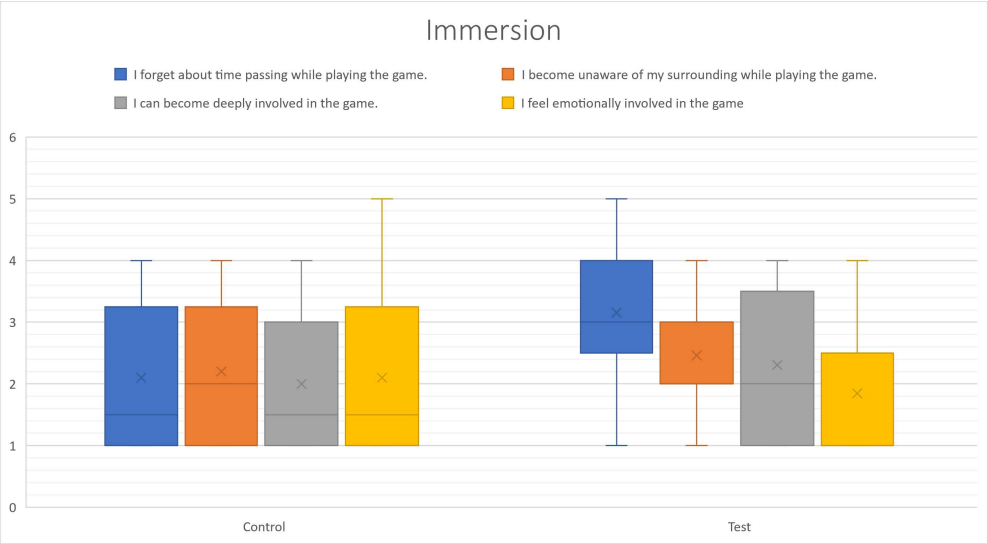


Figure 7.6: Results of the survey subsection: Immersion

Enemies

A key part of the survey and our thesis was to measure whether the generated enemies are easy to understand and if the generated enemies are interesting to the player and not overwhelming. In general, the test group scored lower in most of these indicators.

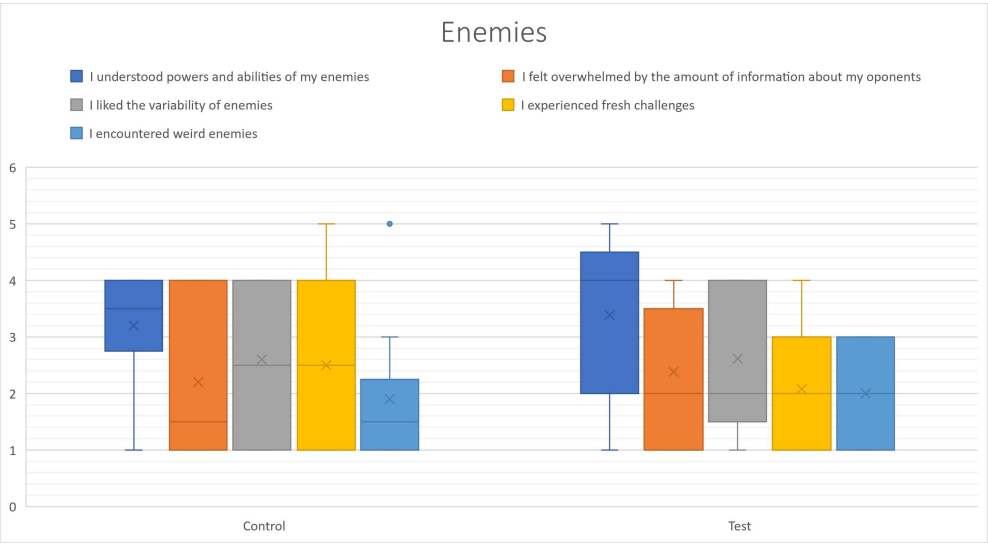


Figure 7.7: Results of the survey subsection: Enemies

Enemy design

We wanted the player's opinion on how did the enemy groups feel in terms of design, and whether they seemed designed by a human or by a computer. The X-axis shows the number of answers each option received.

Comparing the test group to the control group, players in the control group thought that more enemies were generated than those in the test group. In the test group, more players also considered the groups to be well-designed.

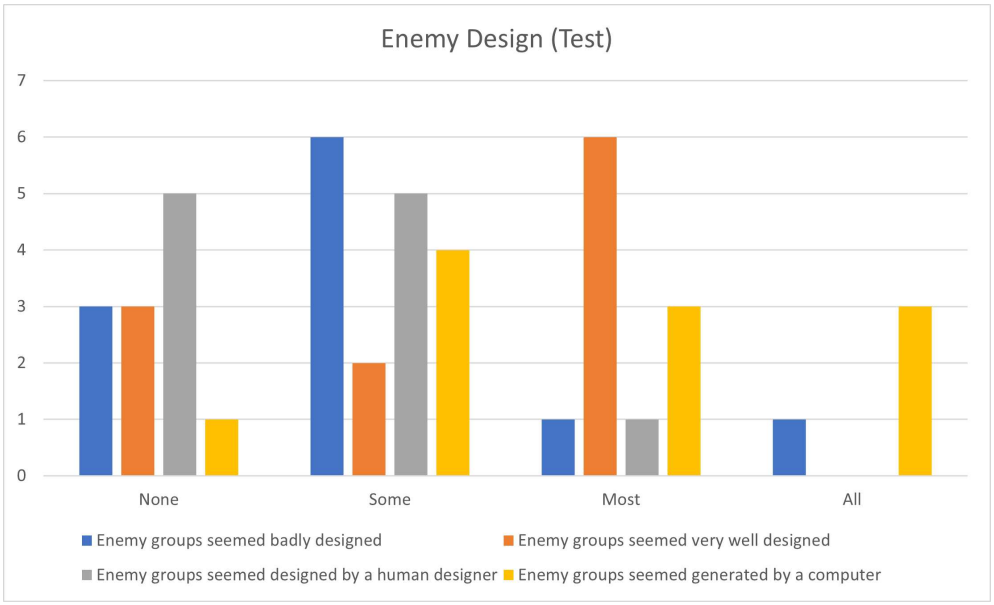


Figure 7.8: Results of the survey subsection: Enemy design (test)

