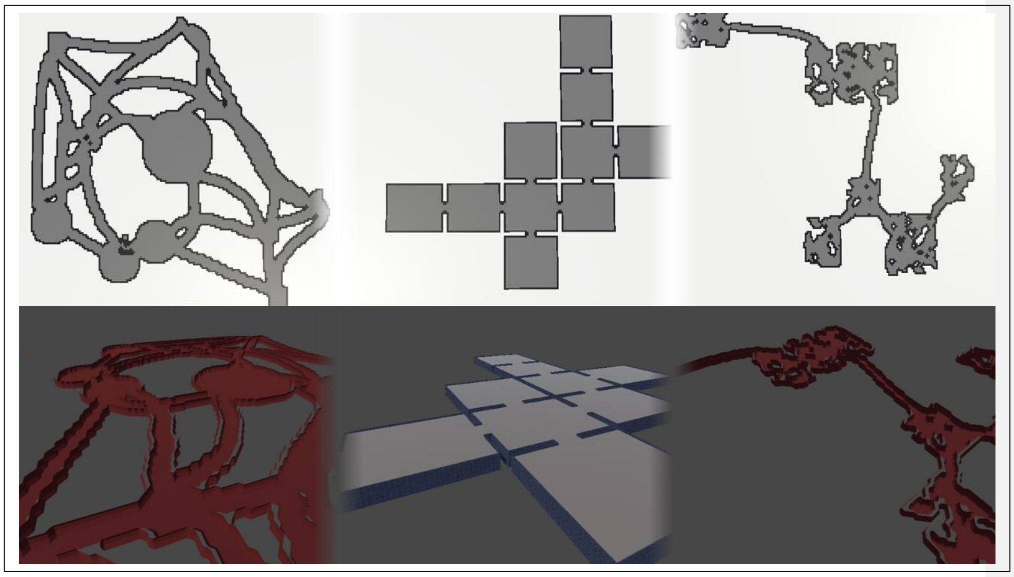
2021/22

Comprehensive Creative Technologies Project: How Do We Develop a Tool for Procedurally Generate a Functional 3-Dimensional Dungeons.

Alessandro Bufalino
alessandro2.bufalino@live.uwe.ac.uk
Supervisor: Joshua Hompstead

Department of Computer Science and Creative Technology
University of the West of England
Coldharbour Lane
Bristol BS16 1QY



Abstract

Procedural generation is a technique widely used in the game industry to create game environments and maps. However, using the existing tools and implementing the algorithms can be difficult for users who are not that technically proficient. This report presents a downloadable asset pack for the Unity game engine that introduces a custom editor, in which the user can pick between the offered generational algorithms based on the user's needs. The user-friendly custom editor allows the user to customize the parameters of each algorithm to their liking with the additional generation of more rooms and other paths. The asset pack also can act as a library so the user does not have to use the built-in custom editor but can use the different algorithms in other scopes.

Keywords: Procedural Generation Algorithms, Unity Asset, Heuristic evaluation, Unity Editor.

Brief biography

Since the start of my journey in computer science, I have always found the automation of tasks to be a fascinating area of programming, which has sparked my interest in procedural generation and artificial intelligence. This project has allowed me to create an asset pack that I see myself using in my future projects, as it provides quick access to various algorithms and more.

Website Portfolio: https://alessandrobufalino3115.github.io/

How to access the project

The project code base is divided into two github repos:

1) Repo for the developlment of the asset pack -- https://github.com/AlessandroBufalino3115/Dissertation_PGC
2) The asset pack to install into unity via the package installer -- https://github.com/AlessandroBufalino3115/Dungeon-Forge

For the Wiki -- https://github.com/AlessandroBufalino3115/Dungeon-Forge/wiki

Project Video URL -- https://youtu.be/7DBc7ZwA-ss

1. Introduction

In the constantly evolving landscape of video game development, procedural content generation (PCG) has emerged as a powerful tool to create diverse and engaging game environments. With the rise of indie game studios (Stephenson, 2022), who often face limited resources and tight budgets, the need for efficient and cost-effective development solutions is more pressing than ever. This project aims to develop a PCG tool specifically designed for generating dungeons for the Unity game engine (Unity Technology, 2005), with a focus on algorithm selection, seamless export processes, and efficient user interfaces (UI).

The importance of this project lies in the ability to streamline the level design process, particularly for indie developers. By automating the generation of dungeon map layouts, developers can save valuable time and resources, enabling them to focus on other crucial aspects of game design.

The inspiration for this project arose from the previously mentioned increase in the number of indie game studios, coupled with the popularity of procedurally generated content in the gaming industry. Investigating this subject is worth pursuing, as it has the potential to significantly impact the quality and efficiency of game development.

1.1 Project Objectives

- Develop a user-friendly PCG tool for generating diverse and engaging dungeon environments in Unity.
- Design an efficient and intuitive UI to streamline the dungeon generation process for developers.
- Give the ability for the user to use their own assets when generating the 3D version of the dungeon.

1.2 Key Deliverables

- Develop software that contains procederal generation algorithms such as Wave Function Collapse, Cellular Automata, Binary Space Partitioning, and more.
- A customizable set of parameters for each algorithm, allowing designers to have more control over the generated dungeon environment and tailor it to their specific requirements.
- A downloadable asset pack for the Unity game engine.

2. Literature review

2.1 When to use or not use PCG

Many games employ Procedural Content Generation (PCG) to varying degrees. Some, like Battlefield 3 (DICE, 2011), use tools like SpeedTree (Interactive Data Visualization, Inc. 2002) to create unique trees for their expansive multiplayer maps. Others, such as Spelunky (Mossmouth, 2013), base their entire gameplay on procedural generation to create captivating and non-repeating levels. While generated content can be enticing, it also introduces challenges, particularly with the "lack of control" it gives designers (van der Linden, Lopes, and Bidarra, 2014). With PCG, outcomes depend on the implementation of the algorithm, leading to possible unpredictability. However, for genres like dungeon crawlers, which typically favour fast-paced gameplay, PCG can offer benefits like uniqueness and endless possibilities, resulting in a different type of game with high replayability, as experienced in Enter the Gungeon (Devolver Digital, 2016).

2.2 What is the definition of a dungeon in videogames

Dungeons in real life differ greatly from those in games. Game designers must consider replayability and level design, among other factors, to maintain player engagement. Shaker et al. (2016) defines RPG dungeon levels as “labyrinthic environments, consisting mostly of interrelated challenges, rewards and puzzles, tightly paced in time and space to offer highly structured gameplay progressions”. Furthermore, Dahlskog et al. (2015) propose a classification system for dungeons after analysing various games, resulting in five classes:

- Open Area Dungeons: Wide open spaces.
- Mazes: Multiple paths leading to one room.
- Labyrinth: Singular paths spanning the whole dungeon.
- Connected Rooms: Rooms connected with each other with no corridors.
- Rooms and Corridors: Rooms connected with each other via corridors.

Each class has unique characteristics and is chosen based on the desired gameplay outcome. For instance, 'open area dungeons' emphasize character movement and can accommodate larger battles or ranged combat, Like in Enter the Gungeon (Devolver Digital, 2016). The Binding of Isaac Rebirth (Nicalis, Inc. Edmund McMillen, 2014) uses the connected rooms Layout to maintain a high pace without unnecessary breaks between rooms. Another reason for selecting a particular class may be performance; corridors can be used to hide rooms and save resources. Implementing occlusion culling allows developers to improve performance by not rendering objects hidden from the player's view. By strategically placing corridors to hide certain rooms, games can potentially save computational resources and maintain smoother gameplay experiences.

2.3 How to build an engaging tool/software UI

When providing users with multiple options for dungeon creation, it is essential to have an intuitive and efficient UI layout, as Anderson et al. (2010) explains that doing this will minimize the time and frustration needed to accomplish the desired goal from the user. McKay (2013) tries to break down what an “intuitive” UI really means but brings up the fact that for “most people the definition of intuitive is, well, intuitive itself”. Nevertheless, McKay additionally, explains the most common attributes of a fully user-appreciated UI:

- Discoverability: Clear start and end point.
- Understandability: Quick understanding of what the UI element does.
- Affordance: Contains visual properties that give hints to the user of the element’s usage.
- Predictability: UI delivers what it expects.
- Efficiency: minimum number of inputs needed to perform an action.
- Responsive Feedback: Give the user feedback that the action has been initiated.
- Forgiveness: the ability to undo or fix a “miss input.”
- Explorability: The UI is free to be explored before any commitments.

2.4 Previous Existing Tool for PCG

Within the Unity ecosystem, several other level procedural generation assets packs can be found. Two notable examples include Roguelike Generator Pro - Level Dungeon Procedural Generator (RGP) (Nappin, 2022) and Dungeon Architect (DA) (Code Respawn, 2022).

RGP is an asset pack that focuses on generating procedurally designed dungeons, with a primary emphasis on roguelike games. It offers customizable parameters and a user-friendly interface, allowing developers to create various dungeon layouts and room configurations. The main strengths of RGP lie in its ease of use and the ability to generate interconnected dungeons specifically tailored for roguelike games. This strength of RGP is important because it helps streamline the development process and ensures a consistent gameplay experience across different levels for the user. Another strength of the asset pack is its ability to create 2D, 2.5D, and 3D levels, catering to a wide range of visual styles and game types.

DA, on the other hand, is a more versatile procedural generation tool that can be used for creating more complex types of game environments, including dungeons, caves, and outdoor terrains. It comes with a powerful visual node-based editor, which enables developers to design their levels by connecting various building blocks. DA's flexibility stems from its modular approach, allowing developers to combine different components to create unique level structures. Its extensive features cater to different game genres, making it a valuable tool for a broader range of projects. This flexibility is crucial for developers as it provides the freedom to create diverse and engaging game environments, ultimately enhancing the player experience.

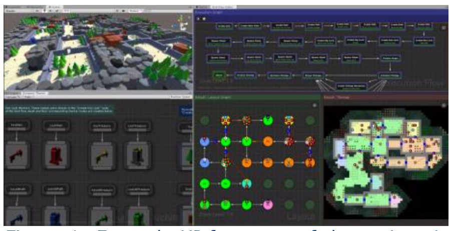


Figure 1: Example UI from one of the options in Dungeon Architect (Code Respawn, 2022)

3. Research questions

The main research question for this report is: How do we develop a tool to procedurally generate a functional 3-dimensional dungeon.

