Spring 2023

Discussion of Game Design and Construction of a Videogame Utilizing PCG, CA, and ABM

Angel Obergh
Bard College

Discussion of Game Design and Construction of a Videogame Utilizing PCG, CA, and ABM

Senior Project Submitted to

The Division of Science, Math, and Computing of Bard College

by

Angel Obergh

Annandale-on-Hudson, New York

May 2023

Dedication:

This dedication goes to everyone that never lost faith in my journey.

***Acknowledgements**:*

Above all, I want to acknowledge God who is a strong presence in my life.

I want to acknowledge my Sproj Advisor Bob who allowed me to work on a personal project that was in my field of interest. I am thankful that I could base my project on what I envision myself doing post-college rather than something that I am not interested in. While building this project was hard, Bob was there every step of the way and helped me when I was lost.

I want to acknowledge my best friend Allan Raposo who despite being miles away from me was always the closest friend I could ask for. For years, you have actively shown me that you care about me and my future. You never gave up on me and always found ways to help me. I hope you know that I am incredibly grateful that you are part of my life and I hope that I have equally given you as much as you have to me. Te quiero manito.

I equally want to acknowledge my mother, father and all my siblings Paloma, Leda, and Alondra because they are huge video game nerds who played video games with me my entire life. My favorite memories with them are playing Kirby and Zelda while making up stories and seeing each other play through the game. My interest in building games in the future is linked to them as they were always a support system and evidence that videogames are more than just games.

I also would like to acknowledge Keith O' Hara. During my freshman year when I didn't think I was cut out to be a Computer Science major, Keith was my professor. He told me that I should stick to it, at least to the end of that semester and if I didn't like it, I could go onto some other major. You had more faith in me than I did to myself, and I grabbed that with me tightly and promised myself to push through even if I didn't think I would ever make it. I thank you for your kind words, quality teaching, and faith that you provided me with.

I am eternally grateful for my friend, mentor, best friend and soulmate Mey Colindres who has taken a huge part of life. I became the luckiest kid at Bard when you became part of my life. Te amo!

Big gratefulness and love to Quincy who always lit me up with his genuineness and love. I thank you for being one of the most caring humans I have ever met.

I want to thank the faculty at Bard college for supporting me and never giving up. Special thanks to Jane Smith who was my first ever college professor. Thanks to Mirelva for helping me navigate the best and worst moments at Bard. Thank you coach Adam and Tyler for preparing me for the real world, as a student and an athlete. Thank you to you and the long list of faculty I can't quite include because it is too long!

*I am innately grateful for my friends, colleagues, and everyone that made my college experience a better one. Big gratitude to the Frisbee gang, my volleyball squad, the Code Blue *Basketball* and BAB! Love y'all!*

Lastly, I would like to acknowledge my beloved OEI Community and every faculty member in this department. Since day until now, they have adamantly worked in making me feel supported, included, and appreciated. They have done far more than this, making my college experience the best experience I could ever ask for. I will be forever indebted to all these individuals and their countless contributions and investments to me. Thank you Claudette, Wailly, Danny, Kim, Kelsey, Jessica, Khan, and many more!

Table of Contents

List of Figures:

Introduction

As Computers have become stronger, faster and more efficient, the creation of games have become way more flexible, unique, and representative of their genres. This is because it became much easier to focus on a game design/gameplay that provided an unique experience to the player rather than using limited resources and space to create them. Thanks to this, concepts like Procedural Content Generation have been used to concentrate interactively rather than conservation of resources. There are now a lot of ways to create, explore and develop levels in video games. Modern Rougelike and mini RPGs games are some genres that have benefited from this openness to just build without limitations. New creative games like *Hades*, *Spelunky*, *Terraria*, and many others provide a fulfilling experience to its audience. For this reason, it has become my interest to explore the process of designing these types of games while simultaneously building one utilizing Procedural Content Generation (PCG), Cellular Automata (CA), and Agent Based Modeling (ABM). The focus of this project is to research the intricacies and styles of these concepts and laterally build a game, Bubu, to explore all the benefits and disadvantages of level designing through random computation in relation to these genres– and others. This project will go over each of these concepts and explain how they will be utilized to build this game. Moreover, I will question how these concepts facilitated (or made harder) the creation of Bubu while explaining how they benefit games in general.

This is a quick overview of all covered topics:

Chapter I:

History of RPGS and Roguelike Games: Quick overview of what RPGs and Rougelike genres are in video games, when they were built and popular games that currently are under this umbrella.

Procedural Content Generation: This goes over what is PCG, how it's typically used and how it can be implemented in games. Moreover, its relationship to roguelike games, how they benefit and what are some challenges you can face when using this algorithm as opposed to designing a game manually.

Cellular Automata: History and overview of what CA is, quick go-through of The Game of Life, and how to build your own CA.

General Overview of ABM: Explains what is it Agent Based Model and the benefits and withdrawals of using this model.

Biggest Takeaways to Level Design a Rougelike Game Using PCG: Analysis of what seems to make PCGs game great games despite being built on randomness.

Chapter II:

Chosen Technology: The tools that I have decided to use to build this game using these concepts. This goes over some of the useful components Unity has as a gaming engine, as well as the benefits of using C# strongly typed language and object oriented.

Game Bubu and Game Design Approach: Goes over some of Unity's components used for the game Bubu and explains how they allowed the creation of PCGs (Especially Unity's Tilemap system.)

Discussion: Consider roguelike games, how indie developers were successful in their creation of these games and how it's linked to the way they work in relation to their audience.

Conclusion Future Work: Goes over the things I learned while building this project and potential future work moving forward.

CHAPTER I: Game Design in RPGs and Rougelike Genres

1.1
History of CRPG and Roguelike Games:

"There are many genres of computer game, but none offer its cocktail of thrilling combat, tactics, strategy, character development, branching storylines, fantastic worlds to explore, and personal enrichment. It just doesn't get any better than quality CRPG" (Barton and Stacks). Computer role-playing games (CRPGs) and roguelike games both came to life in the early days of computer gaming, with their development dating back to the 1970s and 1980s. CRPGs were influenced by tabletop role-playing games like Dungeons & Dragons which were popular at the time while Rougelike games adapted practices efficiently to generate levels. One of the first commercial CRPG is widely considered to be *Akalabeth: World of Doom*, which was created by Richard Garriott in 1979.