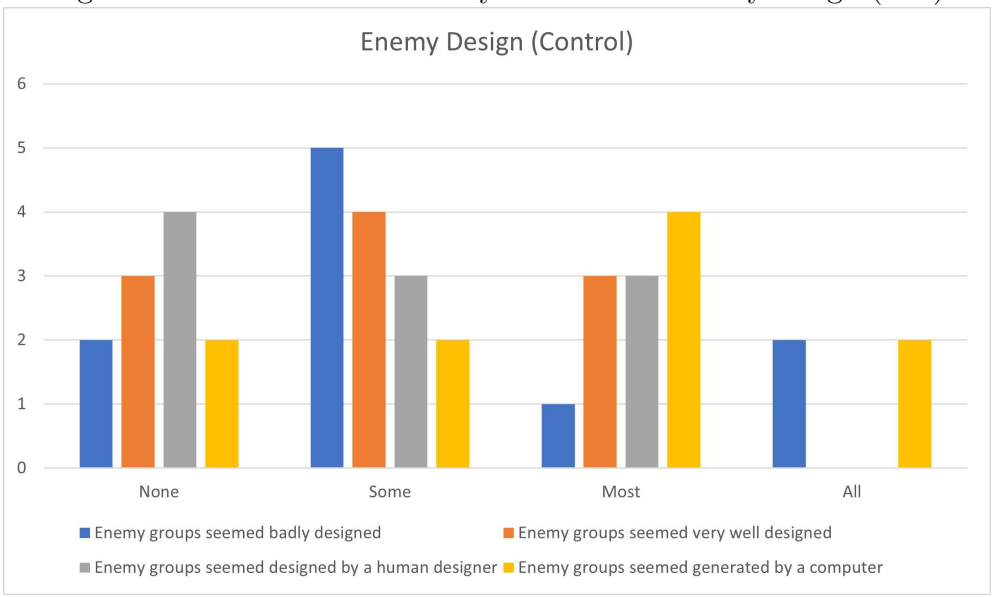


Figure 7.9: Results of the survey subsection: Enemy design (control)

Generated levels recognition

When tasked with assigning an individual level to be either human-designed or computer-generated, both the control and test groups scored very similarly, assigning most groups to be computer generated, with the exception of the first one.

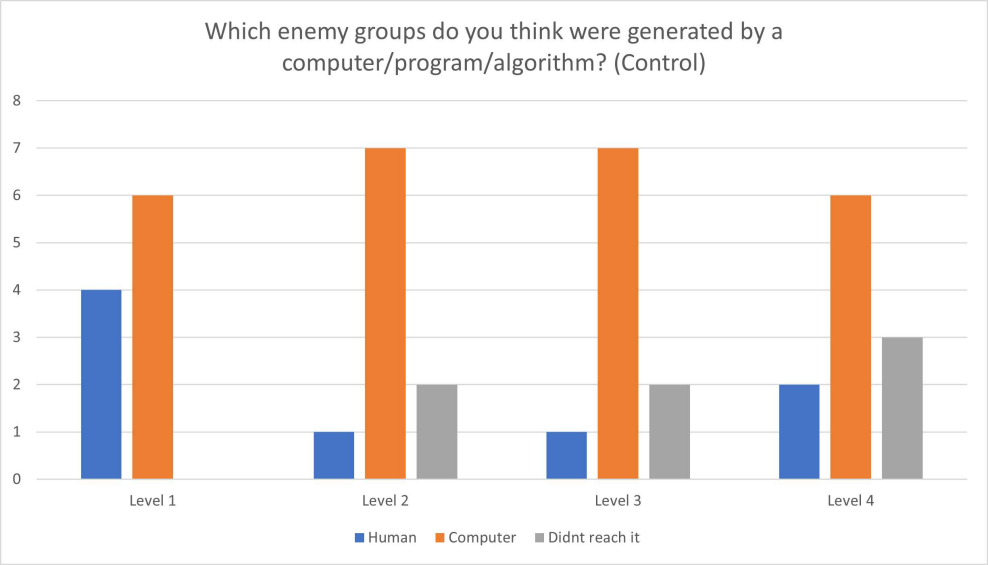


Figure 7.10: Results of the survey subsection: Generated levels recognition (control)

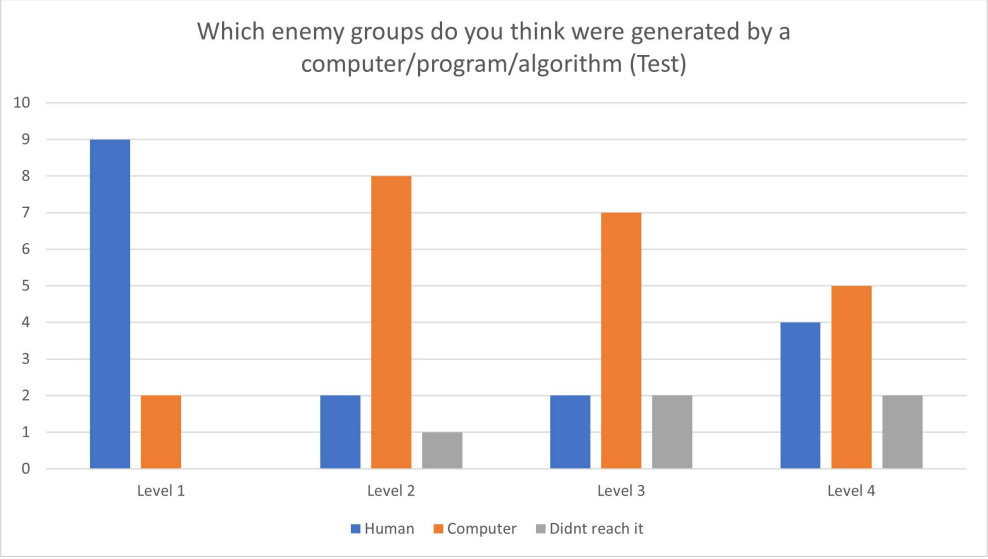


Figure 7.11: Results of the survey subsection: Generated levels recognition (test)

8. Discussion

In this chapter, we will discuss the results of the experiment and the possible changes and improvements in either the algorithm itself or the methodology of the experiment.

8.1 Testing game evaluation

We are quite satisfied with the game designed for the purpose of this experiment, as it fulfills all the criteria of a rogue-like RPG that we described, while being nor too simplistic or not overly complicated. The game implements a robust component system and is easily scalable implementing new components, mechanics, and levels. In the game, we have failed to implement abilities hitting multiple enemies and we have failed to properly make use of positioning and switching positions on the battlefield.