This can further be broken down into additional research questions:

- What role do procedural generation algorithms play in optimizing the development process and maintaining the balance between level design quality and resource constraints?

Procedural generation algorithms streamline the development process by automating the creation of diverse environments, saving time and resources. Fine-tuning these algorithms allows developers to prioritize performance, visual appeal, or gameplay mechanics, aligning with their goals and available resources.

- What are the essential features and design considerations for a custom editor in Unity and how can they be implemented and evaluated?

A custom Unity editor should incorporate a clean and intuitive user interface, easy access to algorithm selection and parameter adjustment. Additionally, it should be able to walk the user step by step to ensure undesired behaviours do not occur. A heuristic evaluation approach can be performed on the Editor UI to self-assess its usability and effectiveness.

- What are the limitations and challenges of using procedural generation for level generation and how can they be addressed in future research and development?

Procedural generation can generate repetitiveness, lack of control over the design, and it is important to consider possible performance headroom needed to run these algorithms. Future research and development can address these challenges by exploring new, more high-performance ways to write the existing algorithms and enhancing customization options.

4. Research Method

The research methodology for this project will involve secondary research, focusing on journals, books, and research papers published in the fields of procedural generation, game-related applications, and the creation of efficient user interfaces. The primary resources for locating relevant sources will be Google Scholar and Google Books, chosen for their extensive databases and ease of use.

To find pertinent papers, specific keywords derived from the central research question's theme were used, including:
- UI in software
- PCG in games
- Procedural generation algorithms
- Dungeons in Games

The rationale behind choosing these source types lies in their comprehensive explanations and in-depth coverage of the subject matter. These sources serve as reliable references, providing sufficient insight into each algorithm, which is crucial for the development and expansion of the artifact.

Self-serving bias is a cognitive bias where individuals attribute their successes to internal factors and their failures to external factors. This can lead to distorted perceptions and evaluations of one's own abilities or work (Forsyth, 2008). In the context of this project, as it is subject to self-evaluation, there is a risk of self-serving bias affecting the assessment of the artifact's functionality and outcome.

To mitigate self-serving bias, a set of binary goals was established at the beginning of the project. Binary goals are clear and objective, with either a "yes" or "no" outcome, leaving little room for biased interpretation. The established goals are as follows:

- Implement at least 4 of the basic algorithms chosen from the initial proposal, including:
    - Cellular Automata.
    - Perlin noise.
    - Perlin Worms.
    - Voronoi Generation.
    - Wave Function Collapse.
- Allow the mixing of algorithms to give the user more control.
- The generator outputs 3D environment for the user to build upon.
- Build an in-Editor Interface for the user to use the tool.

These goals will serve as indicators of not only the project's progress but also the overall functionality of the artifact. An additional evaluation criterion involves assessing how closely the generated dungeon resembles and feels like those found in other games. As part of the project's objective is to create a ready-to-design dungeon layout for games, it is essential to incorporate features that have proven effective in previous games.

Moreover, a crucial aspect of the project, which will determine its success, is the ability to export the created product as a .FBX file and by allowing users to implement the artifact via the package manager in the Unity engine, importing the package into their own projects.

By setting binary goals, the evaluation process becomes more objective, reducing the impact of self-serving bias on the assessment of the project's success.

5. Ethical and professional principles

With the introduction of Procedural Content Generation (PCG) within a project, there is the possibility that job roles may be overtaken by a computer, depending on the scope of the system's integration. For example, in the case of the developed artifact, the goal is to provide users with different ways to create a map for their game. This means that a level designer's position might be partially filled by the system, with professionals still needed to give overall direction to the developers regarding the level's appearance.

This gradual job takeover is reminiscent of the current trend with Artificial Intelligence (AI) and robots. "Fewer people work in manufacturing today than in 1997, thanks in part to automation" (Rotman, 2013). While AI and PCG differ in many ways, they share some common principles, such as the ability to perform repetitive tasks quickly. Computers and machines excel in this field, whether it is quickly prototyping a new map using PCG or performing monotonous movements like a robot.

To minimize the impact of the artifact on potential job positions, it is essential to implement as many features and characteristics as possible without making the tool too complex for individual developers. By offering scalable complexity, the project can cater to a single developer's needs while still allowing a level designer to delve deeper into creating the perfect level. This approach ensures that the tool remains an asset without eliminating job opportunities for professionals in the field.

Furthermore, to ensure inclusivity and accessibility of the tool for a diverse range of users, it is important to include features that accommodate disabilities such as dyslexia or colour blindness. For example, in the future, this could be done by increasing the font size or changing the colour of the buttons in relation to the background of the editor. By prioritizing accessibility and inclusivity in the design of the custom Unity editor, the tool becomes more versatile and user-friendly, promoting broader adoption and a more inclusive game development community.

6. Research findings

6.1 Advantages and disadvantage of PCG in Dungeon Generation

One of the most significant findings in the research phase was the trade-offs involved in whether to use PCG in dungeon generation. The analysis of various games revealed that PCG could offer increased replayability and uniqueness in dungeon crawlers, such as Crypt of the NecroDancer (Brace Yourself Games, 2015). These advantages directly influenced the decision to implement PCG in the dungeon generation tool, as it provided the potential for creating a more engaging and dynamic experience for players.

However, the research also highlighted the challenges of unpredictability and lack of control over outcomes associated with PCG, as noted by van der Linden et al. (2014). This finding emphasized the importance of incorporating customizable parameters in the dungeon generation tool, to allow designers to exert more control over the output generated. Additionally, the concept of user intervention mechanisms, such as before the generation of the level, where the user can individually change the tile type to their liking, can help users correct or even open sections of the generated dungeon to fix issues or connect different parts of the game that are not related to the dungeon generation. By considering user-defined settings in the design process, a dungeon generation tool can address some of the drawbacks of PCG while still benefiting from its advantages.

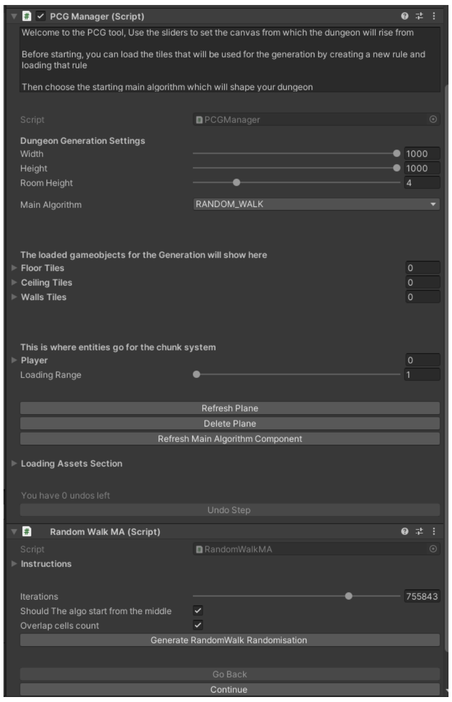
6.2 Characteristics and Impact of Different Dungeon Types in Games

The classification system proposed by Dahlskog, Björk, and Togelius (2015) proved to be invaluable for understanding the unique characteristics of each dungeon type and their impact on gameplay. The analysis revealed that the choice of dungeon type could significantly influence the player's experience, dictating factors like pacing, level design, and resource management.

For example, the research showed that 'open area dungeons' can emphasize character movement and accommodate larger battles or ranged combat, as seen in Enter the Gungeon (Devolver Digital, 2016). On the other hand, the connected rooms method, used in The Binding of Isaac Rebirth (Nicalis, Inc. Edmund McMillen, 2014), maintains a high pace without unnecessary breaks between rooms. These findings led to the inclusion of multiple dungeon generation options and inclusion of even more algorithms that were previously proposed in the tool, allowing designers to choose the type best suited for their intended gameplay experience.

6.3 Principles of Effective UI Design for Dungeon Generation Tools

The research on intuitive and efficient UI design, as presented by Anderson et al. (2010) and McKay (2013), guided the development of the dungeon generation tool's user interface. The identified attributes of user-friendly UI, such as discoverability, understandability, and efficiency, informed the layout and design choices for the tool. By prioritizing these principles, the tool's UI minimizes the time and frustration needed for users to achieve their desired goals.


PCG Manager (Script)

Welcome to the PCG tool, Use the sliders to set the canvas from which the dungeon will rise from

Before starting, you can load the tiles that will be used for the generation by creating a new rule and loading that rule

Then choose the starting main algorithm which will shape your dungeon

Script: PCGManager

Dungeon Generation Settings
Width: 1000
Height: 1000
Room Height: 4

Main Algorithm: RANDOM_WALK

The loaded gameobjects for the Generation will show here
Floor Tiles: 0
Ceiling Tiles: 0
Walls Tiles: 0

This is where entities go for the chunk system
Player: 0
Loading Range: 1

Loading Assets Section

You have 0 undos left

Random Walk MA (Script)
Script: RandomWalkMA
Instructions

Iterations: 755843
Should The algo start from the middle: ✓
Overlap cells count: ✓

Refresh Plane
Delete Plane
Refresh Main Algorithm Component
Undo Step
Generate RandomWalk Randomisation
Go Back
Continue

Figure 2: The final UI of the tool with the random walk algorithm (Appendix D) chosen as the main algorithm.

Incorporating these research findings on effective UI design, the dungeon generation tool delivers an enhanced user experience, allowing designers to effortlessly create engaging and dynamic dungeons.

6.4 Comparison and Analysis of Existing PCG Tools

Analysing popular existing PCG tools within the Unity ecosystem, such as Dungeon Architect (DA) and Roguelike Generator Pro - Level Dungeon Procedural Generator (RGP), led to the identification of essential features and functionality that could be incorporated into the project. These findings significantly influenced the development process and the features included in the final version of the tool.

The analysis highlighted the importance of comprehensive documentation and wikis to explain the product at hand, regardless of its simplicity. Users' concerns regarding DA's documentation (Code Respawn, 2022) quality informed the decision to provide a detailed wiki for the project. This ensures that users have access to a reliable source of information and guidance, improving the overall user experience. Furthermore, the research revealed that users wished to have a way to save the generated dungeon layouts for future use and access. This finding guided the decision to include this feature in the project, adding a layer of convenience to the tool. This allows users to revisit and refine their creations without the fear of losing a potentially useful layout they might want to utilize later.

