Kalle-Eemeli Paananen

# DESIGNING A GAME FOR EMERGENT GAMEPLAY

## Developing Gatedelvers

Bachelor's thesis

Degree programme in Game Design

2020

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

| Author (authors) | Degree title | Time |
|---|---|---|
| Kalle-Eemeli Paananen | Bachelor of Culture and Arts | November 2020 |
| **Thesis title**<br><br>Designing a game for emergent gameplay<br>Developing Gatedelvers | | 43 pages |
| **Commissioned by**<br><br>Self-commission | | |
| **Supervisor**<br><br>Marko Siitonen | | |


**Abstract**

Emergent gameplay can offer great benefits to a game's design when used effectively. The objective of this thesis was to create a design tool for adding emergence to a game and then use that tool in order to design a game product.

This thesis collected information through literary and game analysis to create a tool that designers could use to add emergent gameplay to their games. The design tool focuses on actions the designer can take during the design process to encourage emergence.

A game product was then created with the help of the tool. The game product was named Gatedelvers and is a dungeon crawler with a focus on combining emergent gameplay with procedural generation to create gameplay variety. Playtesting was conducted on the game with generally favorable results, although playtesting was limited due to external circumstances.

Both the design tool and game product can be considered to be successful, but further testing and experimentation would be necessary to confirm the tool's effectiveness.



**Keywords**

game design, emergent gameplay, thesis

**CONTENTS**

LIST OF FIGURES

# 1 INTRODUCTION

The goal of this thesis is to develop the core gameplay of Gatedelvers, a co-operative dungeon crawler with a focus on designing for emergent gameplay. The first half of this study includes analysis of existing literature about emergent gameplay and games that use emergence as a large part of their play experience, with the goal of aggregating information on how to encourage emergence in a way that improves player experience into a set of guidelines. The latter half of the thesis will focus on the process of using the collected information to design and develop a short, playable demo for Gatedelvers.

# 2 EMERGENT GAMEPLAY

Emergent gameplay is a common topic in both recent and less recent game design discussions. Many successful indie titles released during the last decade, such as Minecraft and Spelunky, rely heavily on emergent gameplay. Larger game studios such as Ubisoft have also experimented with focusing their games more on emergent gameplay (McKeand 2016).

## 2.1 Defining emergent gameplay

The lack of clarity surrounding the definition of “emergent gameplay” can make addressing the topic difficult. By one definition of the words, emergent gameplay refers to gameplay situations and options that arise in a game not as the result of an explicit rule, but the interaction between multiple, simpler rules (Sylvester 2013). However, there is also a relatively widespread alternative interpretation that requires something to be unintended by the game’s designers in order for it to qualify as emergent gameplay. The difference between these definitions can cause confusion and discord in conversations about the topic.

The definition of the word that this study will use ignores designer intent, as whether something was intended by a game’s designer is very difficult to find out and has no real effect on the player experience. In addition, when approaching the matter from the field of game design, requiring emergent gameplay to be unintentional greatly reduces the term’s usefulness to a game designer. As such,

the definition for “emergent gameplay” used for the rest of this paper will be “A situation or option in a game that arises not from an explicit game rule but from an interaction between several simpler rules.”

### 2.2 The advantages and disadvantages of designing for emergence

One of the major advantages of emergence-focused design is its ability to give the player a feeling of freedom in how they approach a game. Allowing the player to create their own plan and solution to achieve an in-game goal has multiple benefits, both within the player experience and beyond it. Accomplishing a goal can feel much more rewarding if the player feels that their method of achieving it was their own, instead of being planned in advance by the game’s designer. The challenges posed by the game can feel more authentic if they can be completed in any way that makes sense within the game’s systems, instead of being limited to specific “correct” solutions.

When emergent gameplay systems are combined with a level of randomness and instability in the game’s scenarios, either through procedurally generated levels or scenarios that are randomized in smaller ways, they can give the game a massive amount of replayability and create gameplay scenarios the developer never considered. Much of the roguelike genre and games with roguelike elements rely heavily on this combination to create the variety the genre’s repeating nature requires to work as an experience. At its best, this kind of design can result in massive amounts of unique, interesting content from a relatively small amount of actual game assets and objects by creating novel combinations of the existing pieces.

As a result of allowing players to create unique plans to achieve their goals and the non-designed situations emergent games can cause, emergent games have the potential to create unique player stories during gameplay. While games with non-emergent, scripted narratives can generally only tell a relatively limited number of stories, an emergent gameplay system can lead to a massive amount of different experiences. This can grant emergence-focused games an advantage in marketing through word of mouth, as people are more likely to share a story of

a game that's unique to them instead of being a universal experience for everyone that plays through the game. These games tend to also be a good fit for video content on platforms such as Youtube and Twitch due to their ability to generate stories; an extreme example of this is Dwarf Fortress, with Youtube channels such as Kruggsmash (Welcome to Kruggsmash… 2018) focusing primarily on telling stories of their fortresses in the game. This can be a great source of visibility and new players for a game.

A major downside of emergent game design is that in giving the player and game systems more control over the state of the game, the designer gives away control over specific parts of the experience. A game that wants to tell a scripted, coherent narrative has to be careful about giving the player tools that could let them skip important parts of the story the game is trying to tell. Trying to set up scripted, emotionally impactful moments in a game with emergent core systems can also go wrong as a result of the player's actions interfering with the emotional impact. For example, placing a barrel in the middle of a conversation between characters would likely cause the characters to resume their conversation while ignoring that both of them appear to be talking at a barrel. This could be solved by having the characters react to the interruption, but due to the massive possibility space emergent systems can create, accounting for everything is either impossible or would require unrealistic amounts of work.

Another option for combining emergent gameplay systems and traditional, more linear storytelling is to simply disable the emergent systems during important story beats. The Legend of Zelda: Breath of the Wild (2017) opts for this solution; carrying a Cucco, the game's equivalent of a chicken, to the climactic final fight against the game's primary antagonist simply causes the chicken to disappear in the cutscene before the fight, and most story beats happen in flashback cutscenes the player has no control over. While this solution can work to solve the problems emergent gameplay poses for linear narratives, it can also pull players out of the experience if their agency over the game systems is abruptly taken away at unpredictable times.

Games with a focus on emergent gameplay can be very difficult for a game designer to balance due to the massive possibility space these games present. An emergent dominant strategy can slip through until the game is released simply due to the developer not being aware of it; this makes playtesting emergent games very important. The ability to update the game after release is very useful for fixing balance issues that arise from emergent strategies players find; this makes releasing on online stores that make patching games easy, such as Steam and Epic Games Store, a very beneficial choice from the designer's perspective.

# 3 LITERATURE REVIEW

Before starting new research on how to design games for emergence, it was important to look into existing research and information on the topic. For this reason, this section of the thesis will focus on a literature review of existing works on the topic of designing games for emergence. A literature review is an account of written material on a topic (Taylor 2001).

## 3.1 The Art of Game Design: A Book of Lenses

Jesse Schell's book The Art of Game Design: A Book of Lenses (Schell 2020) includes information on many topics related to game design, including emergent gameplay. The book includes a list of traits a game's design can have that encourage emergence, similarly to the goal of this thesis' research-focused part. As such, it is a natural source to start gathering information from and can be used as a basis for the design tool.

Two of Schell's (2020) tips for creating emergence, "Add more verbs" and "Many subjects" focus on adding more subjects and rules to the game. More subjects and more ways for these subjects to interact naturally creates more possibility for interesting interactions, and density of subjects increases how often they are likely to interact with each other. Schell also recommends allowing player actions to interact with multiple objects in order to create more interactions without additional actions the player has to learn. He also highlights the importance of

goals that can be achieved in multiple ways, in order to encourage the player to experiment with the game's mechanics.

The last tip offered in the emergent gameplay section of Schell's book is "Side effects that change constraints". As an example of this, Schell uses checkers. In checkers, the positioning of each piece affects the movement options available to all others. By having the actions and state of each game object affect all others, player choices can gain great depth while the basic rules of a game remain simple.