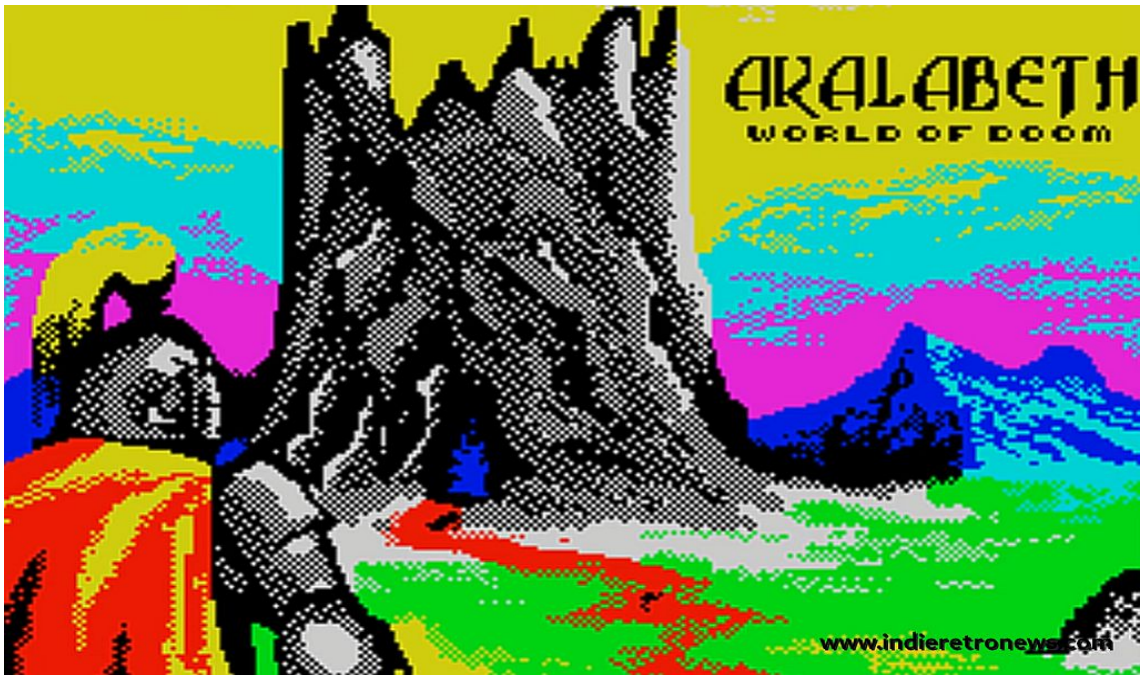

Figure 1: Intro image of the game *Akalabeth.*
[Source: https://www.indieretronews.com/2022/11/akalabeth-world-of-doom-classic-lord.html]

Over time, CRPGs evolved to include more complex narratives, larger game worlds, and a greater emphasis on character development and customization. Some of the most influential CRPGs of the 1990s and 2000s include *Ultima*, *Baldur's Gate*, and *Fallout*, all with unique mechanics and great success.

Roguelike games, on the other hand, originated from the game *Rogue* which was developed in 1980, primarily by Michael Toy and Glenn Wichman. *Rogue* was a dungeon-crawler game that emphasized on procedurally generated levels and permadeath (meaning that when the player's character died, they had to start all over.) *Rogue* spawned a subgenre of games that came to be known as "roguelikes," which were composed of turn-based gameplay, randomly generated levels, and permadeath. Some of the most well-known roguelike games include *NetHack*, *Angband*, and *Dungeon Crawl Stone Soup*.

Figure 2: Image of a Nethack generated level. [Source https://snapcraft.io/nethack]

As described in the article “History of the Roguelike, from Rogue to Hades” written by Coleman Gailloreto: “Roguelikes present a random, hostile world in which time is rarely on the player’s side. In general, wasted turns will come back to haunt the player, by attracting monsters, depleting food or just from plain old opportunity cost.” A huge factor in this genre is the generated environment and its ability to impede the player from succeeding which, of course, is obtained through PCG algorithms.

Both CRPGs and roguelike games have continued to evolve and adapt to modern gaming conventions. CRPGs like *Divinity: Original Sin and Pillars of Eternity* have robustly shown their success, while roguelike-inspired games like *Spelunky* and *Dead Cells* have gained popularity for their fast-paced action and challenging gameplay. It isn’t hard to see why these gaming genres are so popular and unique. They provide limitless possibilities, worlds to explore, great challenges, self-improvement, and even a personal connection to your own heroes. As it’s more fascinating, some of these games and their worlds/levels are built through randomness and few rules. How can we accomplish this? For the game I will be building, Bubu, I will go over how using these concepts we can generate cool levels to play in as well as recognize the distinct ways of building a game without them.

1.2
Procedural Content Generation:

The book "*Procedural Content Generation for Games*" provides a well put definition for how we can think of PCG: "Procedural content generation is the automatic creation of digital assets for games, simulations or movies based on predefined algorithms and patterns that require a minimal user input" (Shaker et al. 7). This means that you can computationally generate content using patterns, randomness and/or rules. As previously mentioned, PCG has been exhausted in roguelike games due its ability to provide endless levels which encapsulates how roguelike games are built. Even though this concept was used to conserve memory and allow unlimited gameplay, it is also considered to be attached to level design as it can provide engaging systems and levels.

One of the benefits of PCG is that it allows developers to create content that is unique and varied for each playthrough which enhances the replayability of a game. Since the content is randomly generated, the player rarely experiences the same level twice. This can be particularly effective in open-world games, where the environment is a critical component of the player's experience. PCG also allows the creation of content more efficiently. Traditionally, designers would have to spend significant amounts of time and resources creating content by hand, but with PCG, much of the work can be automated. This can result in a significant reduction in development time and costs. This is especially true and actively utilized for independent game developers (Indie devs.) Indie developers are very small teams, or even solo developers that create games with their own ideas, story line and mechanics. The rise of indie developers is what has allowed the flourishing of recent roguelike games like *Spelunky*. The creation of a game

can’t be detached from the way it's created. Big game dev companies have the resources and time to build a great game. Indie developers need to be astute, and self-driven as they have way less resources and time for building their games. Later in this paper, I will point out how indie devs are so successful in PCGs despite working in small teams.

1.2.1
Procedural Content Generation Vs. Handmade Game Design:

However, it is undeniable that uniqueness and intentionality of a level design can be lost due to this way of generating content, therefore, running into the issues of repetitiveness or even poor-crafted levels. For example, when there isn’t a possible path to find the exit hence making it impossible to complete a level. Or even worse, enemies are not spawn in reachable areas and you can never experience the satisfying combat experience. This is why PCG despite being randomly generated still requires user input and some manual instructions to assimilate to a more controlled environment. Moreover, it equally requires tons of trial and error to make sure that all the levels that are being generated are playable.