Furthermore, a user (nappin, 2022) noted that the shapes of the generated structures by RGP were not that varied. This finding led to the decision to implement even more algorithms that were not previously mentioned in the project, enabling the generation of even more unique and interesting dungeon designs. By providing designers with a richer set of options to explore and implement in their projects, the personal project sets itself apart from the more predictable structures generated by RGP. This emphasis on diversity and creativity in dungeon layouts ensures that the project offers a robust and versatile solution for game developers, striving to make the tool adaptable to a wide range of genres.

6.5 Impact of Research Findings on Tool Development and Project Stages

The research findings played a crucial role in shaping the development and direction of the dungeon generation tools. They provided insights into the advantages and disadvantages of implementing PCG into games, contributed to the creation of an intuitive UI that users can easily interact with, and informed the comparison and analysis of existing PCG tools. By examining customer reviews, the research helped identify essential features and areas for improvement. Collectively, these findings guided the tool's development, ensuring its adaptability and relevance to a wide range of genres and user needs.

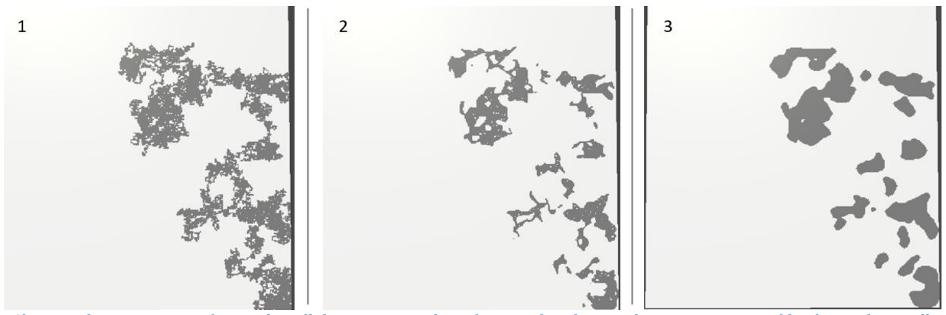
7. Practice

This section will delve into the critical aspects of the project's implementation, focusing on the most significant components that contributed to the project's success. This section will discuss the standardization of algorithm input and mixing, user interface (UI) and custom editor development, user's ability to add their own assets, performance optimization and debugging, and documentation and code quality. Each topic will be thoroughly explored, including the challenges faced and the methods employed to overcome them.

7.1 Standardization of algorithm input and mixing

The development process began with planning a class to standardize the inputs and outputs for all algorithms and functions used, which was a crucial step as it allowed for seamless integration and mixing of different algorithms through a single class. The class was essential for effectively managing and standardizing the inputs and outputs for all algorithms and functions used in the project, it contains multiple variables.

As the project progressed, it became evident that some algorithms complimented each other well, creating a unique and interesting dungeon when combined. For instance, the cellular automata (Appendix D) algorithm is a standard addition as a step (if the player desires to use it) for most of the algorithms available, this is due to its ability to either cut generated structures in the layouts to create rooms or to just smooth out the shape that the outcome will be (Figure 3). Moreover, this is also supported by the ability choose which iteration of the cellular automata to run on the current generated layout, to either run the full iteration of the algorithm where things will be added and taken away, or to run the stage where to only take away tiles. The ability to mix and match algorithms based on their individual strengths and weaknesses allowed for a greater variety of dungeon designs and increased the tool's overall versatility.



Showing the progression of using the cellular automata algorithm to refine the initial generation created by the random walk algorithm.
Stage 1: Shows the initial random walk output.
Stage 2: Demonstrates how using the clean-up variation of the cellular automata algorithm creates independent rooms.
Stage 3: Displays the result of using the normal iteration of the cellular automata algorithm to fill in gaps and smooth out the rooms.

Another significant product of the standardization of algorithms was that, although a handful of algorithms were in a bubble of their own, such as the L-system (as this algorithm is grammar-based), there is still the ability to mix entire canvases together at the stage before the generation of the 3D dungeon. This allows for the mixing of different generated maps that previously used different main algorithms, therefore giving the ability to mix two or more different kinds of layouts.

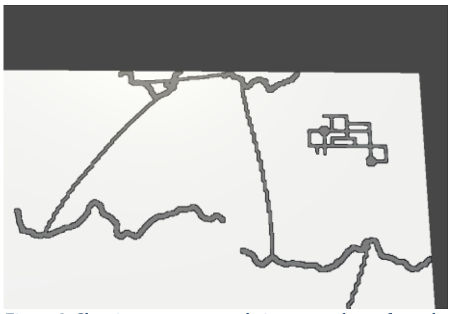


Figure 3: Showing two canvases being merged, one from the L-System algorithm and one from the Perlin worms (Appendix D) to create two distinguished sections to the dungeon.

However, the Wave Function Collapse (WFC) (appendix D) algorithm proved to be an exception. The WFC had to be in its own bubble and could not be mixed with other algorithms because of its nature, which is not based on weight or the current cell's status but on the current asset that is next to the cell. This inherent difference meant that the WFC had to be treated separately, preventing it from being combined with the other algorithms in the same way.

7.2 UI and custom Editor

The custom editor was designed with simplicity and efficiency in mind, allowing users to quickly access and understand all the available settings each algorithm could offer.
One of the most noteworthy achievements in this stage was the implementation of a custom window for creating rulesets for the WFC algorithm. This was made possible by utilizing the experimental GraphView package (Unity Technologies, 2023) provided by Unity, which offers a flexible and extensible framework for creating custom node-based graph interfaces.

The WFC algorithm requires a ruleset that dictates the arrangement of tiles, and the custom editor graph system was created to streamline this process. The graph system consists of three types of nodes:

1. Main Rule node: This node has four inputs (left, right, up, and down) and represents a specific tile. Users can connect other nodes to these inputs to define the tile's connectivity rules.

2. Sub Rule node: This node represents a side tile and connects to one of the inputs of the Main Tile node, specifying the allowed connection for that side of the main tile.

3. Quick Rule node: This node has checkboxes for left, right, up, and down, rather than inputs, and is used for tiles that can connect to any other tile. Users can simply check the relevant boxes to define the connectivity rules for this tile, making it a more efficient option when dealing with tiles that have no restrictions on their connections.

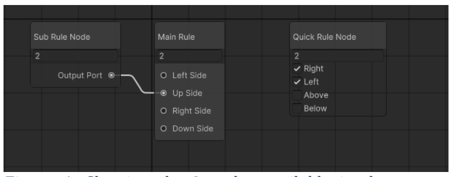


Figure 4: Showing the 3 nodes available in the custom graphing environment for the WFC algorithm.

By utilizing these nodes, users can easily create and modify the ruleset for the WFC algorithm, ensuring their desired dungeon generation outcome.

7.3 User's ability to add their own assets

The dungeon generation tool provides users with multiple options to create 3D dungeons that cater to their specific needs and preferences. By offering two distinct generation methods: tile generation and mesh generation.

Tile generation allows users to define the types of tiles that make up the floors, ceilings, and walls, as well as control the spawn ratio for each tile type. This ensures that the final dungeon generation aligns with the desired aesthetics and design while still maintaining the procedural nature of the tool. Additionally, to ensure compatibility with the broadest range of tiles, users will be prompted to specify whether the tiles being used are directional or non-directional before initiating the generation process. In this context, "non-directional" means the user is using cubes as tiles to generate the dungeon, and their orientation is not important. In contrast, when using "directional" generation, the walls will be generated so they always face the outer side of the dungeon.

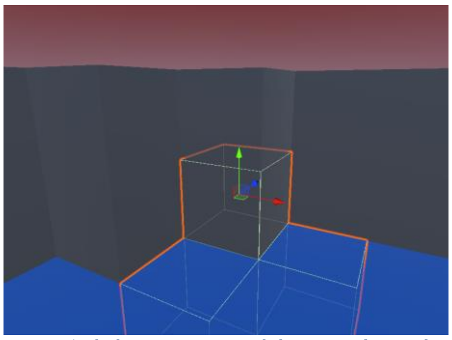


Figure 5: Block generation example having a cube per tile.

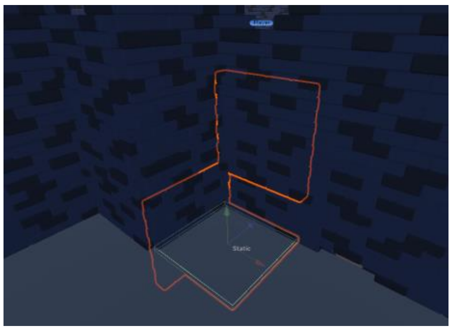


Figure 6: Multiple objects per tile to maximise space and contour with the outside tiles.

On the other hand, mesh generation leverages the marching cubes algorithm to create a 3D mesh backbone that can be imported into 3D modelling softwares like Blender (1994) and Maya (1998) for further customization and manipulation. This option provides users with even more flexibility, as they can refine the dungeon design beyond the capabilities of the generation tool, tailoring it to their specific requirements and artistic vision.

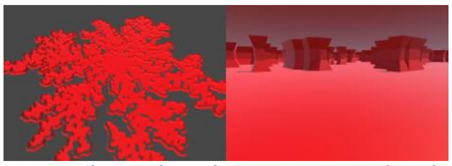


Figure 7: Showing the mesh vertices generation from the outside (left) and inside (right). Dungeon algorithm used is the diffusion limitation algorithm (Appendix D).

Furthermore, after the user has created their 3D dungeon, they can provide their own world object assets, which the tool will place throughout the environment using the Poisson disc sampling algorithm (Appendix D). This algorithm allows for an optimized distribution of objects with an element of randomness, ensuring that the placement appears natural and visually appealing while avoiding excessive clutter. Users can dictate the radii used by the Poisson algorithm and the ratio at which different object assets are spawned, offering greater control over the density and distribution of the objects within the dungeon. Moreover, users can determine the height at which these objects spawn, which means they could choose to spawn objects containing lights. By adjusting the radii, users can create either gloomy or bright scenes.

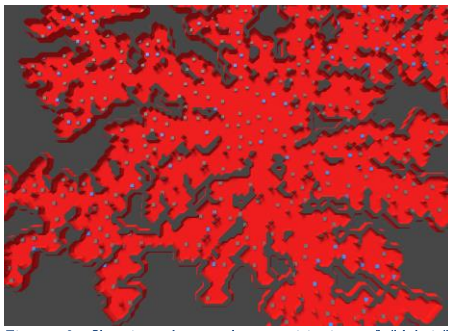


Figure 8: Showing the random positioning of "debris" around the map, with the grey box being 4 times as likely to appear in the map.