Another potentially useful topic Schell raises in his book is the idea of gameplay modes: states of the game during which the actions available to the player change. While the book mainly approaches this subject from the perspective of avoiding player confusion regarding control schemes, it might have a lot to do with the potential for emergence in a game. Games that change their mode when dealing with different tasks, such as entering a shop, prevent the player from interacting with those scenarios with the options they have available for the rest of the game. In order to maximize potential for emergent strategies, it might be best to avoid major forced input mode changes whenever possible. Spelunky, further discussed in the Game Analysis section, is a good example of this: many of the often shared emergent stories of that game are only possible because the game's shops operate under the same rules as the rest of the game world, allowing the player to attempt to attack the shopkeepers.

### 3.2 Developing the Mechanics of Plusminus

In his thesis, Toikka (2020) discusses the design process of Plusminus, a physics-based puzzle game with a focus on emergent puzzle-solving. Much of the thesis is focused on finding a balance between realistic physics and intuitive simplicity. While realistic magnet physics naturally lead to emergent behavior, the behavior can be complex and difficult for a player to predict. In order for emergent gameplay to be meaningful to the player, it is important for the behavior to be predictable. Emergent behavior with complexity beyond the players capacity to understand can appear as random, nonsensical events.

Toikka (2020) also mentions the idea of reducing the number of variables in an object, then using the same variable for multiple purposes in order to simplify the development process for designers and developers. The example used for this was having the mass and charge of physical objects in Plusminus be proportional to the mass of the object instead of manually set. In addition to reducing developer workload, this ties these traits to a visible feature of the object, making them more consistent for the player.

## 4 GAME ANALYSIS

In order to gather more information for the creation of the emergent gameplay design tool, game analysis was conducted on games that successfully employ emergent gameplay in their gameplay experience. To gain a thorough understanding of the games analysed, it is necessary to use a combination of playing the games and watching playthroughs by other players and reading related material online (Game Analysis Guidelines 2011).

### 4.1 Spelunky

Spelunky (2012) is an action-platformer with roguelike elements such as procedural level generation and permanent character death, with the player usually getting sent back to the start of the game if their character died. The game was originally created and released by Derek Yu as freeware in 2008, and later received a HD remake in 2012. This analysis of the game will focus on the HD remake.

Spelunky has a strong focus on emergent gameplay, with much of the gameplay loop focusing on the player attempting to navigate through randomly generated levels filled with obstacles such as traps and enemies with no designer intent regarding the correct path through the level. The game starts the player with a limited amount of ropes and bombs, tools that can be used to change the level around the player. Bombs can destroy almost any wall or obstacle within the game, allowing the player to create new routes through the level (Figure 1).

Ropes can be thrown to hang vertically, then climbed in order to reach higher locations. With these two tools combined, players can skip entire levels worth of threats by reshaping the environment to create a more favorable route. The limited amount of bombs and ropes the player has to work with turns them into an option for dealing with overwhelmingly unfavorable situations: the player usually has the option of solving an obstacle by the use of these items, but has to decide whether the obstacle is dangerous or difficult enough to use them for. The game softens the blow of the most unfair or difficult outcomes of its random generation by letting the player decide whether to accept the generated challenge or skip it. The player having those universally available tools with great flexibility of use leads to many of the emergent gameplay situations the game offers.


Figure 1: Using a bomb to destroy the environment in Spelunky HD (2012). Screenshot by author.

Spelunky is also very consistent in having all of the objects in its game world follow most of the same rules. Beyond using bombs and rope, the player only has access to a few actions: moving, jumping, whipping, picking objects up and throwing. Much of the game's emergent options revolve around two of these actions: picking up and throwing. Instead of only allowing the player to pick up specific objects, nearly everything in the game can be picked up. The player can pick up items and weapons, but also unconscious enemies, treasure chests, items in stores they have not paid for yet and non-player characters, such as the

shopkeepers or “damsels”. This leads to a lot of player options a more strictly limited system would not have allowed. The player can pick up items in a shop and run out without paying, angering the shopkeeper; or they can knock an enemy unconscious and carry them to an altar to sacrifice them to Kali in exchange for powerful items. In multiplayer, players can even pick up other players and throw them to reach difficult locations.

While picking up and throwing objects is one of the mechanics the philosophy of consistent effects is most visible in, it applies to almost everything in the game: traps affect enemies the same way they affect the player, some enemies can pick up and use items in a way similar to the player and getting knocked unconscious on a Kali altar will cause the player to sacrifice themselves to Kali. With everything in the game having not just their specific behaviours but also a host of generic behaviours that apply to the category of object they belong to, everything in the game has a massive amount of potential uses, allowing players to improvise solutions in almost any situation due to everything being potentially usable in the players favor. A Rock Paper Shotgun article (Wiltshire 2016) looks into how this leads to many of the emergent stories that arise during Spelunky gameplay, and features Derek Yu and Spelunky HD programmer Andy Hull discussing how programming a game like this can lead to emergent gameplay. This object class inheritance-focused style of programming and designing gameplay elements has a history in roguelike games. The Berlin Interpretation, a somewhat controversial interpretation of what makes a game a roguelike created during the International Roguelike Development Conference 2008, featured both having enough complexity through interactions between game elements and monsters being mechanically similar to players as factors that made a game a roguelike.

### 4.2 Dwarf Fortress

Dwarf Fortress (2020) (Figure 2) is a genre-mixing colony management freeware game, at the time of writing still in development by Zach and Tarn Adams. Much of Dwarf Fortress public media attention has focused on the emergent narratives it creates, with written stories and popular Youtube series telling stories about

events that happened in playthroughs of the game. The most popular gamemode has the player attempting to manage a dwarven colony in a randomly generated fantasy world with a similarly randomly generated history. The game is notoriously complex, having numerous systems for the inner mental workings on individual dwarves, world history, political intrigue, physics and economy. This collection of complex, interlocking systems appears to be what leads to much of Dwarf Fortresses emergent gameplay and narrative.

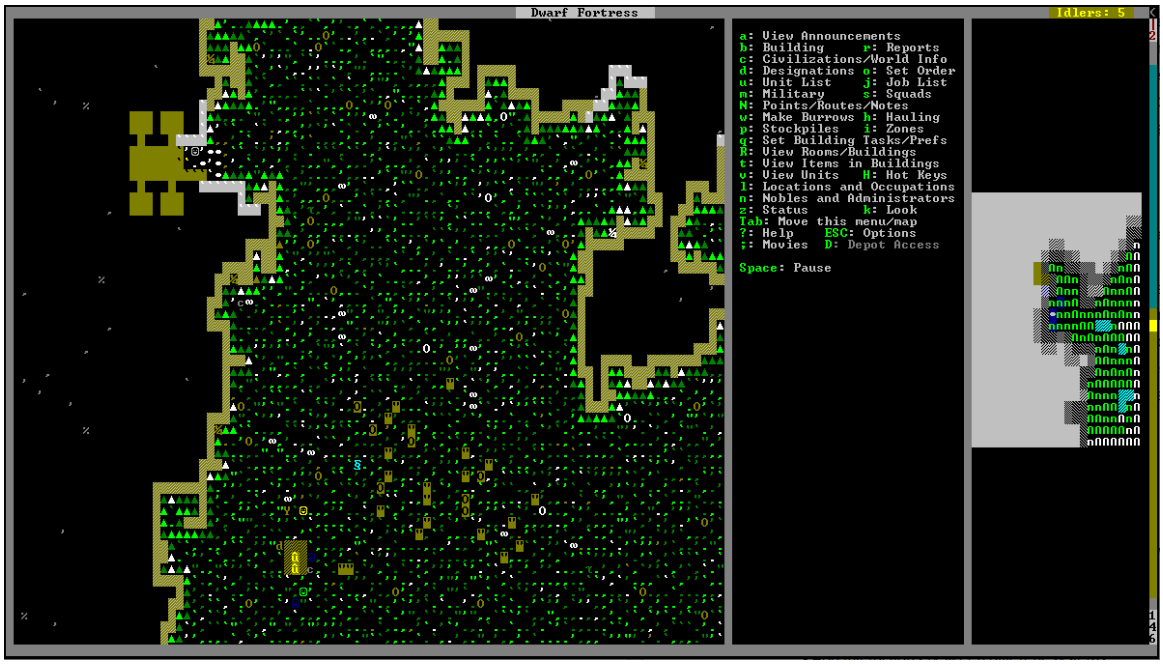



Figure 2: Dwarf Fortress Classic 0.47.04 (2020). Screenshot by author.