8.1.1 Goals

The generated enemies should be beatable

84.6% of players report finishing all four levels, and only 46.2% report having lost at some point and having to repeat an encounter. Only one participant reports having lost ”many times”.

We are quite satisfied with such results, making the game playable and beatable, but not too easy at the same time.

The generated enemies should be adequately difficult throughout the game

76.9% of players rate the statement ”The level of difficulty in this game is suitable for me.” with a score of 3 or higher. Three participants even rated this statement with a 5, meaning ”definitely agree”. 69.2 % of players rate the statement ”The game tasks are adequate for me.” with a score of 3 or higher.

We are satisfied with the result of this goal, as having adequately difficult enemy encounters was the main goal of this thesis.

The generated enemies should present new challenges

61.5% of players rate the statement "The game provides new challenges at an appropriate pace." with a score of 3 or higher. However, only 38.4 % of the players rate the statement "The game provides different levels of challenges tailored to different players." with a score of 3 or higher. However, only 46.1 % rated "I liked the variability of enemies" such scores.

This goal has not been met very well and we consider it unsuccessful.

The generated enemies should not overload the player with information

69.2% of players rate the statement "I understood powers and abilities of my enemies. " with a score of 3 or higher. On top of that, only 46.1% of the players rate the statement "I felt overwhelmed by the amount of information about my opponents." with a score of 3 or higher, signifying some amount of information overload.

We are satisfied with this goal, even though we wished for better results, considering the information overload has been one of the repeated fears with the AGE algorithm. However, the results are still quite satisfactory.

The player should enjoy the game

76.9% of players rate the statement "I forget about time passing while playing the game." with a score of 3 or higher. 69.2 % of players rate the statement "The game tasks are adequate for me." with a score of 3 or higher.

We are reasonably satisfied with the high scores in the questionnaire regarding the player's enjoyment altogether. Considering the limited amount of time and effort put into the game, such a result is satisfactory to the author and motivates them to further pursue game development.

However, a recurring problem was a lack of clear instruction and a generally bad user experience. The tutorial level has proven to be satisfactory in teaching how to control the game but still did not provide enough clarity of instructions. Unfortunately, there simply was not time for proper game testing before the experimental part of the thesis has been launched.

The player should not be able to recognize procedurally generated enemies from human-designed ones.

Unfortunately, we have not been able to achieve great results in the area of players being or not being to able discern between human-designed and procedurally generated enemies, mostly due to the fact that most people thought even the human-designed encounters to be procedurally generated. This only speaks to the author's game design ability, but not a lot more.

The goal was technically met, but not the way we would hope for, not pointing at the quality of the algorithm but rather the opposite with the human design. Looking back, the question in the experiment might have been put too suggestively, prompting respondents to guess the computer generation more.

8.1.2 Component system

The component seems to be a promising robust catch-all system for implementing most mechanics in the game. The component system may be scaled to not only influence the combat, but also animations, graphics, or storytelling.

The component could be made used in rendering graphics for a character, with the component adding visual effects or adding new sprites. For example, in the current version of the game, there is a component called "LongWeapon" and such a component could likely easily be programmed to render the creature sprite with a spear, adding multi-functionality to the system.

In the current version of the game, there are story screens before some of the encounters, showing the dialogue of the player characters. Such screens could also be generated in relation to the component presented in some of the player characters dynamically reacting to the upgrades picked by the player or by other events happening during or out of encounters.

8.2 Procedural generation evaluation

Overall we are satisfied with the results of the PCG. Even though the initial plays were much more ambitious, we even hoped to make the algorithm gather information about the player's playstyle and take it into consideration in the enemy generation. However, such goals have proven to be worthy of a larger piece of academic work.

The simulation-based PCG has proven to be viable, especially in a turn-based game, where combats can be simulated efficiently and fast. Implementing

a similar approach in any real-time game would likely be a lot more processing power intensive and therefore much harder to achieve.

The difficulty balancing part has proven to be successful, with testers rating their experience to be moderately difficult throughout the game with no spikes or falls in difficulty.

8.2.1 Experiment

The experimental phase has unfortunately brought fewer respondents than expected, plus two of the have been mistakenly sent an older version of the form lacking two questions. Besides that, some respondents have claimed the game crashed during some of the simulations, and on top of that, there has been a bug found causing the player's group to completely heal before starting a new level. As far as we can tell, the bug has not been noticed by any of the respondents and does not invalidate the results, but still is unfortunate and a pity.

Conclusion

In this thesis, we developed and tested an approach to procedurally generate enemy groups for turn-based RPG rogue-like games. The developed approach used simulations running in the background to create interesting adequately difficult enemies. We have implemented the approach in a custom game bearing similarities to games in the genre to run experiments with it and to measure its success. The results of the testing phase have shown that the enemy groups generated by our algorithm generally seem enjoyable and moderately difficult, and the experience facing them has been overall rated as slightly positive. As a secondary goal, we have tried to test whether players would be able to properly tell a human-designed enemy group from the procedurally generated ones, and in that sense, the experiments don't bring a lot of results since almost all of the encounters were thought to be generated. The written algorithm is designed to work within the custom game created for the purpose of this thesis, but the tested approach can be expanded upon or implemented in games that would desire to either partially tweak or completely generate enemies using simulations.

Bibliography

[1] Mark J. Nelson Noor Shaker, Julian Togelius. *Procedural content generation in games*. Springer, 2016.

[2] Steven L. Kent. *The Ultimate History of Video Games*. Random House International, 2002.

[3] Entertainment software association. *2020 Essential Facts About the Video Game Industry*. Entertainment software association, 2020.

[4] Pratama Atmaja, Sugiarto, and Eka Mandyartha. Difficulty curve-based procedural generation of scrolling shooter enemy formations. *Journal of Physics: Conference Series*, 1569:022049, 07 2020.

[5] Ken Arnold Michael Toy, Glenn Wichman. Rogue. `https://web.archive.org/web/20080715035939/http://roguelikes.sauceforge.net/pub/rogue/index.html`. Accessed: 2022-12-10.

[6] Mojang Studios. Minecraft. `https://www.minecraft.net/en-us/about-minecraft`. Accessed: 2022-12-10.

[7] Hello Games. No Man's Sky. `https://www.nomanssky.com`. Accessed: 2022-12-10.