7.4 Performance Optimization and Debugging

Performance optimization and debugging played a crucial role in the development of the tool. Ensuring that the project ran smoothly and efficiently was vital to providing a positive user experience.

One of the first performance-related challenges encountered during development was the flood fill algorithm, which is used to determine the boundaries of each room currently on the canvas. The initial implementation utilized a recursive algorithm, leading to stack overflow issues when the canvas size exceeded 400 by 400. The issue stemmed from the fact that each function call in a recursive algorithm is placed on the stack, which has a limited size. To address this problem, the algorithm was redesigned using a standard loop instead of recursion, eliminating the need for additional stack allocations. This modification not only resolved the stack overflow issue but also improved the performance of the algorithm, as using a loop is generally less resource-intensive compared to placing multiple function calls on the stack.

Another performance optimization implemented during the project was the introduction of a chunk system for the 3D dungeon generation. When users choose to provide their own tile sets for the dungeon generation stage, many objects are instantiated in the scene. This is because, depending on the size of the canvas, each coordinate is potentially an object, with the height also needing to be considered. Because many objects can be created, a chunk system was implemented to try and reduce the number of objects drawn in every update. The chunk system is only available when the game starts, and the user has the choice of which objects to draw chunks around. This allows for even multiple setups where two players are in different sections, and the chunk system can still work, additionally providing the ability to choose how many chunks to draw around the player for even more performance and customizability.

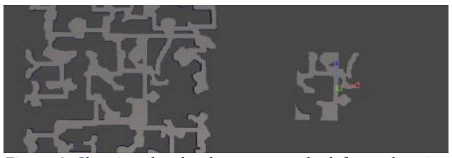
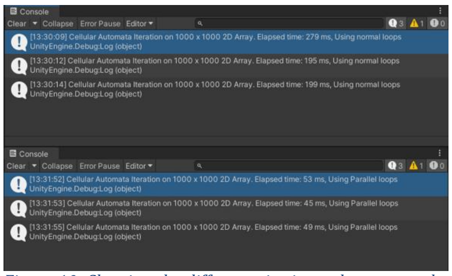


Figure 9: Showing the chunk system on the left not drawing the sections of the dungeon not close to the user to save on performance.

Finally, parallel programming in C# was employed as a performance optimization technique during the development process. By utilizing the 'System.Threading.Tasks.Parallel' namespace, array loops were processed more efficiently by distributing the workload across multiple CPU cores, leading to faster execution times. This parallelism allows for concurrent execution of multiple iterations of the loop, thus significantly improving the processing speed. However, it is important to note that parallel programming introduces additional complexity, such as potential race conditions or synchronization issues. Therefore, algorithms that depend on multiple loops in sequence, such as the Voronoi algorithm (appendix D), which has one loop to deal with the algorithm itself and another loop to handle the drawing of the rooms, were not suitable for parallel processing. On the other hand, algorithms such as Cellular Automata have seen great improvement with parallel programming, achieving up to an 80% reduction in time taken, as shown in Figure 10.



```
[13:30:09] Cellular Automata Iteration on 1000 x 1000 2D Array. Elapsed time: 279 ms, Using normal loops
UnityEngine.Debug:Log (object)
[13:30:12] Cellular Automata Iteration on 1000 x 1000 2D Array. Elapsed time: 195 ms, Using normal loops
UnityEngine.Debug:Log (object)
[13:30:14] Cellular Automata Iteration on 1000 x 1000 2D Array. Elapsed time: 199 ms, Using normal loops
UnityEngine.Debug:Log (object)
```

```
[13:31:52] Cellular Automata Iteration on 1000 x 1000 2D Array. Elapsed time: 53 ms, Using Parallel loops
UnityEngine.Debug:Log (object)
[13:31:53] Cellular Automata Iteration on 1000 x 1000 2D Array. Elapsed time: 45 ms, Using Parallel loops
UnityEngine.Debug:Log (object)
[13:31:55] Cellular Automata Iteration on 1000 x 1000 2D Array. Elapsed time: 49 ms, Using Parallel loops
UnityEngine.Debug:Log (object)
```

Figure 10: Showing the difference in time taken to run the cellular automata algorithm for a 1000 by 1000 grid.

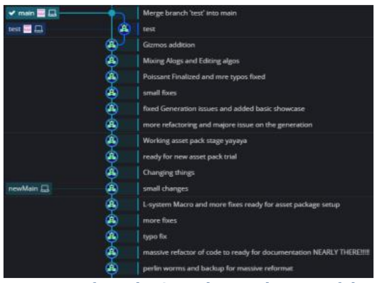
Debugging was an essential aspect of the project, as identifying and fixing errors and performance bottlenecks was crucial to maintaining the tool's functionality and efficiency. The flood fill issue mentioned earlier is an example of a bug that was identified and resolved during the debugging process. By continuously monitoring the tool's performance and addressing any issues that arose, the project was able to achieve its goals and provide a versatile and efficient dungeon generation solution.

7.5 Documentation and Code Quality

Documentation and code quality played a significant role in the efficiency of this project's development. By ensuring clear documentation and well-organized, maintainable code, the tool became easier to understand, modify, and expand upon by others.

One of the measures taken to maintain high-quality code was adhering to a specific coding standard, which helped promote consistency and readability across the project. In this case, the typical C# coding conventions (Microsoft, 2023) were followed, which include guidelines such as using PascalCase and camelCase for naming. PascalCase capitalizes the first letter of each word in a compound word or phrase (e.g., FunctionName), while camelCase capitalizes the first letter of each word except for the first one (e.g., localVariable). The conventions dictate using PascalCase for method and property names, and camelCase for local variables. By following these conventions, the codebase became more accessible to others who may be familiar with the C# language and its common practices.

Another aspect of maintaining code quality was the use of a version control system – specifically, GitHub (2008). This allowed for easy tracking of changes and a means to revert to previous versions of the code when necessary. For instance, during the transition to the asset pack implementation, the setup of assembly files required for C# and Unity to recognize the asset pack presented minor yet substantial complexity. To mitigate potential risks, a backup of the project was created in case major issues arose, and reverting was necessary. Additionally, GitKraken (2014), a graphical user interface for Github, was used as a visual aid to better understand the repository's history and changes. This combination of tools made it easier to manage the codebase and ensure its stability throughout development.



```
Merge branch 'test' into main
test
Gizmos addition
Mixing Alogs and Editing algos
Poissant Finalized and mre typos fixed
small fixes
fixed Generation issues and added basic showcase
more refactoring and majore issue on the generation
Working asset pack stage yayaya
ready for new asset pack trial
Changing things
small changes
L-system Macro and more fixes ready for asset package setup
more fixes
typo fix
massive refactor of code to ready for documentation NEARLY THERE!!!!
perlin worms and backup for massive reformat
```

Figure 11: View from the GitKraken application of the main GitHub repo of the project showing the saving of the progress before major changes and tests to the codebase.

8. Discussion and outcomes

The primary contribution of this research lies in the development of a versatile procedural dungeon generation tool that accommodates multiple algorithms and the ability to mix different algorithms together. This innovation allows for the creation of unique and engaging dungeons by leveraging the strengths and weaknesses of various algorithms, resulting in diverse layouts that can be tailored to specific game design requirements. A key aspect of this achievement is the standardization of the classes used by each algorithm, which streamlines their integration and usage within the tool, enhancing user experience and providing more control over the generation process. By enabling the combination of different algorithms, the tool provides game developers with a powerful means to create a wide range of dungeon environments, enhancing the overall gaming experience.

The creation of a wiki to support this versatile procedural dungeon generation tool further enhances its practical applicability in the game development process. Additionally, the creation of a documentation simplifies the integration of the tool into existing game development workflows and provides developers with an accessible and organized framework for utilizing the various algorithms and features of the tool in their own projects.

8.1 Evaluation of the Generated Layout

The expressive range of the tool's generated dungeon layout is essential for its overall success. In this case, the layout refers to the generated 3D model or 2D texture representing the structure of the dungeon, which designers can use as a starting point for their level design. As Smith et al. (2010) suggest, the generator's worth is better judged by the style and range of levels it can create. The layout generator in this project offers a multitude of base algorithms, parameters, and customization options, enabling designers to craft a diverse range of outcomes.

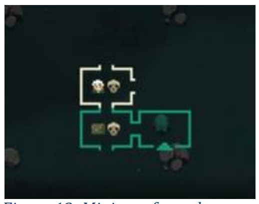
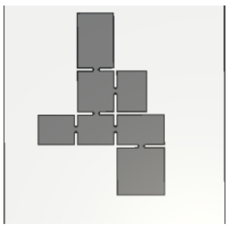
This, in turn, allows striking a balance between creative input and automation, leading to more efficient and effective level design processes. Furthermore, the high-level parameters and features provide designers control over the potential placement of gameplay properties, while the generator handles most of the design work.

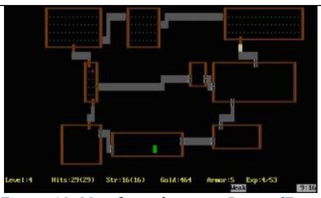
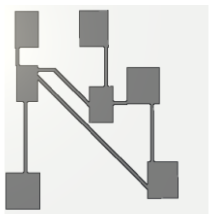
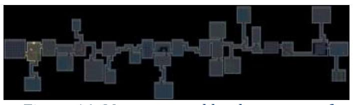
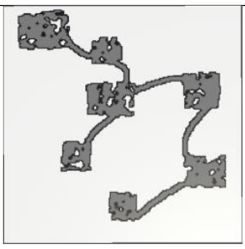
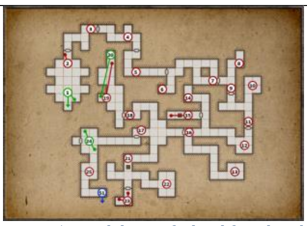
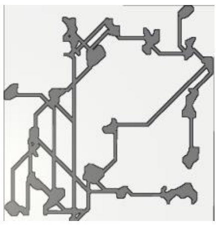
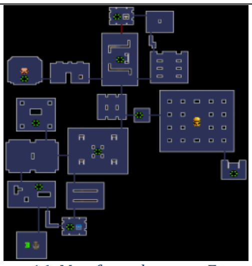
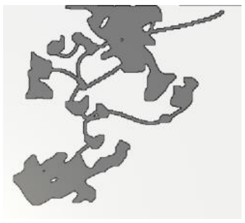
A crucial aspect of the layout's usability as a base for level creation is its ability to generate a solid foundation that designers can build upon and modify to suit their specific requirements. The layout generator is designed to create and act as a canvas to which the user can add their own gameplay mechanics with ease, whilst keeping the complexity of the generation as a key selling point. By providing a robust and adaptable starting point, the layout generator streamlines the level design process, enabling designers to focus on refining and polishing their levels.