Much of what allows Dwarf Fortress to have been built to such a high level of complexity is the game's approach to graphics: the game world is represented primarily by colored text symbols. Due to its simple visuals, additional features need no graphical assets or animation and can be implemented with dramatically less development costs. In Noclip's youtube documentary (Dwarf Fortress creator… 2020) on Dwarf Fortress, developer Tarn Adams discusses how this ease of development led to the game growing from a smaller side project to the massive web of systems it is today.

The most notable disadvantage Dwarf Fortress suffers from its complexity-driven, non-visual approach to creating emergence is its learning curve. Dwarf Fortress is notoriously difficult for new players to learn, with fan-made mods and entire

books created for the purpose of making the game more approachable to beginners. The time investment necessary to learn enough of the game's systems to be able to play the game without the help of an external tutorial scares off many players. The actual active playerbase of Dwarf Fortress is hard to determine, as it is not yet on a platform that allows for finding active player numbers.

Dwarf Fortress serves as a great example of the extents to which emergence can be taken in even a single player game. However, its impenetrable learning curve caused by its overwhelming complexity makes it clear its approach is not practical for most games. The benefits of adding complexity to a game have to be weighed against the loss in ease of learning and approachability. Dwarf Fortress represents an extreme on this spectrum, favoring depth through complexity over approachability, and the benefits of this design strategy are visible in its somewhat niche success.

## 5 GUIDELINE SUMMARY

The information gathered from the game and literature analysis will now be used to create a design tool for designing for emergent gameplay. The goal of the tool is to provide a practical, easily understandable aid to adding emergent gameplay to a game. In order for the tool to be applicable in as many situations as possible and easy to parse, the information in the tool will take the form of a list of actions a designer can take to increase the possibility of emergence in their game.


Figure 3: A draft of the emergent game design tool (Paananen 2020)

The medium selected for the design tool was an infographic, as infographics are easy to share online and allow information to be visually grouped. Figure 3 shows an early draft of the tool, with suggested actions split into five categories based on similarities in the way the actions work. This serves two purposes; to make the information provided in the tool easier to parse, and to help designers apply the information in ways that best fit their game by highlighting the larger design patterns that lead to emergence.


Figure 4: The final version of the emergent game design tool (Paananen 2020)

The five-category structure was carried over to the final version of the tool (Figure 4). The first category in the final tool focuses on increasing interactions, as emergence inherently relies on interaction between multiple subjects or rules to occur. The second category encourages reducing constraints to allow the player to interact with the game mechanics in creative ways. The third category, “Add more subjects”, aims to increase interactions by increasing the number of things capable of interacting with each other. The fourth category focuses on making the game system unstable in order to cause interactions and emergence, as the possibility of interactions does not affect the player experience if those interactions never happen. The fifth and final category encourages connecting game mechanics with real life logic the player is already familiar with to encourage experimentation and help the player intuitively understand the game’s systems.

# 6 DESIGNING THE FIRST PROTOTYPE

The rest of this study will be focused on the design, iteration and development of a game created alongside this study, which will be referred to by the working name Gatedelvers. Gatedelvers is a dungeon crawler with roguelike elements, intended to feature meta-progression outside of the bounds of a single run and an emphasis on collaborative online multiplayer. The part of the game the rest of this study will focus on is the gameplay within a dungeon, with the meta-progression systems, multiplayer social design elements and other parts of the game's design left out. The primary design goal of Gatedelvers is to create a game that combines the replayability and depth of roguelikes with social elements to make a non-competitive multiplayer environment with staying power similar to competitive multiplayer games. These goals of replayability and depth combined with the game's multiplayer nature make it a good candidate for emergent design, and emergent stories would lend themselves well to the long-term player acquisition an online game needs to succeed.

The method used for developing Gatedelvers for this section focuses heavily on rapid iteration. This iterative approach meant that the feature focus of development shifted multiple times on a daily basis, making chronological documentation of the development process difficult to parse. To make the following documentation of the development process as understandable as possible, it will focus on each feature of the game separately. After creating a complete prototype, playtesting will be used to collect information. This information will then be employed to develop an improved version of the prototype with more content.

The prototype was developed as a one-person project, but some graphical assets used were taken from previous collaborative projects with the permission of their creators. This study will primarily focus on gameplay and game system design; the process of creating the games' visual or sound assets is not included.

The player objective of a Gatedelvers run, a single playthrough of a dungeon, is to have the player character collect as much treasure as possible and escape the

dungeon safely. A dungeon consists of multiple floors. In most floors, the player must find and reach an elevator to move to the next floor. The elevator can usually be reached through multiple means and only requires the player to stand on it to activate. This means players are not required to fight the enemies or solve the traps on a floor in order to proceed, as long as they reach the elevator. However, every fourth floor of a dungeon is a special boss floor, a level where a boss enemy has to be defeated in order for the elevator to be accessible. While the usual goal of reaching the elevator can be achieved while avoiding most threats and enemies on a level, the knowledge of an upcoming boss level should push players towards taking risks in earlier floors to acquire equipment to gain an upper hand in the unavoidable boss fight.

Gatedelvers' gameplay happens on a grid, with characters and other objects having clear locations on the grid and only moving in full squares. This is combined with real-time gameplay. Real-time grid-based gameplay is an unusual combination but serves as a good testing ground for emergent gameplay for multiple reasons. First, the clear, segmented nature of positioning on a grid can make the effects of events less vague. It is easier for players to predict the outcomes of events when the unit used for distance is measured in large, visible tiles as opposed to a unit such as centimeters, which can be difficult to gauge quickly. The unusual gameplay can also push players away from using old habits drawn from other games to approach Gatedelvers and prevent them from bringing in a mindset that prevents them from experimenting with the game's mechanics. Every object in the game world existing and clearly taking space on a grid can also help visually clarify which objects are gameplay elements, encouraging players to interact with them.

### 6.1 Selecting gameplay actions

For the gameplay to have high potential for emergence while remaining accessible and elegant, it was important for the actions available for the player to be flexible but limited in number. The actions chosen to start building this moveset from were grabbing and throwing. Being able to grab and carry objects allows the player to re-position most objects in the world, creating massive

potential for emergent strategies by re-arranging the game world into a more advantageous state, and when combined with the ability to throw picked up objects, allows players to weaponize the environment against enemies (Figure 5). By extending this ability to most enemies, players gain the option to pick up enemies to disable them in exchange for becoming unable to interact with most other things while their character's hands are full, and throwing enemies at either walls or other objects to deal damage and free themselves up to grab something else. In order to facilitate items with uses beyond throwing at things, a third primary action was added in the form of using. Use allows for the activation of held objects that have actions attached, such as swinging a sword or drinking a potion.

Figure 5: A player character throwing a barrel at a skeleton (Paananen 2020)

While being able to pick up a single object is plenty to create interesting short-term tactical situations, it does not allow for very diverse long-term strategies or items as resource management; when players can only carry one object, they are likely to default to a comfortable, consistently useful weapon or tool such as a sword and avoid experimenting with other options or grabbing and throwing objects in order to not lose their preferred object. To help with this, the player has a small inventory that allows them to store 5 items, with "items" being a subgroup of objects that are small enough to be carried around and usually have direct active uses or grant passive benefits by being carried.

The throw action was first designed and developed to allow the player to hold the throw button to determine how far and fast the held item would be thrown. This also meant that objects could be dropped on the ground without harming them by quickly tapping the throw button. Informal playtesting quickly revealed this form of input to feel rather awkward; players felt that throwing should be quicker and having to stand still while charging a throw made the action very risky. After some experimentation, throw was reworked; pressing the throw button now immediately threw the held object with maximum force. As combining throw and drop into a single input appeared to be difficult without compromising input intuitiveness, a separate drop button was added to allow for objects to be unhanded without causing damage to them or the environment. In order to balance out the new, nearly instant throwing speed the impact damage of thrown items was halved. Non-item objects, such as barrels and characters, retained their high original damage to encourage players to use their environment to their advantage.