If we consider a hand crafted game as *Hollow Knight*, you will soon notice these minor but yet, important qualities of a game that are lost when randomly generating content. *Hollow Knight*, a game built by Team Cherry, was accurately designed, providing an incredible story line and mechanics. From sounds, to skills, to maps, to bosses, everything is interconnected in a way that outstands a player’s expectation. A game this carefully designed is arguably what the audience wants. While this game follows story mode and doesn’t produce new content as is achieved through PCGs, it’s important to note that it isn’t necessary. In fact, it’s probably best

that the game has a rigorous story mode. This is also because *Hollow Knight* is more commonly embraced as part of the Metroidvania genre. Metroidvania is a sub-genre term for action-adventure games where the level design is guided by non-linearity and utility-gated exploration and progression: “These games usually feature a large interconnected world map the player can explore, although parts of the world will be inaccessible to the player until they acquire special items, tools, weapons, abilities, or knowledge within the game” (“Metroidvania”).This creates a sense of uncertainty and dangerousness. While the game is progressive and eventually you will beat the boss and win the game, the player is always altered by knowing that there are secret locations, additional skills and potential milestones. The player has to then make decisions such as “Do I explore as much as I can?”, “Do I speedrun through the game?”, “Should I ignore these secret spots?” “Should I play it safe and not risk wasting my resources?” While cumbersome to some, these are some of the qualities that many players want to face when playing a video game. It is therefore important to consider when designing a game what exactly you want to achieve and how it connects to the genre you’re interested in pursuing.

1.2.2
PCG and Roguelike Genre in Spelunky

A perfect example that utilizes PCG is the game *Spelunky* by Dereck Yiu. In *Spelunky*, players take on the role of an adventurer who must explore a series of underground caves filled with dangers such as traps, enemies, and environmental hazards. The ultimate goal is to reach the end of the cave and retrieve a treasure known as the "Golden Idol."

Figure 3: Gameplay of Spelunky [Source: https://store.steampowered.com/app/239350/Spelunky/]

PCG in Spelunky uses a set of rules to generate levels that are both challenging and balanced. It starts off by dividing the level into a grid of tiles, each representing a small section of the level. The PCG system then generates the level by placing tiles in the grid based on a set of rules. These rules take into account various factors, such as enemy placement, item distribution, and level layout, to ensure that the generated level is balanced and challenging.

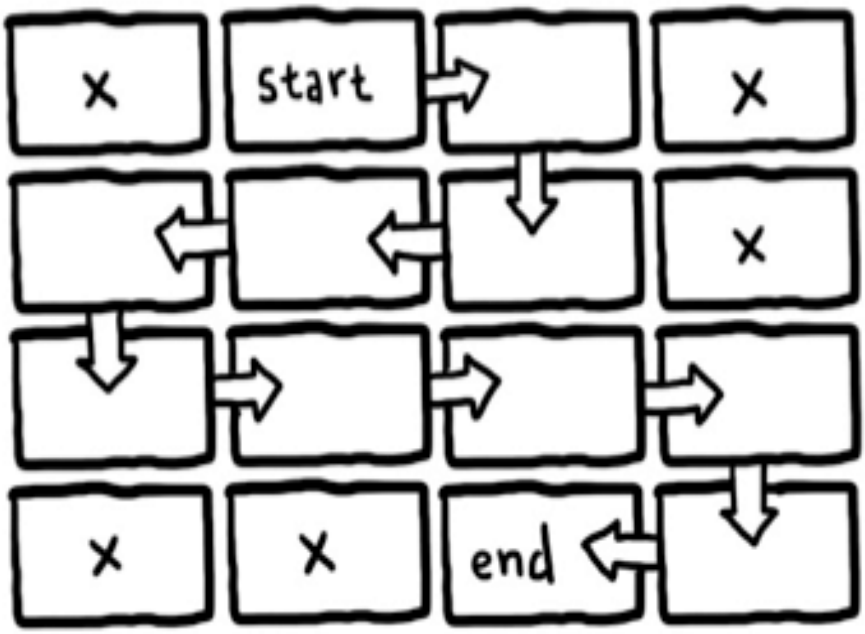

Figure 4: Graph that shows the generating of a continuous path from start to finish in Spelunky
[Source: https://www.researchgate.net/figure/Level-generation-in-Spelunky-Adapted-from-10_fig3_309279824]

One of the key benefits of using PCG in *Spelunky* is that it creates an infinite variety of levels. This means that players can play the game multiple times and still encounter new and unique levels. This also allows the game to remain fresh and challenging even after multiple playthroughs. Because the levels are generated randomly, players never know what to expect, creating a sense of unpredictability and discovery. The levels are also designed to be challenging, with the PCG system generating enemies, traps, and other obstacles in strategic locations to create a difficult but fair gameplay experience. However, a more distinct characteristic of *Spelunky* is just how well designed the game is through randomness. Most of the players don't realize that this game is randomly generated since it really considers its parts as a whole to produce a fun challenging level. Overall, the PCG system in *Spelunky* is a key aspect of the

game's design, replayability, and challenging gameplay. It is a great example of how PCG can be used effectively in game design to create unique and engaging gameplay experiences.

“With Spelunky, you are never learning a 'piece' of music.... It's still a game about repetition and learning, but what you are learning is the overall composition, understanding the overall system and how it works, and becoming fluent in that" (Terrell). The main takeaway from this style of level design is that the game forces the player to understand how the game itself works. Every level, every playthrough, will be composed of random factors that will work as a whole to challenge the player. This works so well in RPGs and Rougelike games precisely because it truly puts your skills into question. It provides a more satisfying experience because going through a level isn’t just “muscle memory.” thing, it requires you to weigh each decision out and take the best course of action, whether you are right or wrong.

However, a common issue among roguelikes is the inability to provide a good reward system to the players to balance out how challenging the playthrough can be. Given that these types of games use permadeath, it is often dissuading to keep playing after a few runs because any potential progress is never preserved. In the discussion in Yiu’s book, *Spelunky*, this is pointed out as losing due to the game and not your lack of skill can be detrimental: “But from this game it seems you suck at difficulty.” While he and I agreed that a player’s death should always be attributed to their own mistakes, we disagreed on what actually constituted a player’s mistake. In his mind, a death that resulted from his own self-described lack of ‘natural skills’ at video games—including good dexterity and reflexes—was the game’s fault and not his. “Feeling cheated and insulted by a game is not fun,” he exclaimed. “Unless a person is brainwashed or mentally handicapped.” When I die in a game, it’s frustrating but it makes me want to keep

trying and improve. To him, it was insulting" (Yiu 55). Although there are different ways to consider solving this issue, it's important to note that there is no easy fix, especially because losing all your progress after playing means restarting from all over. Finding the right balance between challenge and skills can be very difficult as you're dealing at the end with randomness.