The adaptability of the layout generator is another essential factor when evaluating its usability for level creation. As Pérez-López et al. (2011) mention, it would be advantageous, especially for small game developers, if level generators did not require reconstruction or recoding for specific games. In this project, the layout generator is designed to be adaptable across different games and genres. By creating layouts suitable for various types of games, the tool demonstrates its versatility, making it an asset for developers with diverse needs. Additionally, its ability to cater to different design requirements and preferences allows users to create levels that accommodate a wide range of gameplay styles, further emphasizing the layout generator's adaptability and potential for use in different game development contexts.
Furthermore, when evaluating the generator output in the context of the classes proposed by Dahlskog et el. (2015), the layouts generated do match what each of the classes' main descriptive features. This alignment with the established dungeon classes demonstrates the generator's ability to cater to various gameplay styles and preferences, further highlighting its versatility and adaptability in the field of procedural dungeon generation.

Evaluating the layout generator's ability to support gameplay beyond roguelike games is also crucial. Dormans (2010) notes that while some algorithms excel at creating roguelike games, their output often supports limited gameplay. A useful layout generator should be able to create levels that enable a wide range of gameplay styles, rather than being confined to a specific genre or style. The tool developed in this project provides numerous customization options and parameters that facilitate the creation of diverse layouts which can be imported into any genre, for example, the room-to-room generation could be integrated into a building layout the user can explore in an exploration game, demonstrating its capacity to support various gameplay styles and genres beyond roguelike games.

To evaluate the dungeon layout generator's effectiveness and versatility, a table will compare the dungeon types defined by Dahlskog et al. (2015) with the tool's output. The table includes the dungeon type, a detailed description from Dahlskog et al., a real game example, and a version generated by the tool. This comparison addresses the criterion mentioned in the methodology section, assessing the resemblance between the generated dungeons and those found in other games or asset packs.

Dungeon Type	**Description**	**Real Game Examples**	**Tool's Generated Layout**	**Algorithms & Comparison**
Connected Rooms	This type of dungeon primarily features a series of discrete, interesting sites or rooms connected without explicit corridors, pathways, or tunnels. Players move from one room to another, with each	 *Figure 12: Minimap from the game Moonlighter showing rooms connected to rooms (Moonlighter Wiki, 2018)*		A simple random room generation was used, ensuring the rooms were adjacent to each other. The generated layout closely resembles the Connected Rooms style found in games like Moonlighter, with seamless connections between rooms being the strong point. The weak point might be a lack of variety in room shapes.

	room often serving as a focal point.			
Rooms & Corridors	Sparse dungeons with few rooms joined by simple, non-branching corridors.	 Figure 13: Map from the game Rouge (Epyx, inc., 1985)		Random room allocation was employed, and the rooms were connected using the A* pathfinding algorithm. The generated layout is similar to the level found in the game Rogue.
Labyrinths	Unicursal structures with a single path leading through the dungeon.	 Figure 14: Map proposed by the creator of shattered pixel (Debenham, 2017).		Random room allocation was used, with Cellular Automata applied to the rooms for more variation. The rooms were connected using a Bezier curve to lay down the corridor. The generated layout is comparable to the generation shown in Figure 14, due to its universal nature that is also demonstrated in the example, with the occasional off-leading path.
Mazes	Multicursal layouts with multiple paths leading through the dungeon.	 Figure 15: Breakdown of a level found in the game called Legend of Grimrock (Game Guide, 2016)		Perlin noise was used to create small rooms across the canvas, and they were connected using the A* pathfinding algorithm to create the corridors. The strength of this layout is the high complexity and numerous branching paths.
Open Areas Dungeons	These dungeons consist of extremely open spaces (for a dungeon) with obstacles (e.g., thin walls) that hinder free manoeuvring. Tactical manoeuvring has greater	 Figure 16: Map from the game Enter the Gungeon (Boris, 2019)		A combination of the random walk algorithm and cellular automata was used to create the open areas. Cellular automata were applied to smooth out the generated layout. Bezier curves were used to create the connecting corridors. The generated layout shows a vast open space with organic shapes. Although this

	importance here, and corridors are uncommon.			is different from the rectangular nature of the rooms shown in Figure 16, the main idea of having large rooms with a couple of small ones is achieved.

Table 1: Comparison of generated dungeon layouts with Dahlskog's classifications, real game examples, and the algorithms used

In conclusion, the table demonstrates the alignment of the generated output with the dungeon types and showcases its similarity to other games. This confirms the layout generator's capacity to cater to various gameplay styles and preferences, making it an asset for game developers with diverse needs. By providing a wide range of customization options and parameters, the layout generator supports a broad spectrum of gameplay styles and genres beyond roguelike games.

8.2 Heuristic Evaluation of UI

Heuristic evaluation is a usability inspection method involving experts examining a product's user interface to identify issues based on established heuristics (Nielsen and Molich, 1990). In general, heuristics are a set of problem-solving strategies or principles that are based on practical experience and knowledge. In the context of usability evaluation, heuristics are a set of general principles or guidelines used to evaluate the usability of a product's user interface. As McKay (2013) states, "Design is communication, and a well-designed user interface is good communication." Heuristic evaluation ensures effective communication between users and the product. In cases where user feedback or testing is not feasible, Nielsen (1994) asserts that heuristic evaluation helps "find the usability problems in the design" and provides insights for improving usability. When a single evaluator conducts heuristic evaluation, biases can arise. Nielsen also cautions that "the evaluator's own background and knowledge will unavoidably influence the evaluation."

To mitigate bias, McKay (2013) suggests that evaluators should "think like their users" by familiarizing themselves with usability principles and considering different user profiles. Additionally, Nielsen (1994) recommends refining and revaluating the user interface iteratively, as "iteration of the design is the best way to improve the system."

Based on the decision of not gathering feedback from other users, a heuristic evaluation of the tool's UI interface was conducted, following Nielsen's ten usability heuristics (Nielsen, 2020):

Heuristic Number	Heuristic Name	Heuristic Explanation	Heuristic Application
1	Visibility of system status.	Keep users informed about what is happening within the system through appropriate feedback, such as progress indicators or notifications.	The UI present on the project provides real time feedback in the form of a progress bar, when heavily performant algorithms are freezing the inspector. This helps users understand the process and the progress being made.
2	Match between the system and the real world.	Use language and concepts that are familiar to the user and make sense in their context, making information appear in a natural and logical order.	Familiar terminology and concepts from game development and procedural generation are used to ensure that users can easily understand and utilize the tool.
3	User control and freedom.	Allow users to easily undo or redo their actions, providing them with a sense of control and flexibility in navigating the interface.	Users can Undo their actions a maximum number of 3 times but are not able to redo.
4	Consistency and standards.	Use consistent elements, design patterns, and terminology throughout the interface. Adhere to platform or industry standards to make it easier for users to	The project maintains a consistent design language and layout throughout the UI, adhering to common Unity inspector Elements to keep the visual uniform.

		understand and use the system.	
5	Error prevention and recovery.	Design the interface to minimize the likelihood of user errors by providing warnings, constraints, or confirmation dialogs before critical actions are taken.	Validations checks are present in the code wherever possible in the form of error console messages or error window pop ups.
6	Recognition rather than recall.	Make options, actions, and objects visible and easily accessible to reduce the cognitive load on the user, so they do not have to remember information from one part of the interface to another.	Tool tips and labels are present throughout the whole implementation to help the user remember the functionality of each feature without having to memorize the details.
7	Flexibility and efficiency of use.	Cater to both novice and experienced users by providing shortcuts, customizability, and adaptable interfaces that can be tailored to the user's needs and skill level.	The UI aims to be highly customizable. However, the tool currently offers only one layout, which means that the UI is the same no matter the experience. Furthermore, the system does not contain any short cuts.
8	Aesthetic and minimalist design.	Create a visually appealing interface that is uncluttered and focuses on the essential elements.	The UI in the project is designed to be straightforward and focused on the essential elements necessary for dungeon generation. This ensures that users can easily navigate the interface and perform tasks without being overwhelmed.
9	Help users diagnose and recover from errors.	Have message errors that tell the user the current issue with clear language and a possible way of fixing said issue.	When errors or issues are detected, the system provides clear and informative error messages that guide users towards resolving the problem.
10	Help and documentation.	To provide an accessible documentation for users who may need additional support or information's about the system.	The wiki serves as a valuable resource for the users, offering detailed explanations of algorithms, step-by-step guides on how to use such algorithms and ways to access the library. This ensures the user uses the tool in the most efficient way possible.

Table 2: Heuristic Evaluation of the Tool's UI - A summary of Nielsen's ten usability heuristics, their explanations, and their application to the user interface of the dungeon generation tool.

The heuristic evaluation of the Tool's UI demonstrates that the tool largely adheres to Nielsen's ten usability heuristics, indicating that it provides a positive user experience. However, it was identified that heuristic number 7 (table 2), which pertains to flexibility and catering to users with different experience levels, is not fully satisfied. The tool's interface currently offers only one layout and lacks shortcuts, which may limit its adaptability for users with varying levels of experience.

8.3 Non-Incorporated Features

During the development process, prioritizing the core functionality of the tool necessitated postponing or removing certain features from the project. Two features that were cut include the ratio of tiles that appear as side tiles when using the WFC algorithm, which aimed to combat the randomness of the algorithm output and provide more control to the user. Additionally, a graph-based algorithm using a similar graphing system to the WFC algorithm was not included in the

current iteration, as it required further research and development of a new graphing system to accommodate this novel concept. The idea was to have each room as a tile in the graph, and within that tile, have multiple variables the user could choose to stylize the room. When different room tiles were connected, it would create a corridor in the actual generation. This grammar-based generation would have helped address one of the key issues that PCG brings: control over the generated output. With this algorithm, the user would have had full control over what was generated.