The game is programmed with a similar structure as Spelunky, Dwarf Fortress and many roguelikes, having most game objects inherit from a single parent class in order to share common functionality such as grid movement, receiving knockback and damage, interacting with collision and reacting to attempts to be picked up. This approach combined with the universal utility of the pick up and throw commands very quickly proved their value in encouraging emergent strategies: while testing a development build, a level orb intended for allowing players to power up their characters was found to be a relatively powerful thrown weapon due to its indestructibility and heavy item status. This created an unintended strategic choice where the player could choose to delay their character’s progression a bit in order to use the level orb as a throwing weapon instead. This strategy appeared to only be worth it if the player had not yet found a better weapon, making it a situational but interesting strategy.

The possibility of adding an additional button for activating character-specific abilities was considered. Creating character classes for the player to select from would allow players to choose a gameplay style they favor before starting a run,

and an effective way to differentiate these classes would be to give each of them a single unique ability. This idea was left out for the time being, but the input system was for it was built. The computer-controlled enemies in the game, further discussed later in this section, had their special active abilities programmed as this kind of ability. This way any future mechanics that affected character abilities would by default affect both player characters and enemies similarly, and any mechanic that allowed a player to take control of an enemy character would naturally map their inputs to match the controls for player characters.

### 6.2 Enemies and traps

To create challenge and opposition for the player during gameplay, enemies and traps were added to the dungeon. These were made to work together; enemies pose an active threat, chasing down the player after detecting them, while traps are static obstacles that make the environment dangerous but can be used against enemies.

The first enemy added to the game was the skeleton. The skeleton is treated by the game very similarly to the player character; both inherit from the humanoid class, which includes the player ability to pick up, use and throw objects. While the skeleton is technically a single enemy, it can be randomly spawned holding either a sword, a bow or nothing. A skeleton that has a weapon will attempt to use that weapon’s use effect to attack the player, and an unarmed skeleton will attempt to chase down and punch the player. The information on how an AI should use a specific weapon to attack the player is not programmed into the enemy AI, it is stored within the weapons; this means any future enemies capable of holding weapons should immediately be able to use any of them. The skeleton’s three options of attack effectively turn it into three variants of the same enemy from a gameplay perspective, with unarmed skeleton being the weakest and sword and bow skeletons being very dangerous at their respective effective distances. If the player defeats a skeleton that is using a weapon, that weapon will drop and be available for the player to pick up and use. This makes fighting skeletons with powerful weapons an appealing option for a player looking to get better weaponry.

The skeleton is intended to serve as a weak basic humanoid enemy that serves to introduce the player to the mechanical traits humanoid enemies have, such as carrying weapons. While the skeleton can use weapons and attacks with equal power to the player character, making it capable of causing a lot of damage quickly, it only has 2 health and can be defeated in a single hit from a weapon or thrown large object. This combination of high damage but low survivability on the first enemy type in the game is intended to encourage players to think about how they approach fights: skeletons are very easy to defeat before they get a chance to harm the player, but running at them without a plan or attempting to fight a large group at once can be very punishing.

Two variants of the skeleton enemy were also created. The Armored Skeleton largely functions identically to the regular Skeleton but employs a game mechanic called armor: the damage of any attack against an armored skeleton is reduced by 1. The Armored Skeleton retains the Skeleton's low maximum health of 2, meaning a single attack of 3 or more damage will defeat it instantly. Thrown small items or unarmed punches only deal 1 damage, causing them to have no effect on the armored skeleton; in order to defeat one, the player has to use a proper weapon or environmental hazards such as traps or large objects. The safest way to defeat regular skeletons is often quickly throwing some items at them from a distance; the armored skeleton forces players to use a different approach or avoid fighting it entirely.

The other added variant was the Chaos Mage, a skeleton that teleports to random locations near the player and creates explosive spheres to attack. These spheres can be picked up and thrown by the player before they explode. Chaos Mage was added to create some enemy variety in the latter levels of the prototype, and only spawns on floors 3 and 4. Figure 6 shows all of the skeleton enemy variants featured in the prototype.

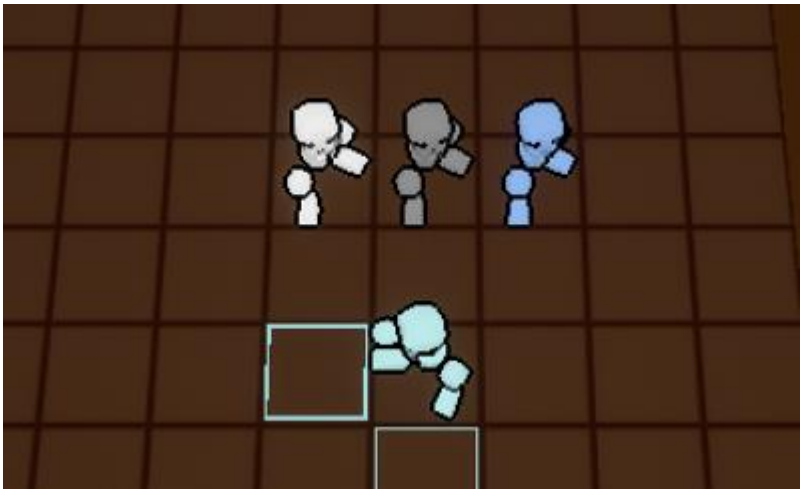


Figure 6: A skeleton, armored skeleton and chaos mage near the player character (Paananen 2020)

Two traps were also added to the first version of the game, the first of which was the spike trap (Figure 7). The spike trap takes the form of a tile-sized plate on the floor with an array of large, round holes on it. Placing a weight, such as the player character, on the plate causes spikes to rise out of the holes after a short delay, damaging anything on top of it. The spike trap is not very threatening on its own, as the player character moves fast enough to be able to run over it before the spikes have a chance to damage them. As such, it mainly exists to introduce the player to traps and recontextualize other traps and enemies around it.


Figure 7: A spike trap activating behind a player character (Paananen 2020)

The other trap added to the first prototype was the arrow trap, which consists of two parts in the game world. First is the trap itself, a solid block of wall with an opening that fires an arrow in the direction it is pointing when activated, and the other is a pressure plate that activates the trap it is assigned to when a weight is placed on it. Unlike the spike trap, the arrow trap cannot be avoided by running through it; the arrow is fired immediately on activation. Outside of simply avoiding the pressure plate, the player has a few options to get past it. They can put a barrel or similar large object in the line of fire of the trap to block the arrow when it is fired, or they can place an object onto the pressure plate to activate the trap before entering its line of fire. The arrow trap is meant to force players to think about the systems of the game beyond basic combat mechanics, as it is effectively impossible to get past safely without using external objects to disarm it. The arrow trap can also be used to the player's advantage, as throwing an enemy onto the pressure plate is an easy way to deal at least 2 damage.

### 6.3 Objects

Barrels and crates were added as environmental objects the player could use to their advantage while navigating the dungeon. They take up 1 square on the game grid and can be picked up and thrown at enemies for 2 damage, making them a very powerful choice for long range damage; however, they are destroyed in the process regardless of whether they hit an enemy or a wall, making them a limited resource, and due to being too large to fit in the player characters inventory, the player can only carry one with them.

After being thrown, barrels start rolling in the direction they were thrown in. This means they do not lose momentum as they travel, making them more effective as thrown weapons. Crates do not share this trait, but contain an item that is revealed when they are destroyed, making them serve as minor objectives for the player. As crates guarantee a random item, the player will usually want to destroy any crates they see in hopes of finding something useful.