1.3
Cellular Automata

Cellular Automata is a discrete model by a mathematical computation that generates complex systems composed of simpler rules. CA is a grid of cells, where each cell has a state– this can be alive or dead, on and off, etc. Each cell has a neighbor and based on some rules, the interaction between the cells and its neighbor will generate the outcome of this system. This output can be deterministic or stochastic. A cellular state can be described as: f(Neighbors, state). This process happens iteratively for each generation.

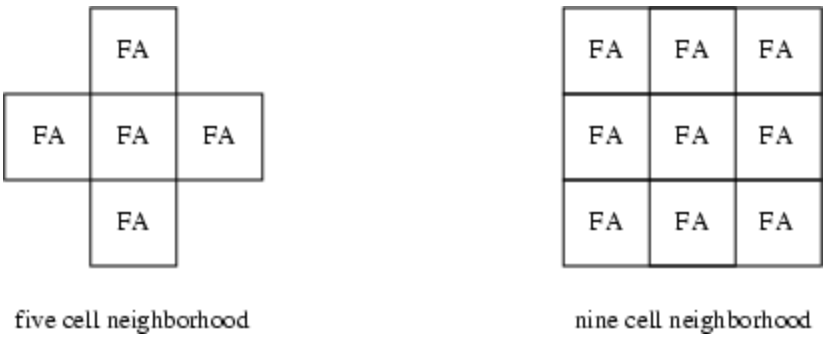

Figure 5: Common neighborhood templates that are used to that are un in a 2 matrix to for CA.This image shows that you can consider in a given cell its 4 adjacent cells vertically and horizontally, or you can consider all its 8 adjacent cells, vertically, horizontally, and diagonally.
[Source: https://theory.org/complexity/cdpt/html/node4.html]

1.3.1
Example, Game of Life:

One of the prominent examples using CA is the *Game of Life (GOL.)* The Game of Life is a cellular automaton created by mathematician John Conway in 1970. It is a zero-player game, meaning that its evolution is determined solely by its initial state, and does not require any further input. The game consists of a grid of cells, where each cell can be in one of two states: alive or dead. The state of a cell at any given time depends on the states of its eight neighboring cells (horizontally, vertically, and diagonally adjacent). The rules for determining the next state of each cell in GOL are as follows:

- Any live cell with fewer than two live neighbors dies, as if by loneliness
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by crowding.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

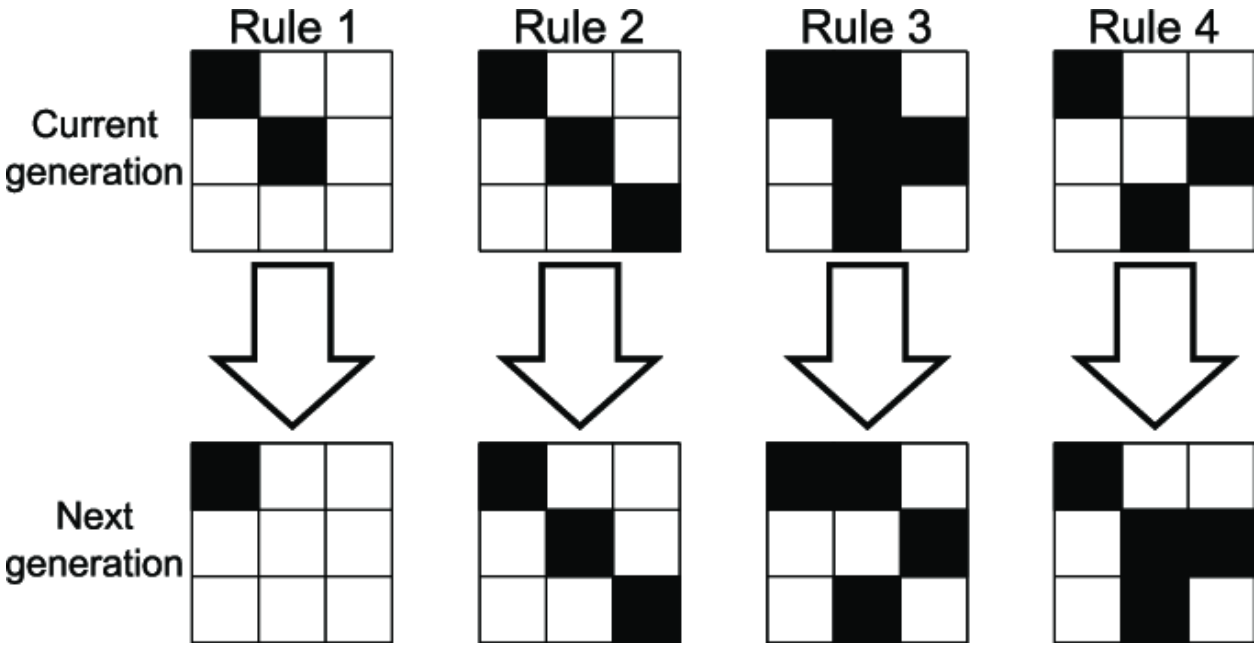

Figure 6: Applying rules to a set of cells in the Game of Life. [Source: https://www.researchgate.net/figure/Rules-of-Conways-Game-of-Life_fig5_339605473]

These rules are applied simultaneously to all cells in the grid, and the process is repeated for each generation, creating a new grid based on the previous one. The Game of Life has been used to study complex systems, pattern formation, and artificial life, and has also been used for entertainment purposes as a simple example of its behavior in a complex system.

To create a simple CA algorithm, we can create a table/grid (a matrix) where each i,j is a cell. You can start off by going one generation to set up the grid, and giving a state to each cell given a probability– typically 45%. You will have a neighbor check function that will check neighbors of the current cell you are at. Using this function, we can then create a set of rules that will dictate each cell's outcome. This can be up to preference or the system you wish to create. Lastly, you will want to have another function that smooths out your system. You can repeat this process for multiple generations to build something (in the case of this project, an island.)

For the purposes of this project and the creation of the game, we can consider CA as an enforcement tool to produce rich results in PCG through well thought out patterns. Utilizing these CA rules, we will be able to form the island after a set of generations in the grid. The binary state– 1s and 0s, will represent water and ground for the game Bubu.

1.3.2
Necessary aspects for Level Design using CA-PCG:

The generation of a game using CA-PCG requires more than just creating an island. Repeatedly generating content with a set of rules means that enough variation has to exist in order for the game to show sophistication. The addition of items, enemies, obstacles, and missions can be considered to produce new content each time. Additionally, the game still needs to account for a goal. This relates to what it is that the player needs to solve in order to win the game. What challenges and obstacles will arise to prevent the player from winning.