[8] Electronic Arts. Spore. `http://www.spore.com/`. Accessed: 2022-12-10.

[9] Matouš Kozma. Procedural generation of combat encounters in role playing video games, 2020.

[10] Leonardo T. Pereira, Breno M. F. Viana, and Claudio F. M. Toledo. Procedural enemy generation through parallel evolutionary algorithm. In *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 126–135, 2021.

[11] Benjamin Cowley, Darryl Charles, Michaela Black, and Ray Hickey. Toward an understanding of flow in video games. *Comput. Entertain.*, 6:1–27, 07 2008.

[12] M. Csikszentmihalyi. *Flow: the Psychology of Optimal Experience by Mihaly Csikszentmihalyi*. CreateSpace Independent Publishing Platform, 2018.

[13] Jesper Juul. *The Game, the Player, the World: Looking for a Heart of Gameness*. 01 2003.

[14] Rhys Frampton. A new taxonomy of difficulty. https://www.rhysframptongames.com/rhysframptongames-blog/a-new-taxonomy-of-difficulty. Accessed: 2023-1-2.

[15] Amra Taggart. *On the Nature of Permadeath Experiences in Video Games*. PhD thesis, Deakin University, 2022.

[16] Ares Lagae, Sylvain Lefebvre, Robert L Cook, Tony Derose, George Drettakis, David S Ebert, John P Lewis, Ken Perlin, and Matthias Zwicker. State of the art in procedural noise functions. *Eurographics (State of the Art Reports)*, pages 1–19, 2010.

[17] Shu-Hui Chen, Wann-Yih Wu, and Jason Dennison. Validation of egameflow: A self-report scale for measuring user experience in video game play. *Computers in Entertainment*, 16:1–15, 09 2018.

List of Figures

A. Attachments

A.1 Attachment 1 - Survey

The survey was presented as an electronic form to fill out, assigning the respondent randomly to a control or test group in the process. The player was first instructed to play the game running on a provided web page and then come back to the form to fill it in.

A.1.1 The form

Have you finished all four levels?

(single choice)

- Yes
- No

Have you at any point lost a level and had to restart?

(single choice)

- Yes, many times
- Yes, a few times
- No

In the following section, the respondent is tasked to rate on a scale of 1-5 how much they agree with a statement regarding their experience. (1 = definitely agree, 2 = agree, 3 = neutral/unsure, 4= disagree, 5 = definitely disagree)

Concentration

- The game grabs my attention.
- The game provides content that stimulates my attention.
- I do not get distracted when I play the game.
- Generally speaking, I can concentrate on the game.
- The game tasks are adequate for me.

Goal Clarity

- I understand the goals of this game from the start.
- The overall game goals are clear to me.
- I understand the overall progress of the game.
- I know what will happen next in the game.
- I understand what I need to do to win/complete the game.

Feedback

- I receive feedback on my progress in the game.
- I receive immediate feedback on my actions.
- I receive information on my game performance immediately.
- I am always aware of the score of the game.

Challenge

- The level of difficulty in this game is suitable for me.
- My skills gradually improve through the course of the game.
- I am motivated by the improvement of my skills.
- The game provides new challenges at an appropriate pace.
- The game provides different levels of challenges tailored to different players.

Autonomy

- I feel a sense of control over my opponent.
- I feel a sense of control over my movements while playing the game.
- I feel a sense of control over my interactions with the game itself.
- I feel that I can use strategies freely.

Immersion

- I forget about time passing while playing the game.

- I become unaware of my surrounding while playing the game.
- I can become deeply involved in the game.
- I feel emotionally involved in the game.

Enemies

- I understood powers and abilities of my enemies.
- I felt overwhelmed by the amount of information about my opponents.
- I liked the variability of enemies.
- I experienced fresh challenges.
- I encountered weird enemies.

Enemy groups

- Enemy groups seemed badly designed.
- Enemy groups seemed very well designed.
- Enemy groups seemed designed by a human designer.
- Enemy groups seemed generated by a computer.

Which of the which enemy groups do you think were generated by a computer?

(multiple choice)

- level 1
- level 2
- level 3
- level 4

Do you play video games as a leisure activity?

(single choice)

- Yes, almost daily
- Yes, a few times a week

- Yes, a few times a month
- Yes, a few times a year
- No or less than a few times a year

What genres of games do you play, if so?

(multiple choice)

- Shooter/FPS
- Strategy/RTS
- Puzzle
- Racing
- exploration/walking simulators
- MOBA/ARTS
- Card games
- Fighting/dueling
- RPGs

What is your relationship to difficulty in games?

(single choice)

- I prefer a challenge
- I don't mind a challenge, but not too hard
- I prefer easy walkthroughs
- I don't have a preference

A.2 Attachment 2 - Technical details

In this attachment, we are going to explain how to set up our project and inspect its inner workings. This thesis has been provided with a folder with several subfolders, one containing the build testing game, a second one containing the unity project and all the code, and a third one containing documentation generated by Doxygen. The folder main is also provided with a form of results from the survey from the experimental phase.

The entire thesis has been prepared on a Windows system and assumes familiarity with this system. We cannot guarantee anything working on a different operating system. Trying to test and explore the game's inner workings requires familiarity with the Unity game engine and development environment.

A.2.1 Running the game build

If you wish to run the game Deep Crawl in the version that was used as the testing version during the experiment, you may do so by opening the folder "DeepCrawl/DeepCrawl-build" and running the file "DeepCrawl/DeepCrawl-build/DeepCrawl.

For running the game we recommended at least Windows 7 or newer, with a CPU architecture of x86 or x64. There are no specific RAM requirements, but we recommend at least 4 GB of RAM.

The game is running in full-screen mode and the only way to close it is by finishing it or by pressing Alt+F4 (or a shortcut of the same function) to exit the application. The game has background atmospheric music and unfortunately no volume control. More on how to navigate in-game is in the project documentation.

A.2.2 Running the Unity project

The Unity project is what holds all the code and showcases how both the game and the procedural generation of enemies are working. To access and control the unity project, you must do the following.