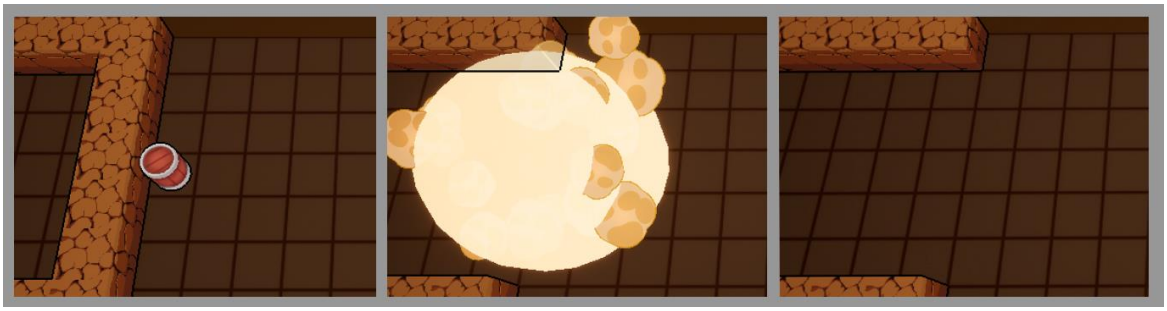


Figure 8: An explosive barrel exploding and destroying a wall (Paananen 2020)

Explosive barrels, a variant of regular barrels, were also added. Explosive barrels behave largely identically to regular barrels but cause a large explosion when destroyed (Figure 8). This explosion can cause heavy damage to characters and objects and destroys any walls in the blast radius. This means the player can use any explosive barrels they find to either quickly defeat powerful enemies or to destroy a wall blocking their way.

### 6.4 Items

The items a player is carrying effectively determine the actions available to them; while the player can always pick up and throw nearby objects or enemies, each item the player is carrying usually offers them another action they can choose to use in any situation. Creating a large variety of interesting, varied items would give the game great replay value, as finding different items during a playthrough would result in the player having different options available to them during gameplay. However, for the purposes of the first prototype, a small initial amount of items were created.

The most simple item added to this first prototype was a rock. The rock item does not do anything, but like all items, it can be thrown at enemies or placed on traps to trigger them safely. From a design perspective, it serves two purposes. The existence of the rock encourages players to think about what the use of an item that does nothing could be, pushing them to experiment with throwing and triggering traps. In addition, an exception was added to the level generator in order to always spawn a rock in the first room of the game, in order to prevent situations where the only way forward required getting past traps that were

impossible to pass without the use of an item. The rock placed in the first room meant that players would always have an item available for disarming traps, but its uselessness compared to any other item meant that players would still replace it at the first opportunity, preventing it from reducing run variety. If players were to start with a more powerful weapon, they might stick to using that single weapon instead of experimenting with other items they found; as any item is a direct upgrade over the rock, this should not be a problem.

Sword was added as the first close range weapon. When used, the wielding character swings it in a target direction, dealing 2 damage in a small area (Figure 9). Unlike punching, the sword can hit diagonally, effectively giving it greater range. Bow was added as the ranged alternative. Pressing the use button causes the holder to start drawing the bow, and releasing the button fires an arrow if the bow was drawn for long enough (Figure 10). The bow usually deals 1 damage but timing the release with a flash visual causes the arrow to fly faster and deal 2 damage.

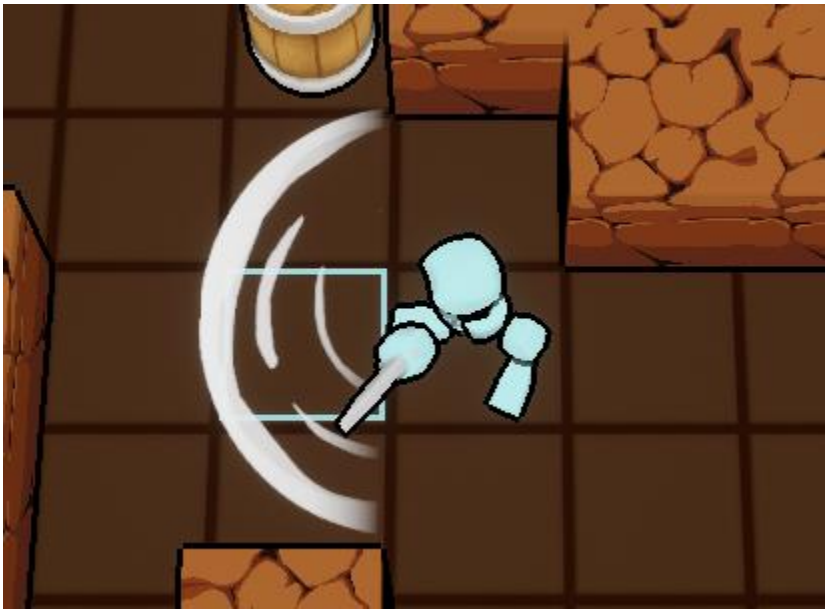


Figure 9: The player character swinging a sword (Paananen 2020)

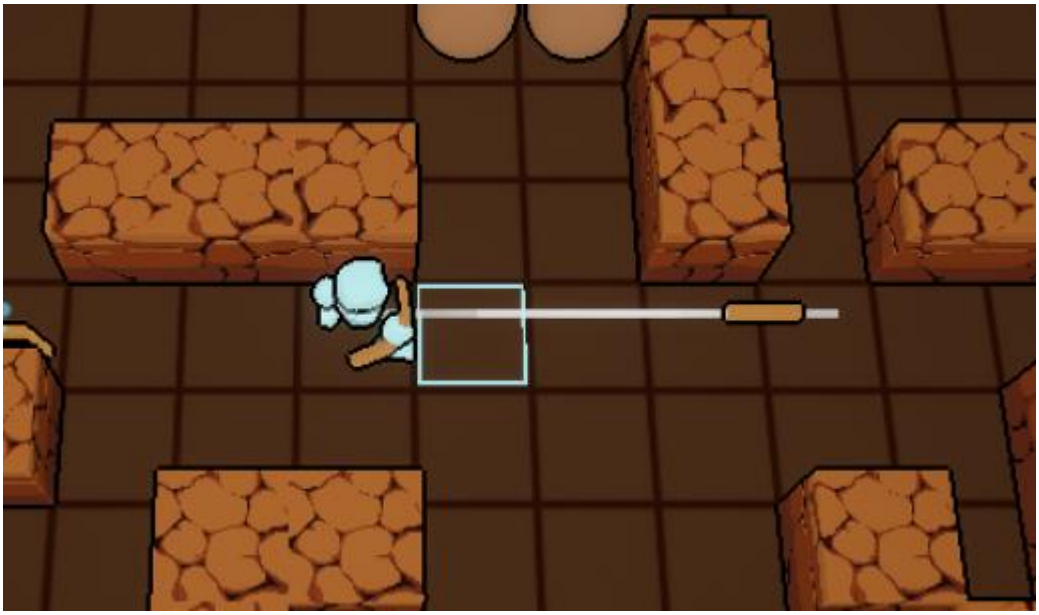


Figure 10: The player character firing a bow (Paananen 2020)

An issue with weapons that can be used an unlimited number of times is that after finding a weapon, the player has no use for additional weapons that fulfill the same role. As an example, a player finding a sword while already carrying one has no reason to pick up another; the found item has no value to the player. In addition, players have reduced need to improvise, as they can always use a weapon to solve combat scenarios after finding one. As this outcome goes against the design goals of Gatedelvers, some options to change it were considered. The Legend of Zelda: Breath of the Wild (2017) solved this issue by giving all weapons limited durability, causing them to break after a number of uses. This solution helped prevent players from sticking to a single weapon and pushed players to engage with the game's many emergence-focused gameplay systems. However, all weapons having limited durability made finding new weapons less exciting and encouraged hoarding behavior, with players refusing to use their best weapons to avoid breaking them.

The solution chosen for Gatedelvers was a gameplay system called Enchantments. Each time the game creates a weapon, it has a chance of applying a passive effect called an enchantment to it. The only enchantment added to the first prototype was Sharpness, which increases a weapon's damage by one. Enchantments have a limited amount of charges, which they lose

whenever the enchantment's effect is activated, such as dealing damage with a weapon that has Sharpness. In this way, enchantments function similarly to Breath of the Wild's durability, but are presented to the player as a positive bonus effect instead of a negative as the player still gets to keep the weapon once the enchantment runs out. The system still encourages players to carry around more weapons in order to have enchanted weapons available when they need them. The hoarding issue present in more standard durability systems is likely lessened by Gatedelvers' run-based nature and small inventory size in combination with the reduced feeling of loss of having a weapon run out of enchantment charges instead of breaking.