A more important aspect that can radically affect the player's game experience is a functional game. Even though this is rather obvious, it can be a real problem when playing with randomness. The same excitement of a random world to explore, is the same fear of an unstructured world that can't be navigated. Especially for creating a map utilizing CA-PCG, it's important to manually set the grid so that it connects its path, at least those that are necessary to traverse through.

Thirdly, it's important to consider the degree of difficulty at any given point in time during this game. A game that's too difficult or too simple it's not joyful to play. If using CA-PCG, are you generating small corridors with a lot of enemies that are impossible to evade?

Are you placing really strong enemies together in the same locations? These are some examples that can gravely affect the generated level and thus the player's experience.

1.4
General Overview of Agent Based Modeling (ABM):

Agent Based Model is a system modeled as a collection of autonomous decision making entities called agents. Each agent has the ability to assess the situation and respond based on a set of rules. It can then be compared to their interactions between agents. This is often used to emulate real-world systems. Moreover, agents can evolve / improve over time, allowing them to demonstrate unique behaviors. For this game's project, you can start considering Agents as enemies AI.

1.4.1
Benefits of using ABM:

The article, "Methods and techniques for simulating human systems," states many benefits of utilizing ABM: Firstly, it argues that ABM captures emergent phenomena, i.e, the interaction between individual entities. By definition, they cannot be reduced to the system's parts: the whole is more than the sum of its parts because of the interactions between the parts. We want to consider ABM for potential emergent phenomena when:

- Individual behavior is nonlinear and can be characterized by thresholds
- Individual behavior exhibits memory, path-dependence.
- Agent interactions are heterogenous and can generate network effects

Additionally, ABM provides a natural description of a system ABM can be the most natural medium for describing and simulating a system composed of behavioral entities. Especially when there's plenty of information about the agent, it is easy to build ABMs that simulate a more realistic model. This is usually desired for naturality when:

- Individual behavior is complex
- Activities are more a natural way of describing the system than processes
- Stochasticity applies to the agents' behavior. Sources of randomness are applied to the right places as opposed to a noise term added more or less arbitrarily on an aggregate equation

ABM is flexible as it can easily add more agent-based models when necessary. Moreover, it proves a natural framework for tuning the complexity of agents (whether that is behavior, degree of rationality, ability to learn and evolve, and rules of interactions.) Moreover, it can be easily aggregated sub-agent, single agents with their own levels of behaviors/rules, providing way more diversity for the construction of Agent-based models (Bonabeau).

1.4.2
Downside of using ABM:

There are downsides of using an Agent Based Model that we need to consider:

- It's specific-based. It can only serve the purpose it was assigned, thus any general-purpose model cannot work.
- Computationally intensive: ABM simulations can be computationally intensive, especially when the number of agents and the complexity of their interactions increase.

This means that ABM simulations can be time-consuming and require high-performance computing resources.

- Difficult to validate: ABM models can be difficult to validate because they often involve many interacting components, and it can be challenging to ensure that the model accurately reflects the real-world system it is meant to simulate.
- Sensitivity to initial conditions: ABM models are sensitive to initial conditions, meaning that small changes in the starting conditions of the simulation can lead to significantly different outcomes. This can make it challenging to compare different simulations and draw meaningful conclusions.
- Lack of standardization: There is currently no standardized methodology for developing and validating ABM models, which can make it difficult for researchers to compare and replicate results across studies.
- Assumption-heavy: ABM models require many assumptions about the behavior and interactions of agents, and the accuracy of these assumptions can be difficult to verify. This can lead to uncertainty and potential biases in the simulation results.

On the bright side, utilizing ABM for building a game eliminates some of these downsides. ABM concept will be used for the creation of smart enemies AIs that can challenge the player in its journey. Therefore, all enemies will have a purpose. For creating this ABM, we can consider each agent's position to the player and make decisions based on this (which it's information we know.) For example, an agent enemy can accelerate when closer in distance to the player to put pressure on the player. Moreover, the Agent based models can interact with other enemies, or their own class attributes. An enemy AI whose health is low can be

programmed to run away from the user. On the other side, an enemy AI that is closer to other enemies AIs can become more aggressive and attack together. Since all the parameters presented for creating these ABMs can be easily calculated, it can be determined at any point in the game what action should be taken.

1.4.3
Biggest Takeaways to Level Design a Rougelike Game Using PCG:

Considering the nature of Rougelikes games, PCG, and level design, I want to draw out some important key points that can help for building the game Bubu based on the research I have gathered. I think it's important to stay within the genre (and by this I mean still creating a game that is based on procedurally generated levels, permadeath, and challenging levels.)

1. **The game has a good system to compensate for permadeath:**

   In the game *Rogue Legacy*, when you die you don't feel like you have totally wasted your time. This is because the money collected from that run will be saved and can be used to buy upgrades that can keep you alive for longer. Similarly in the game *Isaac*, even though permadeath is very frequent, you still have the chance to unlock new characters, items, and/or abilities which don't really decrease the difficulty of the game; however, the amount of unlocked stuff correlates to how much you've progressed in the game, allowing the player to feel rewarded for the time they invest playing.

2. **There still exists good amount of content to entertain the player, it's not just mechanics:**

   One of probably the biggest downsides of building roguelike games is that they can discard the story behind the game itself because there is more emphasis on mechanics and gameplay. As discussed earlier, a well designed game like *Hollow Knight* will stay true to its colors and will provide a satisfying story that the player can follow. For roguelike games, a great illustration is *Hades* since this game uses an astute system to provide content and a compelling story while also incentivizing the players to play more. Even though *Hades* has the permadeath component, the player can be excited to know that new content will be unlocked after the run throughs, keeping the audience entertained even though they keep losing at the game.

3. **The randomly generated levels have enough variety and don't show repetitive boring levels:**

Roguelike games take the advantage of their genre when there is a focus on never lasting gameplay. Good randomly generated levels will not make the player feel like they are playing the same level they just played a few minutes ago. And to achieve this, we can consider randomness not only for the generation of levels, but also to create unexpected events. In *Spelunky,* the breakable objects that players can pick up and use have a 0.05% chance of having an enemy inside. While subtle, an instance where a player is trying to get gold out of a simple breakable object becomes a questionable decision that makes the player consider its current situation before breaking a jar.

CHAPTER II: Creation of Bubu

Author note: All images presented from this chapter and on are from their author (Angel Obergh)

2.1
Chosen Technology:

As it's no surprise, building a game it's hard. This is why I have decided to use Unity as the gaming engine to build Bubu. Unity is a powerful software that allows the creation of 2D, 3D, and rendering animations. Unity has a lot of built in components that will facilitate the creation of CA and PCG for this game, as well as minor tasks like detecting collision, enemies, sound track. Along with Unity, Visual Studio Code is the IDE that I will be using to code the scripts. VS Code is typically used in Unity to code.

