Figure 1: Image of a Dungeon from a novel in Mabinogi (Nexon et al., 2017)

An Investigation into Methods for Procedural Dungeon Generation for Roguelike Games

INDIVIDUAL RESEARCH PROJECT

Lewis Hammond | S1800707 | Computer Games Programming

Contents

Introduction

The focus of this paper is to investigate techniques that can be used to procedurally generate dungeon levels for roguelike games. This paper will assess techniques to generate the structure of the level, furnish the level with rewards and enemies and enhance the level generation process. A rounded view of the techniques investigated will be presented, to assess their strengths and weaknesses. Ultimately, as a result of evaluating the current research, a combination of techniques will be put forward to be implemented in the Unity (Unity Software, 2022) engine.

Figure 2: Procedurally Generated terrain from Minecraft (Mojang Studios, 2022)

Procedural generation is integral to Roguelike games. Procedural generation is the process of generating content with algorithms (Togelius et al., 2011.) rather than requiring human artists to create every asset or level within the game. This technique allows games to have play spaces generated automatically, creating games that would be impossible without procedural generation, such as roguelikes. Another such example is Minecraft (Mojang Studios, 2022), as seen in Figure 2.

The application of procedural generation can offer advantages over human-made levels. Levels can be generated faster and with a greater diversity – reducing the time that designers need to spend within the game development process (Togelius, Shaker and J. Nelson, 2016; van der

Linden, Lopes and Bidarra, 2014). Greater diversity can also increase game replayability (van der Linden, Lopes and Bidarra, 2014) as each level generated can be unique.

Finally, procedural generation is a growing trend within the video game industry. In Horizon Zero Dawn (Guerrilla Games, 2017) almost all the game's vegetation is procedurally placed, as well as the pickups (van Muijden, 2017). Independent developers such as 17-BIT (2015) have also embraced procedural generation techniques, as in GALAK-Z (17-BIT, 2015) all levels are procedurally generated.

AIMS & OBJECTIVES

Aims

- Conduct a review of the relevant literature surrounding procedural dungeon generation
- Research and critically evaluate techniques for level space generation, level furnishing and supplemental techniques, for roguelike games.

Objectives

- Evaluate industry literature to establish the most appropriate techniques for level generation and level furnishing
- Establish additional techniques to support space generation techniques and their appropriate use
- Create an output design of a technique or combination of techniques to generate a dungeon appropriate for a roguelike game.

Literature Review

CONTEXT

Defining Roguelikes

The term Roguelike originates from the video game Rogue (Toy *et al*, *1980*). The exact definition of Roguelike games is disputed, one interpretation is that the games must be as close to Rogue (Toy *et al*, *1980*) in every way, however, this is not representative of modern roguelikes (Zapata, 2017).

The Berlin Interpretation defines several principles that should be followed. It defines that all roguelikes should be turn-based, have randomly generated environments, have a permanent failure state, and not follow a linear path (Hatfield, 2012). However, these principles were determined by the International Roguelike Development Conference 2008 which was only attended by a small group of participants (Zapata, 2017). Additionally, this does not reflect the state of modern roguelike games, such as The Binding of Isaac (McMillen and Himsl, 2011) and Enter the Gungeon (Dodge Roll, 2016).

Bycer (2021) explains that the core elements for modern rougelikes are replayability and variance. It is because of these two factors that procedural generation is central to modern roguelikes. It is important to develop generation algorithms that allow game designers to control to process of generation because balance and difficulty in roguelikes is tightly linked to the generated elements of the game (Bycer, 2021). It is with this definition that this paper will assess the relevant generation algorithms.

Defining Procedural Generation

Procedural Generation, sometimes referred to as Procedural Content Generation (PCG), is defined, within games, as the generation of game content using algorithms (Togelius et al., 2011.). This can include offline generation where content is generated by the game developer and used within the game, often with human modification, and online generation where the content is generated as the game is played (Togelius, Shaker and J. Nelson, 2016). This is the technique used in roguelike games.

PCG has been used in many games to generate the game's environment such as Elite (Braben and Bell, 1984), Minecraft (Mojang Studios, 2022) and Spelunky (Mossmouth, 2013).

This study will look at how dungeons are procedurally generated in video games. As defined by van der Linden, Lopes and Bidarra (2014) dungeons are "labyrinthic environments, consisting mostly of inter-related challenges, rewards and puzzles, tightly paced in time and space to offer highly structured gameplay".

SPACE GENERATION ALGORITHMS

Cellular Automata

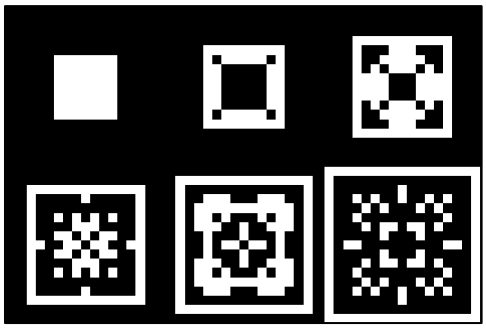

Figure 3: An example of 6 frames of a Cellular Automata evolution, starting from a central square. Generated with Visions of Chaos (Rampe, 2000)

Cellular Automata (CA) is a computational model that is widely used in computer science, such as in modelling growth in vegetation (P. A. Macedo and Chaimowicz, 2017; Shaker et al., 2016). An example is shown in Figure 3. Cellular Automata was popularized by Conway's Game of Life (Gardner, 1970). The model has since been applied to a variety of different problems such as simulating physics (Vichniac, 1984) and modelling road traffic (Maerivoet and De Moor, 2005). More recently, Cellular Automata has been used for the generation of game levels (Johnson, N. Yannakakis and Togelius, 2010; Ziegler and von Mammen, 2020).