### 6.5 Trinkets

In addition to items that are intended for use as weapons or tools, another item type that grants passive benefits to a character when carried in their inventory was added. These items were named trinkets. The goal of trinkets was to create gameplay variety between playthroughs by changing the abilities of the player character depending on the trinkets they found and create a greater incentive to look for more items. Due to the nature of active items, collecting more than one of each has no additional benefit unless the items have enchantments applied, so the player has no reason to look for more items after finding the two weapon types. Trinkets grant an advantage as long as the player is carrying them, so the player will always want to fill up their remaining inventory with them, and choosing whether to use inventory space for additional trinkets or enchanted weapons offers an interesting choice for the player.


Figure 11: The in-game tooltip of the Healing Leaf trinket item (Paananen 2020)

The two trinkets added to the first prototype were Healing Leaf (Figure 11) and Power Glove. When held, Healing Leaf restores 1 health to the carrying character after each floor. The item has no effect for non-player characters, as they cannot travel between floors. The Power Glove improves the players throw distance and how long they can carry enemy characters. Both of these items can only be found inside crates.

### 6.6 Level Generation

While the first gameplay prototype and experiments were made in a small, handcrafted level, the run-based nature and emergent design goals of Gatedelvers made randomly generated levels an easy choice for dungeons. Static, handcrafted levels would have players falling into routines on repeat playthroughs, moving focus away from improvisation towards memorization. In addition, the grid-based nature of the game world made the development of a random level generator much easier on the technical end.

In order to get the benefits of randomly generated levels while still having the generated levels have interesting small scale challenges and a feeling of intentionality, the approach selected for generating levels was to have each floor consist of a larger room grid populated by hand-crafted rooms with minor in-room

randomization. After some experimentation, one floor was decided to be a 6 by 6 grid of rooms, with each room taking up 7 by 7 tiles on the gameplay grid, making a single floor take up a total of 42 by 42 tiles on the gameplay grid. Originally a room size of 6 by 6 was tried, but due to the rooms outer walls usually taking up one line of tiles at the edges of each room the amount of space inside a room ended up often being too small for interesting self-contained rooms, so the size was increased to the current 7 by 7.

In order to ensure that all floors generated by the level generator are consistently beatable, the generator starts each floor by placing the entry room from which the player character starts, then builds a path towards the other end of the room from a collection of “path rooms”, which are designed to ensure they can always be traversed without the use of additional tools, although attempting to do so can pose a high risk to the player character’s health. After the path can no longer travel further away from the start room due to hitting the edge of the level, the generator spawns the exit of the floor in an empty room tile adjacent to the final room of the path. Starting the level generation process like this not only ensures that the player always has at least one relatively fair route to the floor exit but also means that the floor entrance and exit points are usually on different sides of the map, avoiding situations where the exit door is right next to the entrance.

After ensuring every floor is at least possible to clear, the rest of the level generation can follow less strict structural rules. In each standard floor of a dungeon in Gatedelvers, the player has two primary goals: To reach the exit elevator of the floor with minimal damage to their character, as methods for recovering the health of the player character are sparse, and to collect as much valuable resources as possible. While the main path through a floor exists to make the first goal fair, the purpose of the rest of the level is to make the option presented by the second goal as interesting as possible. There are multiple potential resources for the player to gain by taking the risk to explore the entirety of each floor, but the most important one is the an object that will for this study be referred to as the level orb. The generator spawns exactly one level orb in every standard floor of a dungeon, and finding and activating the orb allows the player

to select a bonus that their character gains for the rest of the run. This means that in most situations, the player will want to find the leveling orb before leaving a floor, as leaving a floor without activating the orb ensures that the player character will miss out on additional power for the rest of the dungeon run. After generating the main route through a level, the level generator goes through every potential doorway in the path rooms and attempts to place a room on the other side of that doorway, with these rooms selected from a large pool of highly varied rooms. It repeats this process once, also attempting to place rooms past potential doorways in the newly added rooms during the repeat. However, instead of using a random room, one of the doorways selected for this repeat will always have an empty room with the leveling orb placed behind it. This method of placing the orb ensures that it is usually placed more than one room away from the main path that leads from the entrance to a floor to the exit, encouraging the player to explore and see the majority of the level before leaving. After placing all rooms, every room that has a potential doorway leading to a location that already has a room in it will check if that room allows for a doorway there, and if so, has a 50% chance of creating one. This turns the navigable structure of the level from a single path with multiple dead end branches into a more interconnected whole by connecting the branching paths to each other at random locations. An example level created by the level generation algorithm (Figure 12) has a combination of dead ends and connecting paths.

Figure 12: A level created by Gatedelvers' level generation algorithm (Paananen 2020)

The two primary player goals in the forms of the elevator and the leveling orb in combination with a somewhat interconnected floor structure filled with various threats makes the act of navigating levels a constant series of small choices about which path to take and larger choices such as whether to leave safely after finding the elevator or to explore more of the floor to find the leveling orb. Rooms often also recontextualize each other through the ways they connect; a room filled with enemies becomes much less threatening if the player approaches it from a room filled with explosive barrels they can use to their advantage, and a trap room can become much more threatening if a barrel-launching trap in an adjacent room is constantly hurtling barrels through it from a doorway.

### 6.7 Boss

The fifth floor of the prototype dungeon was made by hand instead of using the level generation algorithm and includes an entrance path and a single large room

with a boss the player has to defeat to beat the prototype. The boss is called Yorick and is a variant of the Skeleton enemy, with drastically increased health and unique attack patterns. Yorick will punch nearby players exactly like a standard unarmed skeleton, but also periodically creates clusters of the bombs Chaos Mages use near the player, causes barrels to roll from the sides of the room or summons more enemies to assist him. These attacks mainly consist of reused assets and objects from other parts of the game, as Yorick was created with minimal development time in order to get playtest information on how bossfights in Gatedelvers play out in practice. Defeating Yorick causes the game to announce the player's victory.

## 7 EVALUATING THE FIRST PROTOTYPE

With the first prototype complete and playable, informal playtesting was conducted to collect information for the next development cycle. Due to the COVID-19 pandemic preventing in person playtesting sessions at the time of writing, playtesting was done by having players livestream their gameplay through Discord. An effort was made to avoid giving the player additional instructions during the playtesting sessions, but due to the lacking in-game tutorialization at the time some external help was occasionally needed.

Players picked up on the game's real-time grid-based gameplay quickly, alleviating worries about the unusual movement controls. The decision to start players without weapons seemed to work as intended, with players quickly resorting to improvised weaponry and stealthy approaches. This seems to have led to players experimenting more with the game's mechanics in general. Players quickly started using items to set off traps and explosive barrels to destroy walls in their way.

In addition to planned emergent gameplay, multiple positive instances of unintentional emergent gameplay arose; players started using spike traps to trap enemies in dead ends by using the enemy AI's trap avoidance to their advantage, and one playtester built a defensive wall out of barrels. Players also started

intentionally activating arrow traps in such a way that the arrows fired hit enemies around corners.

The playtesting results were exceptionally positive. In spite of the game having no sound effects and generally lacking in polish at this stage, most players wanted to keep playing until they beat the game.

## 8 DESIGNING THE SECOND PROTOTYPE

As most of the gameplay mechanics introduced in the first prototype performed well in playtesting, the second prototype focused primarily on adding more gameplay content and improving on what was already in the game.

### 8.1 Level cards

In the first prototype, the three powerup options available to the player when activating a levelling orb were always the same. In the second prototype, each of the three choices available in the level up screen (Figure 13) represents one of three decks of powerup cards the player character had assigned to them. When levelling up, each of these 3 decks is shuffled, then the topmost card on the deck is offered as one of the options. Selecting a card removes it from the deck and each deck only has one copy of each card. Each of the three decks is focused on a specific playstyle and has an associated color. The red deck grants advantages to aggressive, straightforward gameplay by granting increased character survivability, the green deck offers mobility and ranged benefits, and the yellow deck grants item-related utility. This deck-based system means every powerup can only be gained once and its random nature encourages experimenting with different strategies, as the player cannot pick the same options every time.