1. Unzip all the files onto a location on your hard drive
2. Install Visual Studio 2020, Visual Studio Code, or another favorite script editor suited for C#
3. Install Unity Hub 3.4.1 (other versions may work just fine)
4. Open Unity Hub

5. Click ”Open” and ”Add project from disk”
6. Navigate to where you saved the unity project files
7. The Unity Hub is going to prompt you to install a desired version of Unity 2020.3.0f1
8. From the Unity Hub, install Unity 2020.3.0f1 (other versions will very likely not work)
9. Open the project DeepCrawl-Unity Project
10. In the Unity environment in the top bar, find ”Edit-¿Preferences-¿External tools” and set the script editor to the script editor you installed in 2)
11. Project is ready to be browsed, tested, built, and modified

A.3 Attachment 3 - Project Documentation

In this attachment, we provide both the programmer and user documentation for the game Deep Crawl and the implemented AGE algorithm.

A.3.1 User Documentation

You play the game by running the application DeepCrawl.exe in the game folder. The game is controlled via a mouse, and all player input in the game is performed with it by clicking the left mouse button on a UI button or in some cases simply clicking anywhere. In such a case, the text "Press Mouse to continue" will be on the screen in brown color.

Some information is obtained by hovering over objects, either over UI elements or over characters in the game. Hovering over characters on your or the enemy's team tells you their abilities and stats and is considered crucial in finishing the game.

The game consists of 4 enemy encounters and 3 campfire scenes intended for resting, healing, and upgrading your group. You control four characters, which will be situated on the left in battle in a rectangular formation. The enemy holds the same shape on the right.

The game is not 100% transparent and some information is hidden from you not to overload you with information. You are expected to try out things and discover some of the rules and interactions on your own.

You can quit the game by finishing it or by pressing Alt+F4.

A.3.2 Programmer's Documentation

In the files provided as attached, there is documentation automatically generated by Doxygen. To browse the documentation, simply run the file "Project Documentation/index.html" and a website will open in your browser, where you may browse the documentation.

The remaining programmer's documentation can be found within the code itself, as we believe most of the implementation can be understood from there. The game doesn't employ too difficult programming concepts, but there are still mechanics that would benefit from thorough explanations. In the following section, we are first going to explain the concept of Queries and describe how the scenes work and change in Deep Crawl, and finally how one character's move in

combat is resolved.

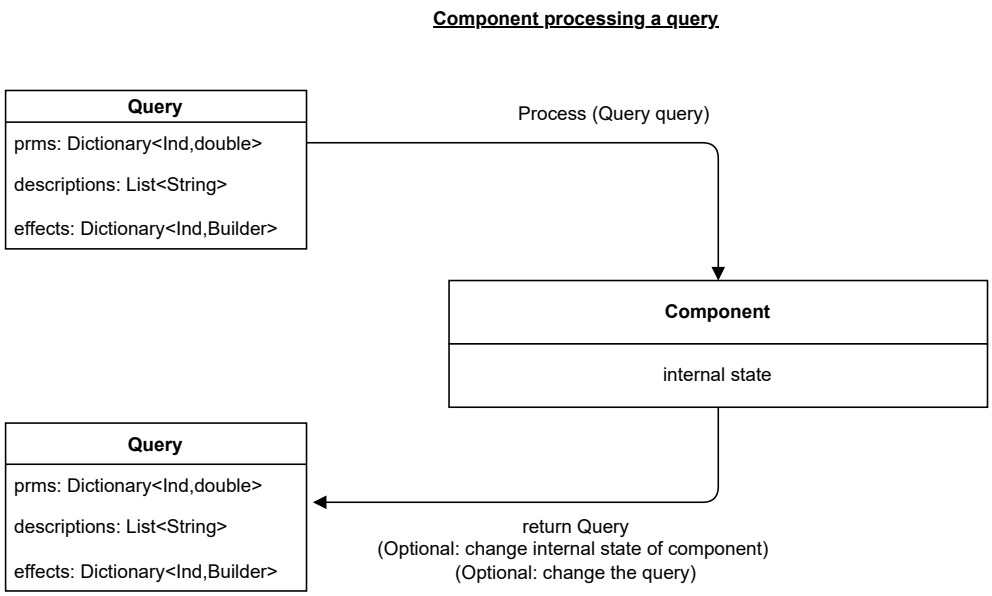
It should be noted that in the comments in the code, the following terms are used interchangeably:

- *party/group/team* - a group of four characters
- *character/creature* - an entity made of components
- *combat/battle/encounter* - a battle between two groups that ends when all on one side are dead
- *query/action* - action is a deprecated term for queries
- *component/status effect* - status effect is a deprecated term for a component

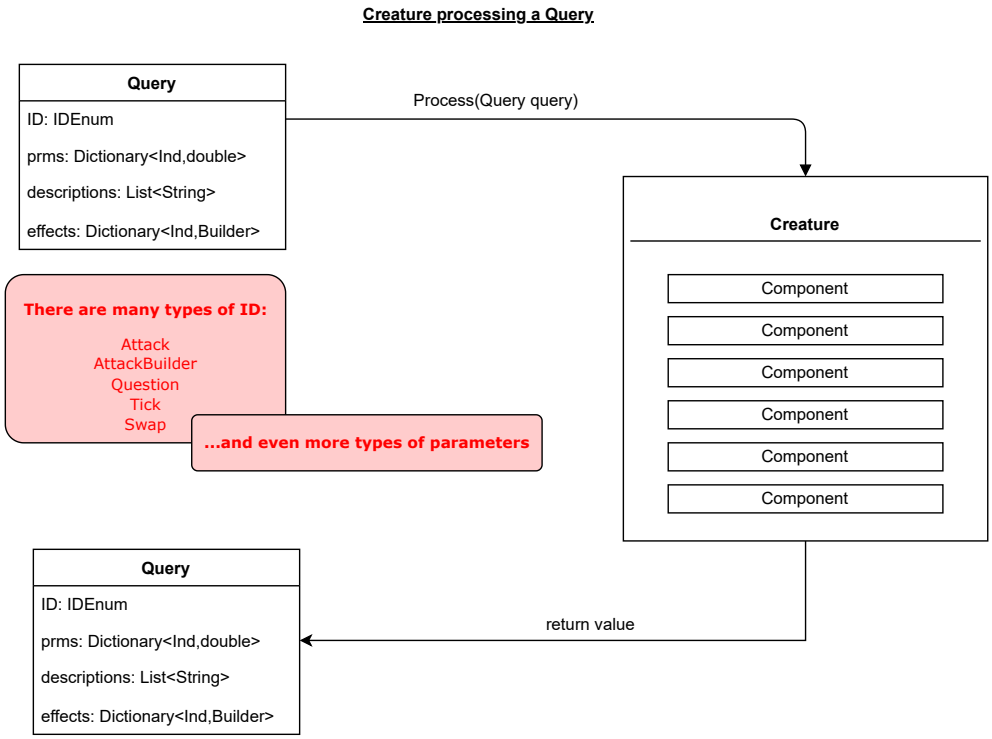
From the thesis, the reader should be familiar with the component system that makes the building blocks of all characters in Deep Crawl. The component system closely interacts with a system of queries, which is a multi-purpose communication tool among entities made of components.