Unity has a built-in physics engine that makes it easy to simulate realistic movements and interactions between objects in the game world. This is important for Bubu as it allows to create a believable and immersive game world where objects interact with each other in a natural way. Unity provides a wide range of tools for creating and animating 2D graphics. This includes a powerful sprite editor which allows the creation of complex animations and spritesheets for characters, objects, and backgrounds. Unity also supports the use of 2D physics, which is especially useful for top-down view games that rely on accurate collision detection and response.

Furthermore, Unity provides a range of scripting languages, including C# and JavaScript, which make it easy to create game logic and mechanics. This is useful to implement ABM, AI behaviors, inventory systems, and character abilities, in a straightforward and organized manner. In addition to this, Unity provides a range of tools for optimizing game performance, which is especially important for top-down view games that may involve large numbers of objects on

screen at once. Some of these tools that I am hoping to include are: Object pooling, occlusion cooling, LOD systems, and Unity Burst Compiler.

C# is an object oriented language which has been beneficial for this game to create enemies classes, a player class, and agent based models. Moreover, C# is a strongly typed language which improved the readability of my code.

2.1.2
Plastic SCM | Version Control:

Because Unity is the software I am using to build this game, I have decided to work with the version control system Plastic SCM. Plastic SCM is a software that assists in managing codebase, including version control, branching and merging, code review, and issue tracking. It supports a range of programming languages and integrates with popular development tools such as Visual Studio, Eclipse, and Unity. While this is typical for any big project, Plastic SCM has been a terrific tool to revert back to versions of my program. This is because sometimes big issues like infinite loops or crashes happened while building this project and this made it super easy to simply go back to a working version.

2.2
Game Bubu and Game Design Approach:

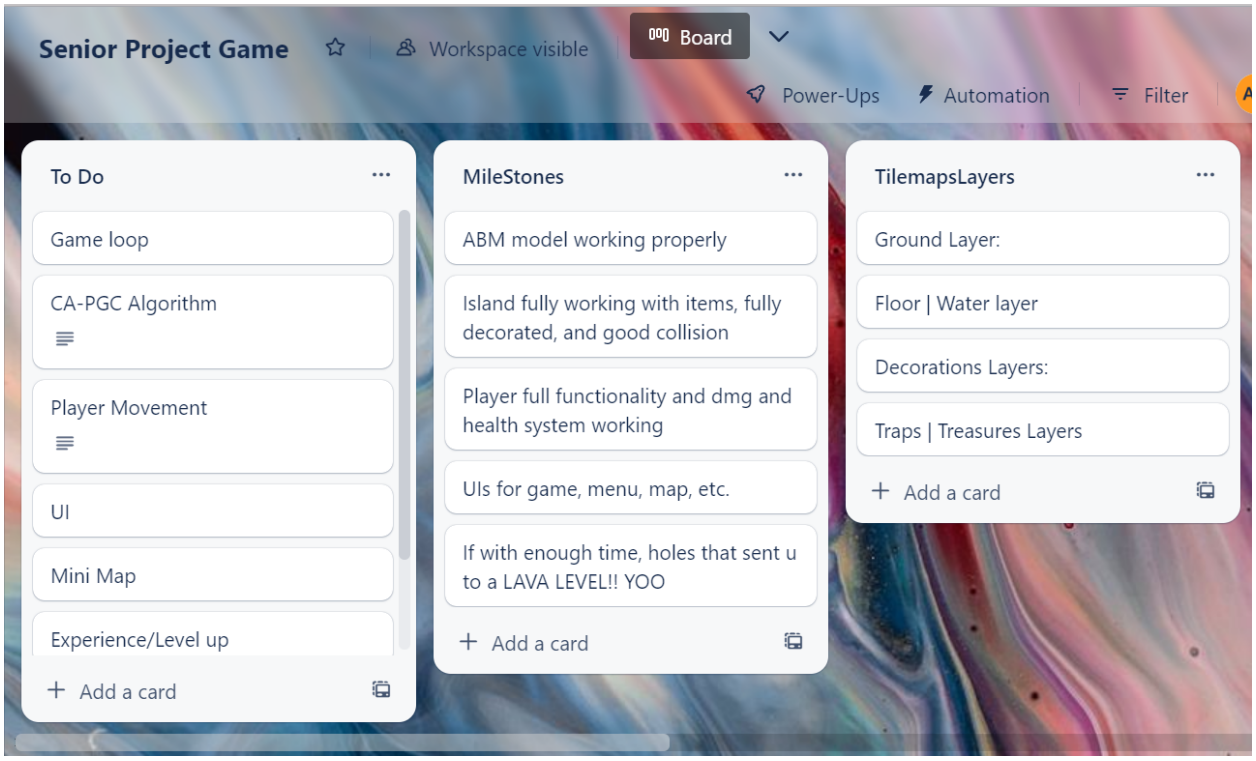

Figure 7: Screenshot of Trello page with some of the goals and to do list of things necessary to complete Bubu.

The game concept directly ties to how we are utilizing PCG and CA. Bubu is a survival Roguelike game where the player wakes up in the middle of nowhere and with little resources, a sword, some arrows, and a pistol with limited bullets, the player needs to fight monsters, evade traps, and survive. In the midst of the unknown, the player can find items and add ups to become stronger. Along in your journey, Spirit Ghosts will be situated at random locations, willingly

ready to help the player and help them find the correct path. The player needs to use this information to win the game.

PCG and CA work cohesively with this survival theme as this becomes a new experience each for each run through of the game. With enough variety, the player can spend a lot of time exploring and figuring out the new paths/missions to win the game. The creation of a long unexplored map will constantly force the player to choose new areas, figure out new ways to escape, and think about the limited resources they have. Because one of the main focuses of this game was to bring a sense of exploration (Adventure rougelike), I have considered adding UI features such as a Minimap and a Boxchat to communicate.

Figure 8: Screenshot of the game Bubu in Unity showing the basic UIs of the game with the box text, mini map, and hearts containers.

2.2.2
Implementation of Cellular Automata for the Creation of an Island

For Bubu, the MapGenerator script deals with the construction of the island. The function GenerateMap() takes in a grid sizes WIDTH and HEIGHT. It starts off by iterating once through the map, giving a state to the cell, 1 or 0 with a 45% probability (1 corresponds to an area that's inaccessible and 0 corresponds to an area that's land.) After fully iterating through the grid, it "smoothes" out the grid given the CA rules. The function int countWalls(x,y) takes in a point in the graph and returns how many neighbors it has given our binary system. If this function returns 4 or more walls, then our smoothing function will convert this cell into water, and if it's less than 4 it will convert this cell into land. This is repeated as many times as we wish to iterate through the map to refine edges and areas that are too unnatural.