Figure 13: Powerup selection UI with placeholder graphics (Paananen 2020)

This deck system also sets up a future progression and customization system for the game in the future. The player could be allowed to customize their decks before starting a dungeon run, and different character classes could have different combinations of deck colors available to them. A character with multiple decks of the same color could get two copies of the same powerup card, giving these characters a unique advantage in exchange for a loss in flexibility.

### 8.2 Pitfalls

While the traps introduced in the first prototype worked well to create areas that were difficult or interesting to traverse, there was no trap that served the simple purpose of making an area inaccessible by default. With much of the game's core mechanics built around characters and objects being launched through the air, either through explosions or being thrown, pitfalls became a natural choice for an environmental hazard with natural interactions with the existing game mechanics.

Pitfalls (Figure 14) are section of the play grid with no floor present, and any object affected by gravity that comes to a halt above one falls into it. The outcome of falling into a pitfall went through a few iterations. A prominent idea was that falling into a pitfall would cause the falling object to drop into the floor below the current one; this way the player could escape into the next floor in

exchange for receiving falling damage, or throw enemies into a pit to delay having to deal with them. However, this solution caused issues when combined with the instanced nature of floors and the multiplayer plans for the game. A player falling into a pitfall in a multiplayer game would cause that player to move onto the next floor before the rest of their teammates, which would require the server machine hosting the game to have to host two floors simultaneously. Alternatively, the player could only land on the next floor after the rest of the players had moved on as well, but this would cause a waiting period for the fallen player. The frustration of this waiting period would likely be amplified if the player's character died immediately after landing on the next floor, taking them out of the game again. Due to the combination of player experience problems and technical complexity, this design was scrapped.

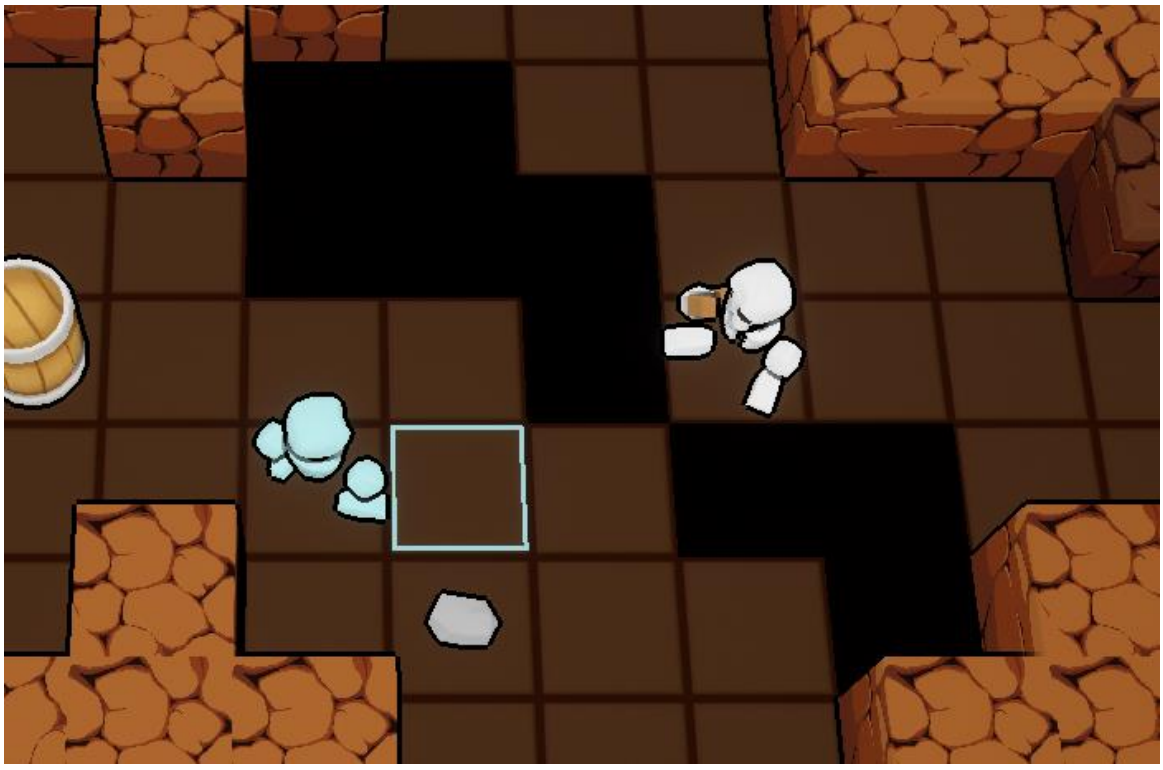


Figure 14: A pitfall between two characters (Paananen 2020)

The version of the pitfall effect that was eventually added to the second prototype had any objects that fell into the pitfall fall back onto a random location on the floor the pitfall was on. While falling onto the floor you were already on is nonsensical from a spatial perspective, this solution was much better for the flow of the game and drastically simpler to execute. Due to the falling damage, pitfalls

were dangerous and usually something the player wanted to avoid, but in an emergency, they could still be used as a final option for escaping. This can also create comedic situations, such as when the player throws an enemy into a pitfall and the fallen enemy lands back right next to the player.

### 8.3 Rock Crabs

The Skeleton enemy and its variants are dangerous enemies but quite fragile, making the best option for dealing with them usually an ambush or direct attack. In order to create variety on the strategies enemies should be approached with, Rock Crabs (Figure 15) and Boulder Crabs were created. Rock Crabs are enemies that disguise themselves as boulders, but if a player walks onto a tile in a cardinal direction from them within their vision range, launch themselves at the player, inflicting heavy knockback and damage.

Figure 15: A rock crab tackling a player into a wall (Paananen 2020)

Rock Crabs are designed to not be worth the effort to defeat. They have 2 armor, making most attacks completely ineffective against them, and they only drop low-value rocks when defeated. In addition, their extremely predictable attack patterns make them largely unthreatening as long as the player is aware of them

and their location. As such, the best option for dealing with Rock Crabs is usually evasion, making them something of a mix between an enemy and a trap from a gameplay perspective.

### 8.4 Additional items

Two new items focused on interacting with the environment were added to increase the options available to the player for traversing the dungeon floors. Dynamite can be used to deal massive area damage or destroy walls, allowing the player to create a new path through the level. The dynamite explosion is identical to the explosion caused by the explosive barrel.

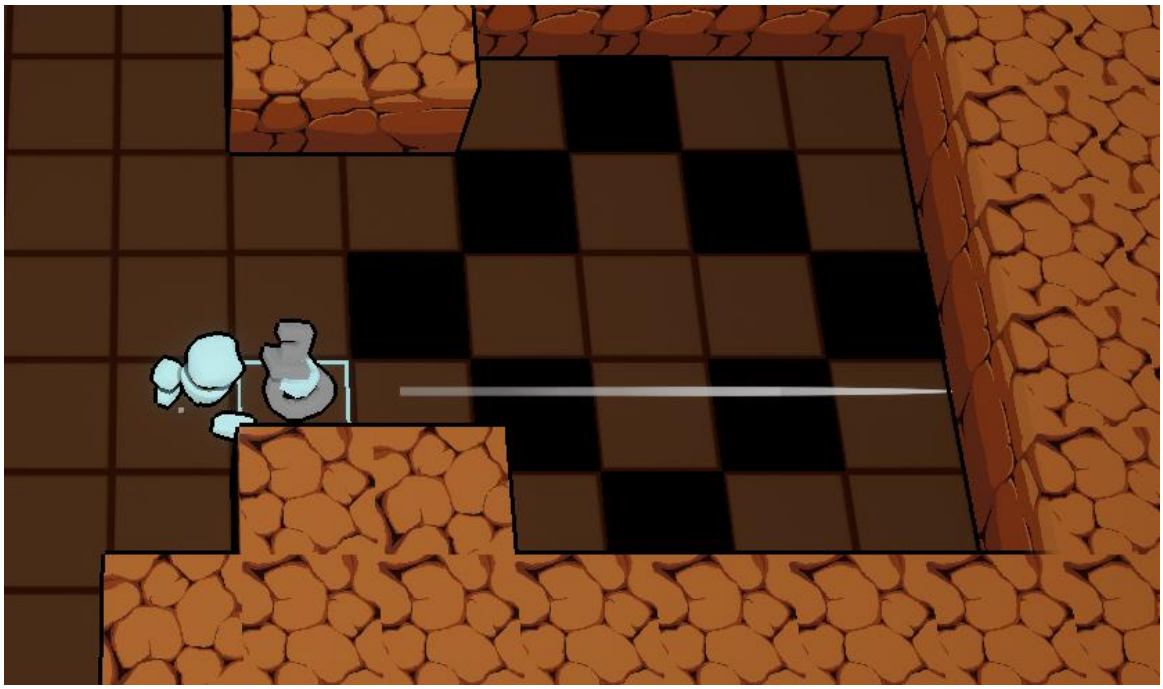
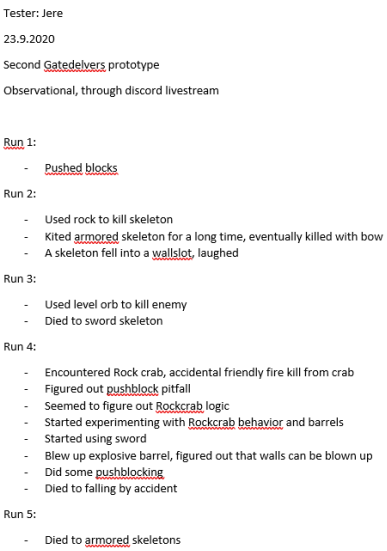