Another aspect that was not included due to being out of scope is the ability to add gameplay features, such as the implementation of doors, traps, and other gameplay-specific elements. These features were considered for inclusion in the dungeon generation tool, but the decision was made to ensure the tool remained versatile and adaptable to a wide range of game genres and design styles. Introducing gameplay-specific elements could limit the tool's applicability. Although the tool is tailored towards the generation of "dungeons", the user could use the tool to create a map for a vastly different type of game. For example, Streets of Rogue (Matt Dabrowski, 2019) is an action top-down game that also creates its levels using PCG mechanics and revolves around exploring different rooms in buildings. The tool could potentially create a room-to-room variant of the layout and be used in this game genre, which is not focused on "dungeons".

8.4 Evaluate the limitations of the tool and areas for potential improvement

Although the tool has succeeded in its main goals and objectives, it still has some limitations that could be addressed in future iterations. One limitation is the absence of certain features, such as doors and other gameplay-specific elements, which were excluded to maintain the tool's versatility and broad applicability, as mentioned previously. However, incorporating these features during the 3D generation step, where players can adjust the tile map on a per-tile basis, could open the possibilities for including such elements. By making this an optional step, the tool's versatility and applicability can be preserved, as users can choose to skip it if desired. Integrating these features in a modular fashion would allow users to select the level of complexity they desire for their dungeon designs, ultimately enhancing the tool's utility without sacrificing its adaptability.

Several constraints and difficulties were encountered when working with the Unity Editor. One such issue was the self-editing feature, which allowed users to move a pointer to change the currently selected grid coordinate by pressing buttons in the editor. This approach had its limitations, as using arrow keys or the WASD keys on the keyboard would be a more widely accepted method for movement. However, integrating these keys could result in compatibility issues, as they may already have binding actions within Unity or the operating system. The addition of a Shift key press was also considered but faced similar constraints. Maintaining the editor button introduced another issue: the pointer is drawn using gizmos, which only update when the user hovers over the editor scene window. Since the buttons are in the inspector window, users receive no visual feedback upon clicking a button and must move their mouse back to the editor scene.

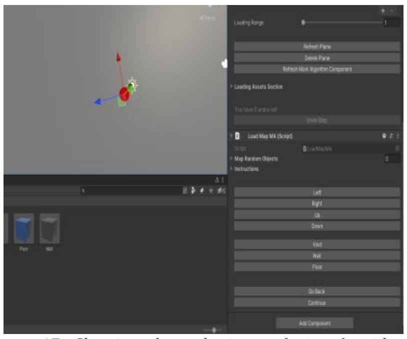
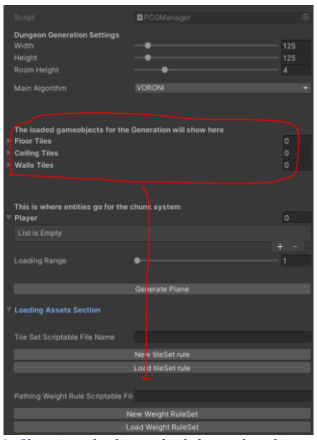


Figure 17: Showing the red gizmos (pointer) with the buttons to move it in the inspector window

Furthermore, there were challenges encountered in managing the order of some inspector elements, particularly when dealing with lists containing custom class types. Due to the serialization process in the Unity Inspector, certain elements, such as those in the PCG Manager script, could not be ordered correctly. The objective was to move the three lists containing the map objects into a drop-down window, along with the loading section, to keep things compartmentalized. However, issues arose when attempting to do so, as the serialization process had difficulty handling custom class types, unlike generic types such as integers.

The resolution to this issue would have required further development time, as the drop-down section needed to be created manually using the PropertyDrawer class (Unity Technologies, 2013). Unfortunately, due to time constraints, this improvement could not be incorporated within the scope of the project.



Script: PCGManager

Dungeon Generation Settings
Width 125
Height 125
Room Height 4

Main Algorithm VORONI

The loaded gameobjects for the Generation will show here
Floor Tiles 0
Ceiling Tiles 0
Walls Tiles 0

This is where entities go for the chunk system
Player 0
List is Empty

Loading Range 1

Loading Assets Section

Tile Set Scriptable File Name

Pathing Weight Rule Scriptable Fil

Figure 18: Showing the lists which keep the object tiles used in the generation phase

Furthermore, the tool's performance could be optimized to ensure the user has a better-quality experience when using it. The current performance may not be optimal, especially for users with lower-end hardware. Addressing these performance issues could make the tool more accessible and efficient, ultimately improving the overall user experience. By focusing on optimization and performance enhancements, the tool can reach a wider audience and provide a more seamless experience for users with various hardware configurations.

Lastly, conducting user testing in the future could lead to a better tool by providing valuable insights and feedback from a diverse group of users. While the heuristic evaluation of the UI offered a sufficient understanding of its effectiveness, user testing can provide an alternative perspective with less self-serving bias. This can help identify overlooked issues or limitations and highlight areas where the tool excels. User testing can guide future development and refinement, ensuring it caters to various users and use cases. Moreover, it can help validate proposed solutions for addressing the tool's limitations, such as the inclusion of gameplay-specific elements and performance optimization. Ultimately, incorporating user testing can result in a more robust, user-centric tool that meets the evolving needs of game developers.

9. Conclusion and recommendations

In conclusion, the project has successfully achieved its goal of providing a downloadable asset pack for the Unity game engine with a wide range of algorithms users can choose from to create their own 3D dungeons. Furthermore, with each incorporated algorithm offering a wide range of customization, the tool enables game developers to create diverse and engaging dungeon environments for a variety of game genres. The project's success is evident by the fact that all the project objectives, key deliverables, and binary goals set at the beginning of the project have been achieved. Moreover, the tool's UI largely adheres to Nielsen's ten usability heuristics, providing a positive user experience in most areas, except for heuristic 7, which pertains to flexibility and catering to users with different experience levels.

For the next steps, it is recommended that user testing be conducted to gather feedback and further refine the tool, including addressing the limitation in heuristic 7. Additionally, improvements to the speed of the algorithms should be explored, as well as the integration of more algorithms to increase the versatility of the project.

The project has the potential to be a fully viable addition to the Unity store that can be sold and used. Alternatively, with the wiki and library side of the project, it could become a bank of algorithms that any user can access and utilize.

10. References

van der Linden, R., Lopes, R. and Bidarra, R. (2014). Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1), pp.78–89.
[Accessed 5 December 2022]
Available at:
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6661386

Shaker, N., Liapis, A., Togelius, J., Lopes, R. and Bidarra, R. (2016). Constructive generation methods for dungeons and levels. *Procedural Content Generation in Games*, pp.31–55.
[Accessed 5 December 2022]
Available at:
https://link.springer.com/chapter/10.1007/978-3-319-42716-4_3

Dahlskog, S., Björk, S. and Togelius, J. (2015). Patterns, Dungeons and Generators. [online]
[Accessed 6 December 2022].
Available at:
http://julian.togelius.com/Dahlskog2015Patterns.pdf

Rotman, D. (2013). *How Technology Is Destroying Jobs*. [online] MIT Technology Review.
[Accessed on 9 December 2022]
Available at:
https://www.technologyreview.com/2013/06/12/178008/how-technology-is-destroying-jobs/.

Unity Technologies, 2023. Scripting API: Experimental.GraphView.GraphView. [online] Unity Documentation. Available at: https://docs.unity3d.com/ScriptReference/Experimental.GraphView.GraphView.html
[Accessed 20 Arpil 2023].

Code Respawn, 2022. Dungeon Architect: Reviews. [online] Available at: https://assetstore.unity.com/packages/tools/utilities/dungeon-architect-53895#reviews
[Accessed 20 April 2023].

nappin, 2022. Roguelike Generator Pro - Level & Dungeon Procedural Generator. [online] Available at:
https://assetstore.unity.com/packages/tools/level-design/roguelike-generator-pro-level-dungeon-procedural-generator-224345
[Accessed 20 April 2023].

Unity Technologies (2023). PropertyDrawer. Unity Documentation. Available at: https://docs.unity3d.com/ScriptReference/PropertyDrawer.html
[Accessed 20 April 2023].

Stephenson, S (2022) UK experiences a wave of video game Start-ups, says TIGA. TIGA [online] 6 June. Available from: https://tiga.org/news/uk-experiences-a-wave-of-video-game-start-ups-says-tiga
[Accessed 2 October 2022]

Mckay, E.N. (2013). *UI is communication: how to design intuitive, user centered interfaces by focusing on effective communication*. Amsterdam; Boston: Elsevier, Morgan Kaufmann.
Available at:
https://books.google.co.uk/books?hl=en&lr=&id=wNozxtKuOKcC&oi=fnd&pg=PP1&dq=effective+UI&ots=v987j5xarU&sig=aYOCzG8gqSu5w89RtOx5yXp-mJk&redir_esc=y#v=onepage&q=effective%20UI&f=false
[Accessed 5 December 2022]

Anderson, J., McRee, J., Wilson, R. and The EffectiveUI Team (2010). Effective UI. 'O'Reilly Media, Inc. Available at:
https://books.google.co.uk/books?hl=en&lr=&id=I7-IP5P-gdMC&oi=fnd&pg=PR9&dq=how+to+build+UI&ots=tLC9oRBkEr&sig=-w0pfNTH5COmOOUd9hLAqNyaFds&redir_esc=y#v=onepage&q=UI&f=false
[Accessed 6 December 2022]

Dormans, J. (2010). Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games. [online] Available at:
https://www.pcgworkshop.com/archive/dormans2010adventures.pdf
[Accessed 11 Aug. 2022].

Khalifa, A., Perez-Liebana, D., Lucas, S. and Togelius, J. (2011). General Video Game Level Generation. [online] Available at:
http://www.diego-perez.net/papers/GVGLG.pdf
[Accessed 8 April 2023].

Smith, G. and Whitehead, J. (2010). Analyzing the expressive range of a level generator. Proceedings of the 2010 Workshop on Procedural Content Generation in Games. doi:https://doi.org/10.1145/1814256.1814260
[Accessed 14 April 2023].

Microsoft, 2023. Coding conventions. [online] Available at: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions
[Accessed 20 April 2023].