Queries

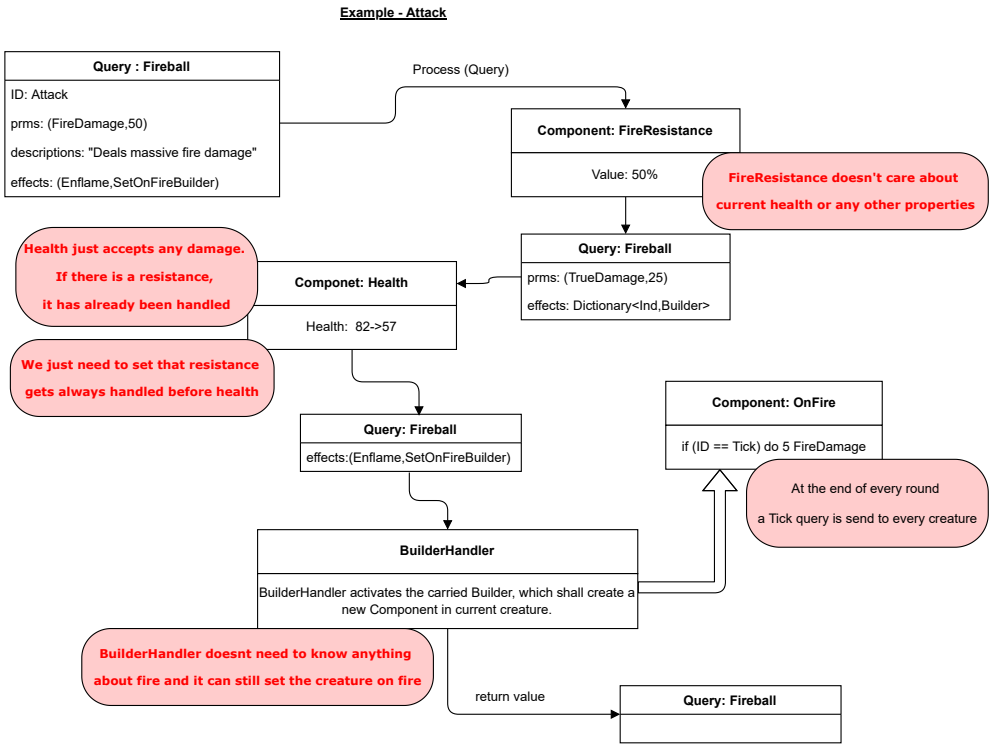
Components respond to queries, which are objects that serve as carriers of information and commands between creatures. Components of different types respond to different queries and either modify themselves or modify the query object itself, sending it modified to the next component. Most in-game commands are carried via queries, which are subsequently processed by all the components in a creature, and the resulting modified query is evaluated. The process is illustrated in the following figure.



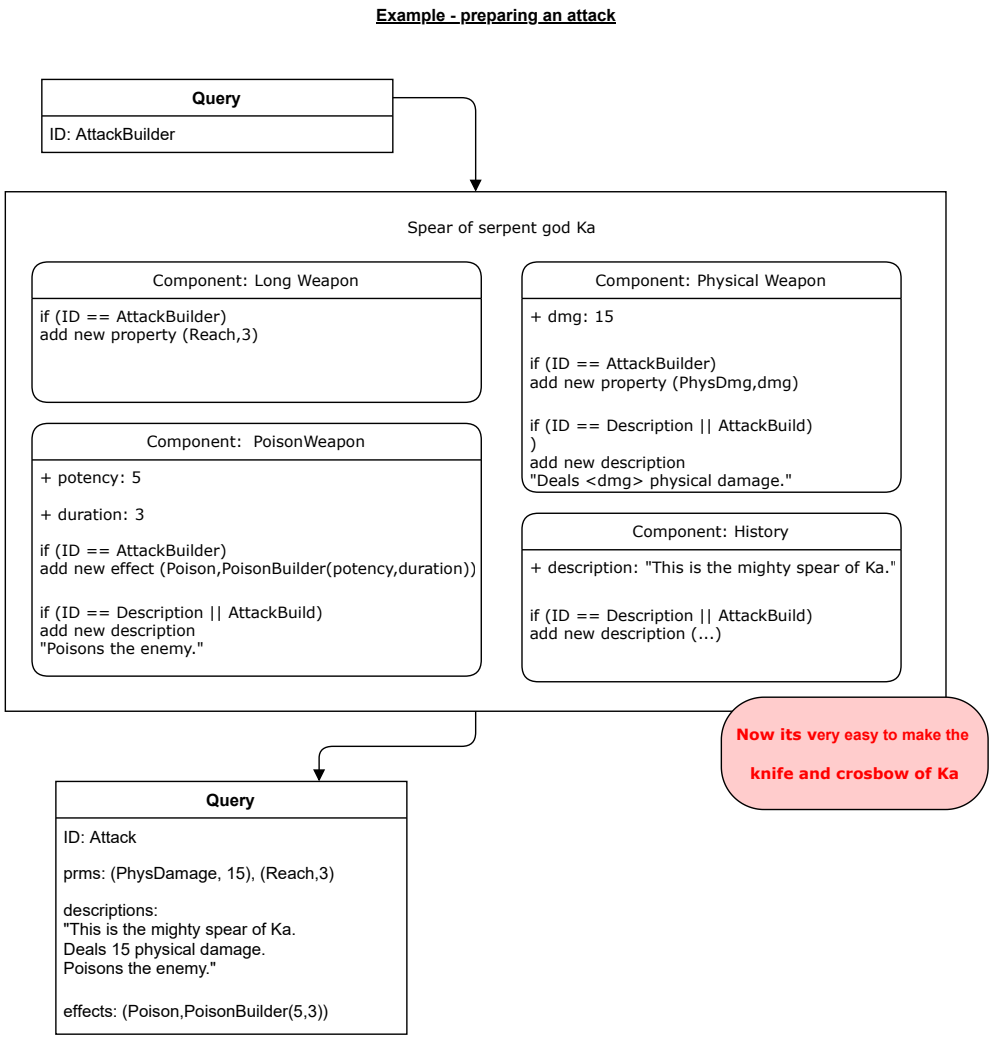
The order in which components process queries is determined by a topological ordering of the components, with each component having requirements for which component it needs to be evaluated before or after (For example, the *Armor* component is required to be evaluated before the *Health* component, so it can deduce the amount of incoming physical damage.