Figure 16: A player character using the grappling hook across a pitfall (Paananen 2020)

The other added item was the Grappling Hook, which could be used to either pull the using character to a wall (Figure 16) or to pull objects or enemies to the player. The Grappling Hook's controlled flight on demand allows players to get past pitfalls if there is a wall to latch onto and a floor to land on behind them. It can also be used to pull out treasure from otherwise inaccessible locations and interacts with all objects in the game, providing the item with a variety of creative uses.

# 9 EVALUATING THE SECOND PROTOTYPE

Playtesting for the second prototype was again done by observing first time players playing the game through a livestream. Four players with past experience playing various genres were watched play the game for about an hour each. During the playtesting sessions, any in-game events related to emergent gameplay or the game's learning curve were written down into a document (Figure 17) to be evaluated later.



Tester: Jere

23.9.2020

Second Gatedelvers prototype

Observational, through discord livestream

Run 1:

- Pushed blocks

Run 2:

- Used rock to kill skeleton
- Kited armored skeleton for a long time, eventually killed with bow
- A skeleton fell into a wallslot, laughed

Run 3:

- Used level orb to kill enemy
- Died to sword skeleton

Run 4:

- Encountered Rock crab, accidental friendly fire kill from crab
- Figured out pushblock pitfall
- Seemed to figure out Rockcrab logic
- Started experimenting with Rockcrab behavior and barrels
- Started using sword
- Blew up explosive barrel, figured out that walls can be blown up
- Did some pushblocking
- Died to falling by accident

Run 5:

- Died to armored skeletons

Figure 17: Notes taken during a playtesting session (Paananen 2020)

The strengths of the first prototype remained visible in the second prototype's playtests, with players quickly learning the game's mechanics without external help by experimenting with the basic actions. The goal of increasing interesting emergent situations by adding more objects with interactions between each other in rooms was successful, with interesting chain reactions and unique situations arising very consistently during playtests. A side effect of this approach was increased difficulty: player fragility combined with many dangerous moving pieces led to most playthroughs ending in abrupt character death. None of the players partaking in the playtest beat the prototype, however, players seemed to find the complicated ways they died entertaining. Regardless, difficulty should be toned down for future development.

The most common problem to arise during playtesting was that the grid movement system could be unclear during combat situations and the outcomes of situations such as multiple characters moving to the same tile simultaneously were hard to predict. The movement system should likely go through a few more design iterations to find a good compromise between position clarity and intuitiveness.

From the results of the playtest, the prototype appears to have achieved its design goals. Despite the relatively small amount of designed content, the game generated a large number of unique situations. Each player developed a unique approach to progressing through the game, favoring different options to get past obstacles. With the addition of further content, the game should be able to expand on this base of player expression and replayability. The game appears to have potential for commercial development.

## 10 CONCLUSION

The goal of this thesis was to gather and aggregate information on how to design a game to encourage emergent gameplay, create a design tool that designers could use to increase emergent gameplay in their games, and use that tool to create a game that uses emergent gameplay to improve the player experience.

To collect information for the creation of the tool, literary and game analysis was conducted. The information from these was then collected and formatted into a tool that offered a collection of actions a designer could take in order to increase emergent gameplay in their game. The tool was then used to design and develop two Gatedelvers prototypes.

Gatedelvers was developed with the goal of creating a game with high replayability and gameplay depth with the combined use of procedural level generation and emergent gameplay. The core gameplay and content were designed with the help of the tool to encourage emergence. A prototype was created and subsequently playtested, with the information from the playtesting used to create a second prototype. Playtesting was then performed on the

second prototype. While the reliability of the playtesting results was likely hindered by the limited scale and method, the playtesting results were positive. The Gatedelvers prototypes mostly achieved their design goals, with different players having unique experiences, creating their own strategies, and encountering unique situations after more than an hour of play even with a very limited amount of gameplay content.

The effectiveness of the design tool is difficult to gauge, but it proved effective for the development of Gatedelvers. In order to determine how well and consistently the design tool achieves its purpose, it would need to be used by more developers on multiple different projects, with the results of this use evaluated.

While both the Gatedelvers game product and the design tool would require further testing to determine their success more confidently, in the constraints of the results from the limited testing conducted, this thesis can be considered a success.

## REFERENCES


The Legend of Zelda: Breath of the Wild (standard edition). 2017. Nintendo Switch [Game]. Nintendo: Kyoto.

Dwarf Fortress (0.47.04). 2020. Microsoft Windows [Game]. Bay 12 Games.

Dwarf Fortress Creator Explains its Complexity & Origins | Noclip. 2020. Noclip. Video documentary. Available: https://www.youtube.com/watch?v=VAhHkJQ3KgY [Accessed 25.9.2020.]

Game Analysis Guidelines. 2011. Massachusetts Institute of Technology. PDF file. Available: https://ocw.mit.edu/courses/comparative-media-studies-writing/cms-300-introduction-to-videogame-studies-fall-2011/assignments/game-analysis/MITCMS_300F11_GameAnaGuide.pdf [Accessed 14.10.2020]

McKeand, K. 2016. Assassin's Creed 2017 to focus on emergent gameplay, which Ubi call "the anecdote factory". Article. PCGamesN. Available: https://www.pcgamesn.com/ubisoft-emergent-gameplay-the-anecdote-factory [Accessed 5.10.2020.]

Schell, J., 2020. The Art of Game Design: A Book of Lenses. Boca Raton: CRC Press.

Spelunky (HD remake). 2012. Microsoft Windows [Game]. Mossmouth, LLC.

Sylvester, T. 2013. Designing Games: A Guide to Engineering Experiences. Sebastopol: O'Reilly Media.

Taylor, D. 2001. The Literature Review: A Few Tips On Conducting It. Article. University of Toronto. Available: https://advice.writing.utoronto.ca/types-of-writing/literature-review/ [Accessed 14.10.2020]

Toikka, J. 2019. Developing the Mechanics of Plusminus: Designing for Emergence and Control in a Physics-Based Game. PDF file. Available: http://urn.fi/URN:NBN:fi:aalto-201908254911 [Accessed 28.9.2020.]

Welcome to Kruggsmash Plays Dwarf Fortress!. 2018. Kruggsmash. Video clip. Available at: https://www.youtube.com/watch?v=yq2cQUpqb3A [Accessed 25.9.2020]

Wiltshire, A. 2016. How Spelunky Creates Amazing Unexpected Situations. Article. RockPaperShotgun. Available: https://www.rockpapershotgun.com/2016/03/04/making-of-spelunky/ [Accessed 14.10.2020.]

## LIST OF FIGURES