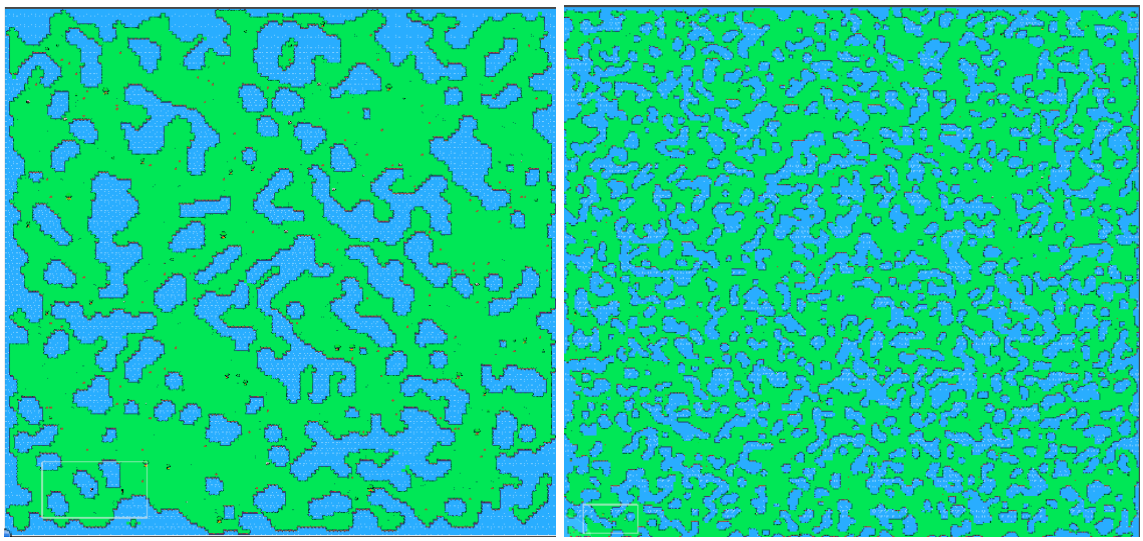

Figure 9: Screenshot of the generated islands in Bubu. Cellular Automata is used to create islands using different parameters, first image uses Width and Height of 150x150, with 7 proceeding generations Second image uses Width and Height of 300x300, with 4 proceeding generations.

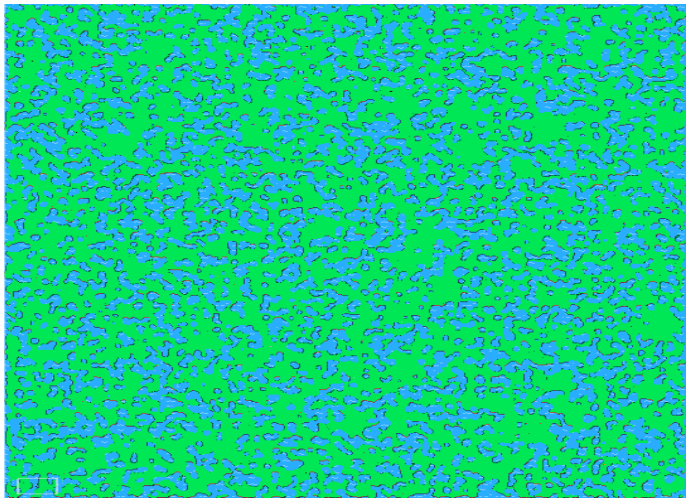

Figure 9.1: 600x600 with 6 smoothing.

After a lot of trial and error, the island started to take shape. I realized that in order to keep a more natural island look, it was necessary for the width and height to be in the margins of 100-300, with 6-10 generations. Drastically increasing the WIDTH and HEIGHT or not adding many smoothing generations created a lot of these unstructured islands where water and land were almost the same.

I took inspiration from building around limitations. As the article "How to effectively use procedural generation in games" argues: "The next time you think of a question like 'What is the best way to generate a system of caves?', maybe think, 'What is a way that I already know to generate a system of caves, why are those caves unsatisfying to me, and what can be changed about everything but the caves themselves to make them satisfying?" (Kazemi). While this is an island and not a cave, the same methodological process can take place to solve some of the issues. Why expect the perfect randomly generated island when instead I can work around its imperfections? I have decided to do this because it feels as if it will allow a more explorative environment. Additionally, this will allow me to think more critically about mechanics that can

enhance the gameplay while also taking in account how to solve for those “unsatisfying” structures. As it’s true that using PCGs brings a lot of challenges, like not having transversable paths, or making sure that items are located in the ground and not water.

After having the map generated, I focused on building other functions to create content. Thanks to the Tilemap component in Unity has layers, it’s easier to have a layer for each respective asset that’s added to the game. This is very useful as it allows you to keep a ground layer on the very bottom, and a trap layer for instance all the way to the top. I created a helper function, FindRandomFloorTile() of type Vector. Given a number of trials, it randomly picks a location in the map and then checks to see if it’s available, if not it keeps checking based on the number of trials. By available, it means that it checks if the randomly chosen tile is a floor tile and not a water tile. If that’s the case, then stores the location and returns it when it’s needed.

CA-PCG in Unity

With the basic idea of how each algorithm works, we can now consider the game Bubu as a whole. Cellular Automata and PCG will be directly connected to each other while Agent Based Modeling will indirectly use this information to create enemies AI. For the instance of Bubu, CA-PCG will simply generate an island with enough variation that is fun to explore. I have decided to do this by using the component Tilemaps from Unity. Tilemaps is a component that you can utilize to facilitate the tilesets and organize which tiles will have collisions and which tiles will not. More in depth, Unity provides a really beneficial component within Tilemaps called Autotile. Autotile allows you to manually set up the rules and behavior of each tile so that

it connects the right tiles to each other. In the MapGenerator script, we can create a public component of Tile Map and draw out the autotile. The Autotile then sets the autotile based on our behavioral rules of CA, and because it automatically creates corresponding tiles adjacent to one another, there's no need to worry about tiles in their incorrect position.

Figure 10: Unity's component autotile in Tilemaps. This creates a single set that contains this tileset and the relationship of each tile with its whole. When our Cellular Automata calls out this autotile (that's a tile you can set on the grid) it always considers its adjacent tile and adds the correct tile at any given point. This happens even after going through different generations Smoothing for our function.

By doing so, we can now create tile map layers that can account for everything necessary to generate a full island. It starts off with the layer of the autotile which simply sets the land tiles and the water tiles in their corresponding position/form. It then uses another tilemap that has all the obstacles like trees, rocks, bushes, etc. and adds new objects depending on the first layout for the island. That is, after creating the full island, there's now another iteration through the map that checks if there's space available to add obstacles. This means that it will only place an obstacle over the tiles that are considered floor. This prevents issues like having trees or bushes on water.