The query system can be used in a multitude of ways. The most common ones are very likely attacking or preparing an attack. An example of how such an attack is resolved can be found in the following figure.



The system could be expanded upon in future work due to its flexibility and simplicity in creating content. In the following example, it is demonstrated how an easy combination of components and the query mechanic can create a special attack from a legendary poisonous spear.



There are at the moment 7 IDs of Queries. There are

- Attack
- Description - for providing a description of abilities and a character's status
- AttackBuild - for preparing attacks
- Question - for checking the character's status - Are they alive? Are they able to play? Are they in full health?
- Animation - Several animations are triggered by the creature having received a special query. However, this function is not explored more widely.
- Tick - Query send to every creature at the end of a round, triggering timed effects (Stun, Poison...)
- Swap - For swapping positions of creatures. The function has been removed from the final game for unresolved bugs.

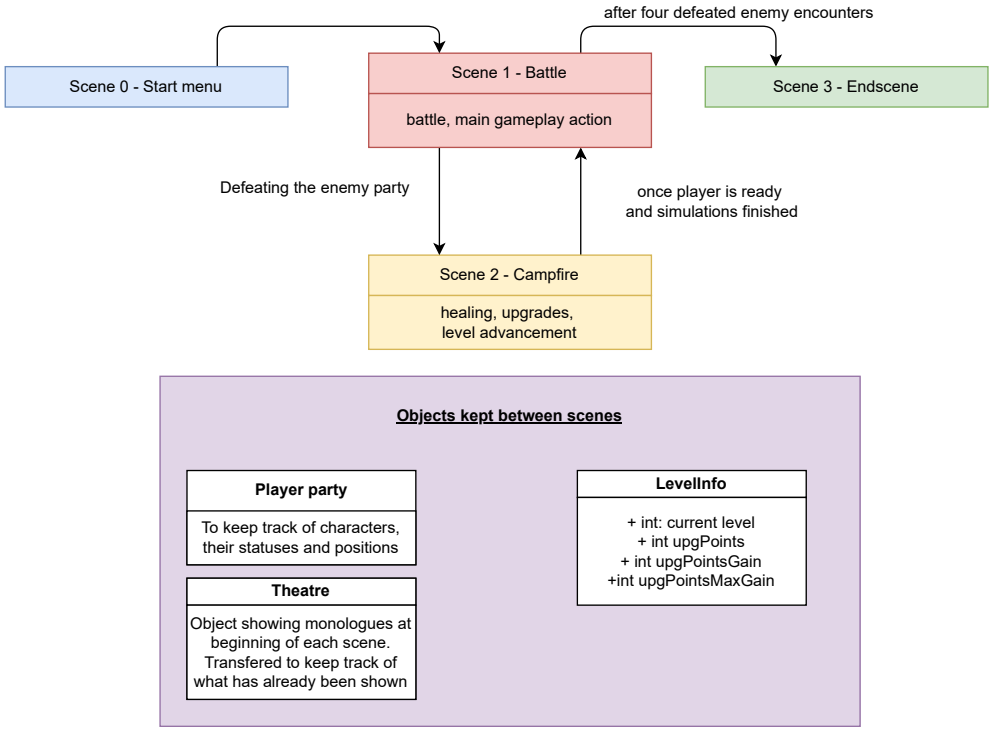
- None

Scenes

There are at the moment 4 scenes in the unity project.

- Main menu
- Game scene - for enemy encounters, is entered from the main menu and the campfire scene is exited when the player manages to defeat the enemy group.
- Campfire scene - For healing and upgrading the player's characters. In the meantime, the simulations are taking place. is entered from the game scene and is exited when simulations are finished and the player decides to go fight the next encounter.
- Exit scene - The final static image with an exit button to close the game. Is entered from the game scene after four encounters have been defeated.

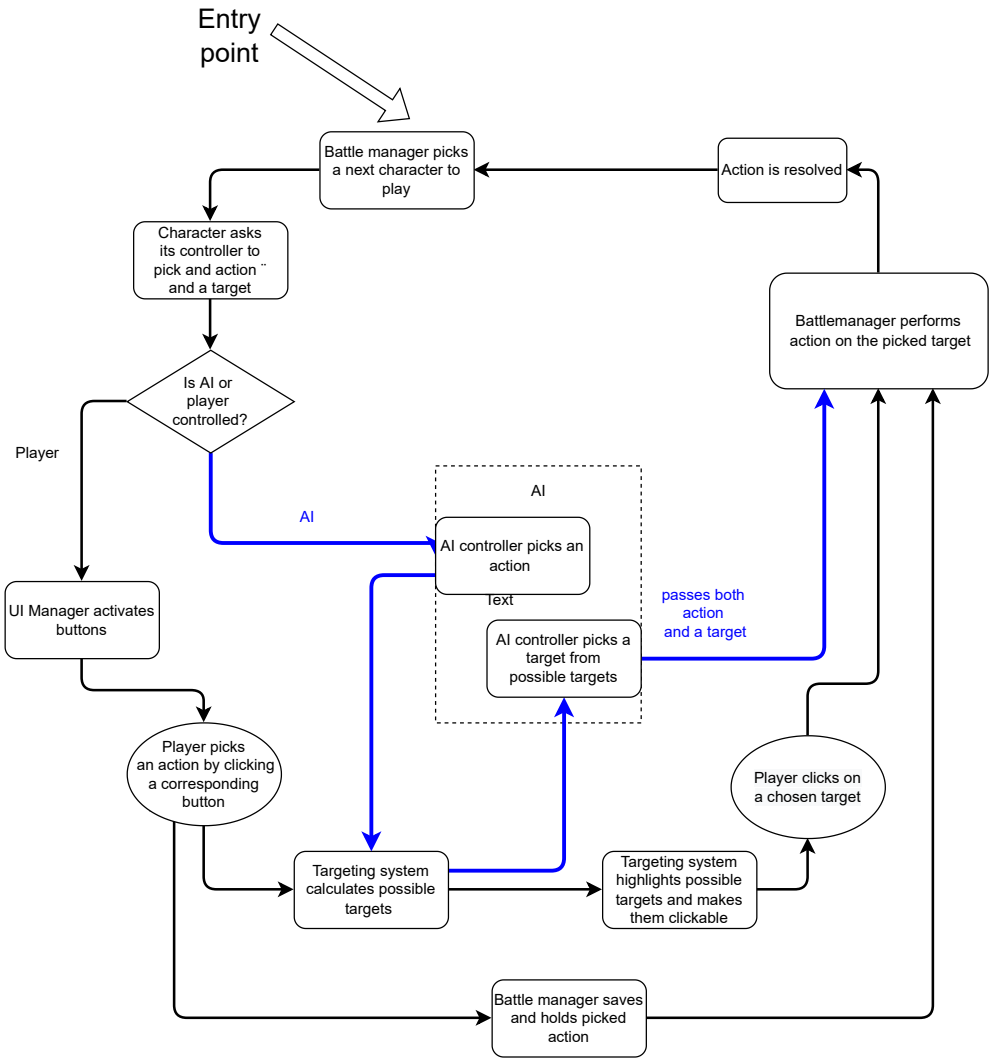
We have decided to reuse one scene for all the combats. This approach required setting up a system to load the scene each time accordingly to the desired level, but this approach also makes room for much easier possible implementation of an "endless run" type of challenge with an endless stream of encounters, which is common in some rogue-likes. The process is visualized in the following figure.