Unity Technology (2005) Unity. [Computer Program].
Available at: https://unity.com/
[Accessed 5 December 2022]

Interactive Data Visualization, inc. (2002) SpeedTree. [Computer Program].
Available at: https://store.speedtree.com/
[Accessed 5 December 2022]

GitHub, Inc. (2008) GitHub [computer program].
Available from: https://github.com/
[Accessed 4 April 2023].

Axosoft, LLC. (2014) GitKraken [computer program]. Available from:
https://www.gitkraken.com/
[Accessed 4 April 2023].

Autodesk. (1998) Maya [computer program].
Available from:
https://www.autodesk.com/products/maya/
[Accessed 4 April 2023].

Blender Foundation. (1994) Blender [computer program]. Available from:
https://www.blender.org
[Accessed 4 April 2023].

Nielsen, J. (2020). 10 Heuristics for User Interface Design. [online] Nielsen Norman Group.
Available at:
https://www.nngroup.com/articles/ten-usability-heuristics/
[Accessed 1 April 2023].

Nielsen, J. and Molich, R. (1990). Heuristic evaluation of user interfaces. Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90. doi:https://doi.org/10.1145/97243.97281 [Accessed 1 April 2023].

Nielsen, J. (1994). Usability inspection methods. Available at: https://rauterberg.employee.id.tue.nl/lecturenotes/0H420/Nielsen%5B1994%5D.pdf [Accessed 1 April 2023].

Forsyth, D.R. (2008). Self-Serving Bias. In International Encyclopedia of the Social Sciences. Retrieved from https://scholarship.richmond.edu/cgi/viewcontent.cgi?article=1164&context=jepson-faculty-publications [Accessed 14 April 2023].

Moonlighter Wiki. (2018). Mini-Map. [online] Available at: https://moonlighter.fandom.com/wiki/Mini-Map [Accessed 20 Apr. 2023].

Boris (2019). Dungeon Generation in Enter The Gungeon. [online] BorisTheBrave.com. Available at: https://www.boristhebrave.com/2019/07/28/dungeon-generation-in-enter-the-gungeon/. [Accessed 1 April 2023].

Debenham, E. (2017). What's coming in Shattered Pixel Dungeon v0.6.0 pt.1. [online] Shattered Pixel. Available at: https://shatteredpixel.com/blog/whats-coming-in-shattered-pixel-dungeon-v060.html [Accessed 20 Apr. 2023].

Code Respawn. (2022). Dungeon Architect [Software]. Unity Asset Store. https://assetstore.unity.com/packages/tools/utilities/dungeon-architect-53895 [Accessed 5 April 2023].

Game Guides. (2016). Level 1: Into The Dark - Legend of Grimrock Game Guide & Walkthrough | gamepressure.com. [online] Available at: https://guides.gamepressure.com/legendofgrimrock/guide.asp?ID=14823 [Accessed 20 Apr. 2023].

Nappin. (2022). Roguelike Generator Pro - Level & Dungeon Procedural Generator [Software]. Unity Asset Store. https://assetstore.unity.com/packages/tools/level-design/roguelike-generator-pro-level-dungeon-procedural-generator-224345 [Accessed 5 April 2023].

Hart, P., Nilsson, N. and Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, [online] 4(2), pp.100–107. doi:https://doi.org/10.1109/tssc.1968.300136. [Accessed 5 April 2023].

Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), pp.269–271. doi:https://doi.org/10.1007/bf01386390. [Accessed 4 April 2023].

Bridson, R. (2007). Fast Poisson disk sampling in arbitrary dimensions. ACM SIGGRAPH 2007 Sketches. Available at: https://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf [Accessed 14 April 2023].

von Neumann, J. (1966). Theory of self-reproducing automata. University of Illinois Press. Available at: https://cba.mit.edu/events/03.11.ASE/docs/VonNeumann.pdf [Accessed 14 April 2023].

Perlin, K. (1985). An image synthesizer. ACM SIGGRAPH Computer Graphics, 19(3), 287-296. Available at: https://dl.acm.org/doi/pdf/10.1145/325165.325247 [Accessed 14 April 2023].

Fournier, A., Fussell, D. and Carpenter, L. (1982). Computer rendering of stochastic models. Communications of the ACM, 25(6), pp.371–384. doi:https://doi.org/10.1145/358523.358553. [Accessed 14 April 2023].

Witten, T.A. and Sander, L.M. (1981). Diffusion-Limited Aggregation, a Kinetic Critical Phenomenon. Physical Review Letters, 47(19), pp.1400–1403. doi:https://doi.org/10.1103/physrevlett.47.1400. [Accessed 14 April 2023].

Prim, R.C. (1957). Shortest Connection Networks And Some Generalizations. Bell System Technical Journal, 36(6), pp.1389–1401. doi:https://doi.org/10.1002/j.1538-7305.1957.tb01515.x. [Accessed 14 April 2023].

Lindenmayer, A. (1968). Mathematical models for cellular interaction in development. Journal of Theoretical Biology, 18(3), 280-299. Available at: https://doi.org/10.1016/0022-5193(68)90079-9 [Accessed 14 April 2023].

Lorensen, W.E. and Cline, H.E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. ACM SIGGRAPH Computer Graphics, 21(4), pp.163–169. doi:https://doi.org/10.1145/37402.37422.

Commented [AB(1]: Self note: to check

Commented [AB(2]: Self note: to check

[Accessed 14 April 2023].

12. Ludology

Mossmouth (2013) Spelunky. [Video game]. Mossmouth. Available at: https://store.steampowered.com/app/239350/Spelunky/.
[Accessed 6 December 2022]

Dice (2011) Battlefield 3. [Video game]. Electronic Arts. Available at: https://store.steampowered.com/app/1238820/Battlefield_3/.
[Accessed 6 December 2022]

Nincalis, Inc., Edmund McMillen (2014) The Binding Of Isaac Rebirth. [Video game]. Nicalis, Inc. Available at: https://store.steampowered.com/app/250900/The_Binding_of_Isaac_Rebirth/.
[Accessed 6 December 2022]

Dodge Roll (2016) Enter the Gungeon. [Video game]. Devolver Digital. Available at: https://store.steampowered.com/app/311690/Enter_the_Gungeon/. [Accessed 6 December 2022]

Brace Yourself Games, 2015. Crypt of the NecroDancer. [video game] Brace Yourself Games. Available from: https://braceyourselfgames.com/crypt-of-the-necrodancer/
[Accessed 22 March 2023].

Epyx, Inc. (1985) Rogue. [Video game]. Pixel Games UK. Available from: https://store.steampowered.com/app/1443430/Rogue/
[Accessed 20 April 2023].

Matt Dabrowski (2019) Streets of Rouge. [Video Game]. tinyBuild. Available From: https://store.steampowered.com/app/512900/Streets_of_Rogue/
[Accessed 22 March 2023].

Appendix A: Project plan

Date of the week	Tasks Done/started	Brief summary of outcomes achieved, research or practical aspect completed	Questions arising and/or tasks to be taken forward
04/10/2022	• Opened and started the repo • Started logbook • Started Proposal document	Started the github repo for version control Started to gather sources for the proposal	
31/10/2022	• Added Drunk walk • Movement in inspector • L_system added • Basic tile set to help with testing • Voronoi algo added	Added some of the most basic algorithms that were listed in the proposal Implement a tile set which will be used to simulate an actual level	
02/10/2022	• First very random room/level maker • A* pathfinding	Made the first dungeon generator using a randomiser built by unity	
03/10/2022	• Cellular automata added • 3D drunk walk • Delaunay triangulation	Added more of the basic algorithms	3D drunk walk will should build on the 3D Perlin worms
04/10/2022	• Started wave function collapse	Started to work on the wave function collapse	
07/10/2022	• Find walls algo to signal outer walls	Algorithm to find the outer tile and tag that tile as a wall	
08/10/2022	• Perlin noise in 2D and 3D • Start of fixing the whole codebase and joining fix together	Started to tidy up the codebase and form the first few mixes	

10/10/2022	• Decided on the UI method to implement in the game	Decided on the in-Game system UI will be dealt with a state machine	
13/10/2022	• More Perlin noise features • Perlin noise worm started • Error message added in case user inputs something not valid	Added an error message pop up and worked on more Perlin noise fixes	Is there a possibility to add a pop up in editor run time?
19/10/2022	• Started research report	Started to gather sources for the	How can the artefact be an ethical problem?
10/12/2022	• Final draft of the report was finished	The final draft of the research report was done, all that is left is to fix the references	
15/12/2022	• Added diamond square algorithm	Implemented diamond square algo to have a variant of Perlin noise	Although seems to work it doesn't form true sections which are needed for the room generation, will need more looking at
20/12/2022	• Added Binary partition system (BPS) • Added prim's algo	BPS will be used to create rooms in a tidier manner but keeping the randomization Prim's algorithm will allow a singular path to be fetched from a triangulation, will be used to create the corridor	
27/12/2022	• Added the first iteration of the GraphView • Remade the WFC algorithm due to previous issues	The player can create the rules in a graph style inspector window.	The GraphView library API provided from Unity is in an experimental stage, whilst development there have been some issued due to the above-mentioned status.
28/12/2022	• Standardisation of classes	With Every algorithm using similar structure now I can mix and match algorithms	
1/01/2023	• Rewriting of most algorithms for better work performance	All the algorithms are now able to be called from a central script	With the centralization of the algorithms when making the UI it should be a simple one-line function call for better organisation
3/01/2023	• Different rulesets have been added for their respective algorithms	There are now objects where the player can store its rules or tiles to be called from the algorithms	
8/01/2023	• The user can now generate the dungeon using its own tile sets	The user is given a choice of what wall tiles and floor tiles to be used when generating the dungeon	When generating the dungeon there is a chance that more than 30000 objects are used, this really kills performance and there will need to be a way to create chunks to load in and out for maximum performance
11/01/2023	• Added the main algorithm generation component to the top toolbar of the unity	The user can now spawn the generator using the tool bar at the top	