After having a generated map, we can start adding enemies. The location of the enemies will be evaluated on the map based on free floor tiles and the location of the player. The goal is to have a dynamic list of enemies that will challenge the player while exploring the island. Now, ABM behavior will be dependent on specific aspects. Primarily, each agent will have the ability to detect the player and follow them within a given distance. As the player loses health, the enemies will become frenetic, adding difficulty and pressure when the player is low. The enemies will also take in account the amount of bullets and arrows that the player has. The lower the amount is, the more likely they will become in getting closer to the player to attack. Also the enemies will consider their surroundings, how many other enemies are around, their own health, and the time elapsed.

This implementation will hopefully create an intense survival game where the player truly needs to be mindful of their time, location, and resources at all times while playing the game. With enough variety, the player can make decisions as to whether to hide, search for

resources, or just go on in a full fight with the enemies. This will be the core of the game. More components can be considered after to create a more sophisticated game.

Discussion:

There's a lot of styles and different ways to create a fun interactive game. Finding the right balance between luck and skill, content and uniqueness, and even rewards systems and repetitive loss makes up for a lot of prominent games. When using PCG and CA as the main components to build a game like this, there needs to be accountability for then dealing with everything else that will make up for the game as a whole. It's imperative to think critically about how you will use your other systems to create a good game.

I also would like to bring up the conversation of Indie developers that take their time to build awesome games with little resources in these genres. As a game enjoyer, over the past few years I have come to see myself playing more and more indie games rather than AAA games. Why is this? A huge reason is probably because I am a nostalgic guy who loves pixel art like *Final Fantasy* and *Zelda*. Indie devs bring a lot of resemblance to these types of games. But more for a more technical answer, I truly like the carefulness and intentionality that's put in each game. I don't want to take away from AAA organizations this credit because they spent years in their big games and of course that means probably more perfection. However, I think I am speaking towards a different kind of thoughtfulness. For indie devs to sustain themselves, they often do marketing and offer their audience to input ideas for their on-going games. While the intention here is for indie devs to be able to work full time in their games, an invisible contract

between the audience and the author is built, even some sense of trust. If a game concept is absolutely rejected by the audience, there is very little chance that the developer will adapt it to the game. Therefore, you can think of these games almost as if they were built for their audience.

Conclusion in Future Work:

While I was unable to complete Bubu, I learned an incredible amount about game design in relation to the algorithms used to build it. Don't underestimate the job of a game designer as even the smallest miscalculation can fatally destroy a video game. If you look at a game like *Celeste* whose mechanics are built completely dependent on the game physics, even a slight change to gravity will prevent the player from reaching the platforms. Games that are accurate are satisfying because nothing but perfection is expected by the player and knowing that you can pass the level is proof that you beat the designer. When randomness takes part in a game, conversely, a whole new scenario is created. It's not just you against the creator; it's also you against logic. It's you against the game, almost as if the level itself is considered an enemy. At the same time, the designer is responsible for yet creating another living creature in their game that's neat, beatable, and joyful to play against it. So what's at stake? Considering science itself as art is by far the biggest lesson I learned from PCGs, roguelike games, and this project. Despite not finishing my project, I realized that it takes more than just knowing how to use PCG in order to build a videogame. Additionally, I have come to value more UI's systems and less direct parts of a videogame. As a creator, I was initially interested in having great mechanics that will produce quality gameplay. But I was wrong. Quality gameplay comes from all the parts as a

whole. A fun compelling story, an inventory with all your items, a minimap UI, present health system, particle systems, and the list goes on. To say the least, PCG for games is not enough.

Overall, I am glad that I could do this research and learn more about level design. I had the opportunity to learn how to use a gaming engine, Unity, and how it works. I also learned a lot about organization and just how important it is for building a game. With this set of skills, I am planning to finish Bubu and start my journey in becoming a Game Programmer. I hope that by finishing this game I will be able to create more fun and interestings games in the genres of roguelikes and RPGs.

Works Cited

Barton, Matt, and Shane Stacks. *Dungeons and Desktops: The History of Computer Role-Playing Games 2e*. CRC Press, 2019.

Blomberg, Johan, and Rasmus Jemth. *An Exploration of Procedural Content Generation for Top-Down Level Design*. Chalmers University of Technology, 2018, https://publications.lib.chalmers.se/records/fulltext/256132/256132.pdf.

Bonabeau, Eric. "Agent-Based Modeling: Methods and Techniques for Simulating Human Systems." *Proceedings of the National Academy of Sciences*, vol. 99, no. suppl_3, May 2002, pp. 7280–87. *DOI.org (Crossref)*, https://doi.org/10.1073/pnas.082080899.

"Exploring Roguelike Games Book by John Harris (z-Lib.Org)." *Studylib.Net*, https://studylib.net/doc/26051747/exploring-roguelike-games-book-by-john-harris--z-lib.org-. Accessed 3 May 2023.

Gailloreto, Coleman. "History of the Roguelike, from Rogue to Hades." *ScreenRant*, 11 Dec. 2020, https://screenrant.com/roguelike-definition-games-rogue-hades-roguelite-dungeon-crawler/.

Harris, John. *Exploring Roguelike Games*. CRC Press, 2020.

---. *Exploring Roguelike Games*. CRC Press, 2020.

Kazemi, Darius. "How to Effectively Use Procedural Generation in Games." *Game Developer*, 10 Apr. 2019, https://www.gamedeveloper.com/design/how-to-effectively-use-procedural-generation-in-games.

LoguidiceBloggerMay 05, Bill and 2009. "The History of Rogue: Have @ You, You Deadly Zs." *Game Developer*, 5 May 2009, https://www.gamedeveloper.com/design/the-history-of-rogue-have-you-you-deadly-zs.

Macedo, Yuri Pessoa Avelar. *An Integrated Planning and Cellular Automata Based Procedural Game Level Generator*. Nov. 2018. *repositorio.ufmg.br*, https://repositorio.ufmg.br/handle/1843/36255.

"Metroidvania." *Wikipedia*, 25 Apr. 2023. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Metroidvania&oldid=1151704310.

Spelunky - Derek Yu.Pdf - Free Download PDF. https://kupdf.net/download/spelunky-derek-yupdf_59603950dc0d60dd012be30c_pdf. Accessed 3 May 2023.

Terrell, Richard. "A Spelunky Game Design Analysis - Pt. 2." *Game Developer*, 15 Nov. 2012, https://www.gamedeveloper.com/design/a-spelunky-game-design-analysis---pt-2.