Game loop

In this section, we are not going to go into too much detail, but we will give a high-level overview of the base game loop for combat. It is highly recommended that you have played the game at this point and maybe even seen the source code, as some of the used terms may make more sense that way.

The goal of this game loop was to make it usable in both the player's game and in the simulations, only tweaking and disabling some functionality to the same processing power.



A.4 Attachment 4 - Control group

In this attachment, we are going to describe the enemy groups used in the control group for the experimental part. In both the control and test group, the respondent was faced with four groups of enemies, but in the test part only the first group was designed by a human - the author. The remaining levels were generated using the AGE algorithm. The groups for all the levels were in this order.

In the further text, *dmg* refers to damage, as in points of possible health points taken if struck by this ability.

A.4.1 Level 1

Position: Top left
Health: 50
Armor: 3
Attack: 15 dmg
Special: Claws: 10 dmg, deals 2x to clawed enemy
Speed: 2

Position: Top right
Health: 60
Armor: 3
Attack: 15 dmg
Special: Powerstrike: 15 dmg, may target only close enemies
Speed: 2

Position: Bottom left
Health: 50
Armor: 3
Attack: 15 dmg
Special: Claws: 10 dmg, deals 2x to clawed enemy
Speed: 2

Position: Bottom right
Health: 60
Armor: 3
Attack: 15 dmg
Special: Powerstrike: 15 dmg, may target only close enemies
Speed: 2

A.4.2 Level 2

Position: Top left
Health: 60
Armor: 6
Attack: 15 dmg
Special: Claws: 10 dmg, deals 2x to clawed enemy
Speed: 4

Position: Top right
Health: 60
Armor: 3
Attack: 20 dmg
Special: Powerstrike: 15 dmg, may target only close enemies
Anger: Deals 20% more dmg below half health
Speed: 2

Position: Bottom left
Health: 60
Armor: 3
Attack: 15 dmg
Special: Claws: 10 dmg, deals 2x to clawed enemy
Speed: 2

Position: Bottom right
Health: 80
Armor: 3
Attack: 15 dmg
Special: Powerstrike: 15 dmg, may target only close enemies
Speed: 2

A.4.3 Level 3

Position: Top left
Health: 60
Armor: 3
Attack: 15 dmg
Special: Claws: 10 dmg, deals 2x to clawed enemy
Anger: Deals 20% more dmg below half health
Speed: 2

Position: Top right
Health: 60
Armor: 3
Attack: 15 dmg
Special: Powerstrike: 30 dmg, hits only close enemies
Anger: Deals 20% more dmg when full health
Speed: 2

Position: Bottom left
Health: 60
Armor: 3
Attack: 15 dmg
Special: Claws: 10 dmg, deals 2x to clawed enemy
Anger: Deals 20% more dmg below half health
Speed: 2

Position: Bottom right
Health: 50
Armor: 3
Attack: 15 dmg
Special: Claws: 15 dmg, deals 2x to clawed enemy
All abilities deal bonus 5 fire dmg.
Speed: 2

A.4.4 Level 4

Position: Top left
Health: 60
Armor: 6
Attack: 15 dmg
Special: Claws: 15 dmg, deals 2x to clawed enemy
Speed: 3

Position: Top right
Health: 100
Armor: 3
Attack: 20 dmg
Special: Powerstrike: 25 dmg, hits only close enemies
Anger: Deals 20% more dmg when full health
Speed: 2

Position: Bottom left
Health: 60
Armor: 6
Attack: 15 dmg
Special: Claws: 15 dmg, deals 2x to clawed enemy
Speed: 3

Position: Bottom right
Health: 50
Armor: 3
Attack: 15 dmg
Special: Powerstrike: 30 dmg, hits only close enemies
Elem resist: 50Speed: 2

A.5 Attachment 5 - Credits

In this attachment, we would want to give credit to all the creators, whose free available assets we have used in creating Deep Crawl. All the mentioned assets are listed with their respective license.

1. *Procedural fire* by **Hovl Studio** - Standard Unity Asset Store EULA - https://assetstore.unity.com/packages/vfx/particles/fire-explosions/procedural-fire-141496

2. Music *Storm, Mountain, Feeling, Legacy* by **AShamaluevMusic** - free to use in non-commercial projects - https://www.ashamaluevmusic.com

3. *Anatomical illustrations* by **Arnaud Eloi Gautier d'Agoty** - in the public domain - https://www.rawpixel.com/board/320761/anatomical-illustrations

4. *Solar system planet pack* by **Batuhan Karagöl** - Creative Commons v4.0 - https://andelrodis.itch.io/solar-system-pack

5. *Living Tissue environment* by **ansimuz** - Creative Commons v4.0 - https://ansimuz.itch.io/tissue-environment