	editor		
13/01/2023	• Added a new node for the WFC • Added tooltips to the UI in the inspector	This new node greatly increases the workflow by being only one node instead of a web of nodes Every UI element now has a tooltip that explains in detail what that button does and its effect	There is yet no way to quickly spawn the nodes
17/01/2023	• Perlin worms algorithms added	Last of the required algorithm has been added.	The algorithm although working fine is not what was expected and only "works" in certain scenarios therefore extensive testing will need to be done
23/01/2023	• Bezier path added • Dynamic sized corridors • Recorder the first draft video for the prototype hand in	Now the user can generate a corridor with a curving direction The user can now decide the size of the corridor, this was originally done due to marching cubes algorithms only working with corridors which are 3 floor tiles wide	There are cases where the ending of the Bezier doesn't reach the ending due to how the function works.
26/01/2023	• First iteration of a functional UI in the inspector for one of the algorithms	The Random Walk algorithm now can be fully used to its full potential with all the possible generation choice being neatly laid out.	Should try to have the UI standardised and be under the call of a function to make the modifying aspect much easier in the future
29/01/2023	• Room to room generation	The user can now generate a random using a room-to-room technique	The room-to-room technique was the last of the needed types of dungeons.
31/01/2023	• Voronoi room to room generation • Undo button	More choices of how to generate the dungeon have been added An Undo button has been added to allow the player to return 1 step behind, to increase ease of use	
17/02/2023	• Added the ability to create rooms that are not PCG • Tidied up the project ready to become an asset • Chunk system added so only the chunks where the player is get rendered once the game starts	The player can now create rooms that are not PCG but will be placed randomly around the map, this is to allow a more familiar feel to the player whilst still achieving the usual PCG scale The player can now drag any entity into a list, and it will dynamically only render in those chunks to save on performance instead of drawing the whole dungeon.	
18/02/2023	• Fixed many bugs • Improved performance • Allow the player to create a canvas where the generation can happen to any size desired • Added dead end corridor • Dynamic amount of undo's • Refactoring of the code	Bezier curves didn't finish in the right spot Flood fill algorithm gave "out of stack error", now fixed Before the was a cap on the size of the canvas due to some limitation but now it's been lifted	

	• New UI • The player can now save the result of the generated dungeon	Corridors that lead to nowhere to increase the gameplay design options I can now set how many undo the user gets The new UI aims to be more streamline as it lowers the number of things on the screen to. After the Player has finished creating the template of his dungeon, before generating it he is asked if he wants to save to file for future use, this is useful to give the user a way to save its favourite dungeons with no worry of losing them.	
3/03/23	• Added Perlin worms algorithm • Added the ability for the player to decide the chunks to draw • Started the wiki/documentation	Perlin worms was one of the main algorithms that until now was missing but has now been successfully added. The player when starting the game can now decide on the amount of the map to show using a built-in chunk system for max performance. The wiki on the Github has been started so the user knows what to do and what to expect from the project once it will be made into an asset.	
6/03/23	• Reformatting ready to be made into a library • L-system algorithms additional changes • Fixed a lot of bugs	A lot of formatting was done to the code to help to achieve the most functional, effective, and intuitive. L-system now has an integrated macro system where the player can implement the generation of rooms into the algorithm instead of building an afterthought, this was done to try and keep most of the algorithms generate different stuff from each other, giving the player more choice. Some major UI bugs, roadblocks and performance uplifts were made to improve the experience.	
10/03/23	• Asset pack created • More bug fixes	The final goal of the project has been achieved, the user can now instal the asset pack via the asset pack menu to be used in any of its desired projects.	For the asset pack I have had to create a new git hub repo where it is hosted, and I will need to investigate how to properly push to release new versions correctly
12/03/23	• Wiki structure started	In the repo with the asset the user can now go into the wiki and read the documentation about how to use the asset and much more info.	
20/03/23	• Fixed the Poissant issue	The Poissant algorithm is used to	

	• Started the report	help the user populate the dungeon created some issues and usability things where fixed	
24/03/23	• Added the ability to self-edit the outcome of the dungeon	The user can now self-editor and change specific tiles on the grid to suit its gameplay more, like opening certain paths or closing them	The UI to use the editing tool is not the best due to the limitations of the Editor itself
05/04/23	• Changed some of the tooltips and order of UI elements to give the player more context • Implemented multithreading	Additional tooltips have been added to the whole project to help the player understand what each element does. Massive improvement in performance for some algorithm has achieved by parallelizing he loops between multiple thread.	Using multiple threads to achieve tasks could lead to thread issues where some tasks are done before others, therefore a lot of testing needs to be done to make sure no issue like that pop up
18/04/23	• Added more tooltips to explain more mechanics • Took out the debris generation for the vertices generation method	The vertices generation of the dungeon does not create chunks it is meant to be a way for the user to export the generated dungeon to the wanted software. The debris need the chunk system to function efficiently.	

Appendix B: Project Timeline

October	Final proposal to be submitted by (25 oct)	
November	Start research. Implement the basis of all 5 algorithms in 2D. Start to test with some mixes of algorithms to create the first maps.	4 days 1 day 17 days 7 days
December	Finalise research report. Implement Cellular automata on the rooms themselves.	12 days 3 days
January	Implement more intricate maps and mixes of algorithms, start to add more levels to the maps. Implement the basis of the UI for the User. Assessed presentation (23 jan).	10 days 2 days
February	Implement Optimisations. Start final report.	5 days 10 days
March	Continue with the report.	30 days
April	Hand-in (24 April).	24 days
May	Viva.	

Appendix C: Assets used in the Project

Unity Technologies, [2023]. [FBX Exporter]. Available at:
https://docs.unity3d.com/Packages/com.unity.formats.fbx@2.0/manual/index.html
[Accessed 20 April 2023].

Appendix D: Algorithms Terminology and Explanation

A* Pathfinding: A search algorithm used to find the shortest path between two points on a grid or graph, taking into account obstacles and movement costs. It employs a heuristic function to estimate the remaining distance to the goal, allowing it to focus on the most promising paths and avoid unnecessary exploration (Hart, Nilsson & Raphael, 1968).

Dijkstra's Algorithm: A graph search algorithm that finds the shortest path between a starting node and all other nodes in a weighted graph. It iteratively selects the unvisited node with the smallest known distance from the starting node and updates the distances to its neighbors (Dijkstra, 1959).

Binary Space Partitioning: A method of recursively dividing a space into smaller, non-overlapping sections to create a tree-like data structure. It is often used in level design for organizing scene elements, optimizing rendering, and generating procedural layouts (Fuchs, Kedem & Naylor, 1980).

Poisson Disk Sampling: A sampling method that generates points uniformly distributed across a surface while maintaining a minimum distance between them. This technique helps create natural-looking distributions of objects, such as trees, rocks, or stars (Bridson, 2007).

Bezier Curves: Parametric curves used to create smooth and controllable paths or shapes, often employed in graphic design, animation, and pathfinding. They are defined by a set of control points that influence the curve's shape without necessarily lying on the curve itself.

Voronoi Diagrams: A partitioning of a plane into regions based on the distance to a set of input points. Each region, called a Voronoi cell, contains one input point and consists of all points closer to that point than to any other input point. Voronoi diagrams are used in various applications, including spatial analysis, procedural generation, and computational geometry.

Cellular Automata: A computational model that simulates the behavior of cells on a grid, with each cell's state determined by the states of its neighbors according to predefined rules. Cellular automata can model complex, emergent behavior and are used in various applications, such as procedural generation, simulation, and pattern recognition (von Neumann, 1966).

Perlin Noise: A gradient noise function used to generate coherent noise patterns, often employed in procedural texture and terrain generation. Perlin noise produces smooth, natural-looking variations, making it suitable for creating realistic landscapes, clouds, and other organic structures (Perlin, 1985).

Diamond-Square Algorithm: A fractal-based terrain generation algorithm that uses a midpoint displacement technique to create realistic heightmaps. It operates by recursively subdividing a square grid into smaller squares and perturbing their heights based on a random offset (Fournier, Fussell & Carpenter, 1982).

Perlin Worms: A method of generating organic-looking tunnels or caves using Perlin noise to influence the direction of a random walk. This technique creates smoothly curving paths that can resemble the burrows of worms or other natural features (Perlin, 1985).

Diffusion-Limited Aggregation: A simulation of particle growth based on the diffusion and aggregation of particles. Particles are released randomly and move through a medium until they encounter a stationary structure, at which point they aggregate. This technique is used for procedural generation of organic structures, such as coral, snowflakes, or fractal patterns (Witten & Sander, 1981).

Wave Function Collapse: A constraint-based procedural generation algorithm that collapses a set of possible patterns into a single output based on input constraints. The algorithm iteratively observes and collapses the least constrained cell, generating coherent structures that satisfy the input constraints.

Random Walk: A stochastic process that generates a path by taking random steps in any direction from the current position, often used in procedural terrain and level generation. The random walk can create organic, unpredictable paths and structures, making it suitable for generating caves, mazes, or other irregular features.

Delaunay Triangulation: A technique for creating a set of non-overlapping triangles from a set of input points, with the property that no input point is inside the circumcircle of any triangle. Delaunay triangulations maximize the minimum angle of all the triangles, producing well-shaped, non-skewed triangles. They are used in various applications, such as mesh generation, interpolation, and computational geometry.

Prim's Algorithm: A greedy algorithm for finding the minimum spanning tree of a connected, undirected graph. It starts with an arbitrary vertex and iteratively adds the edge with the smallest weight that connects a visited vertex to an unvisited vertex, often used for generating mazes and optimizing network connectivity (Prim, 1957).

Flood Fill: An algorithm for filling a connected region in a grid with a specified value. It starts from a seed point and recursively or iteratively replaces the neighboring cells with the target value if they match the original value, often used in image processing, level generation, and pathfinding (Moore, 1959).

L-Systems: A grammar based system used to model the growth of plants and other fractal structures through the recursive application of production rules. An L-system consists of an initial axiom and a set of production rules that transform strings of symbols, often used in procedural modeling, computer graphics, and biological modeling (Lindenmayer, 1968).

Marching Cubes: A surface extraction algorithm for creating polygonal representations of isosurfaces in 3D scalar fields, such as density or elevation data. It operates by examining each cube of the field and determining the intersections of the isosurface with the cube edges, then constructing triangles to approximate the surface, often used in terrain and model generation, medical imaging, and scientific visualization (Lorensen & Cline, 1987).

Appendix E: Documentation/Wiki

The Dungeon Forge project includes a comprehensive Wiki, which serves as a valuable resource for users to understand the tool and its functionalities. The Wiki was created to provide a clear and concise guide on how to use the tool, its various features, and the underlying algorithms and techniques used in its development. By making this information easily accessible, users can quickly familiarize themselves with the project and potentially contribute to its improvement. The Wiki can be found on the project's GitHub page at the following link: https://github.com/AlessandroBufalino3115/Dungeon-Forge/wiki