The model consists of a grid in an *n*-dimensional grid, a set of states and a set of rules where each cell can only be in one state at any one time, for example on or off (Shaker et al., 2016). Rules then determine how the simulation progresses; these rules are generally based on the states of neighboring cells.

Simulations such as the Game of Life (Gardner, 1970) require the cells of the grid to be more dynamic, in such a way that the cells are constantly changing. These implementations of CA are known as Gliders (Sapin, Bull and Adamatzky, 2007). These structures are not helpful in generating game levels because we want to have a stable set of cells to form a level. Aikman (2015) approaches this problem by creating a set of rules that moves towards equilibrium and creating a dense grid of cells.

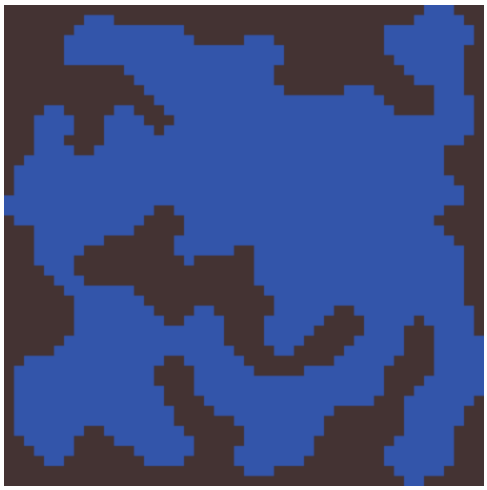

Figure 4: An Image of a 2D cave system generated with Cellular Automata (Cook, 2013)

Cellular Automata as a level generation technique is presented by Shaker et al (2016) as a very flexible technique. Furthermore, CA is praised by many studies for generating natural, organic, cave-like levels, as seen in Figure 4 (Aikman, 2015; Cerny Green et al., 2019; Shaker et al., 2016; van der Linden, Lopes and Bidarra, 2014; Wolverson, 2020). Particularly, Johnson, N. Yannakakis and Togelius (2010) who state that CA can create levels that appear to be good with a very low computational overhead, however this study only compares a CA generated map with a completely randomly generated map, rather than another algorithmic technique, and does not take in to account any gameplay considerations.

While CA can generate realistic looking cave levels in a fast efficient way, taking 1.4×10^{-4} milliseconds to generate a 50 by 50 cell room (Johnson, N. Yannakakis and Togelius, +2010). These levels often have playability or design issues. One such issue is that CA can generate areas of the map that the player is unable to reach (Cerny Green et al., 2019). To remove these unreachable

areas additional algorithms are required to run after the level has been generated, which reduces the efficiency of a level generation process.

Aikman (2015) claims that CA is not an appropriate method of generation for an entire level as it is difficult to place gameplay elements, such as treasure and enemy patrol paths, within the level. However, Cerny Green *et al* (2019) argue that a different implementation of CA, where cells are visited randomly with variable neighbourhood size as well as restricting the number of items of a specific type, can be used to furnish the level.

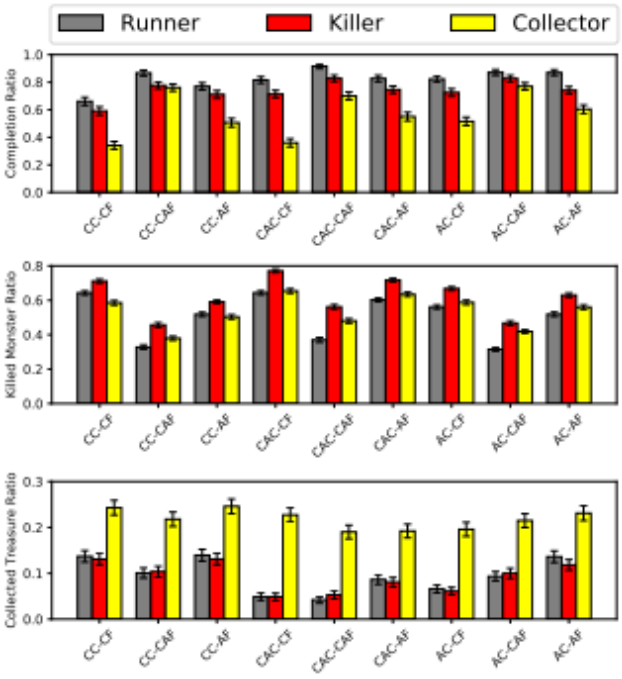

Figure 5: An evaluation of combinations of different level layout creators and level furnishers (Cerny Green et al., 2019, p. 6, fig. 4).

CC - Constraint based Creator, CAC - Cellular Automata based Creator, AC - Agent based Creator.
CF - Constraint based Furnisher, CAF - Cellular Automata based Furnisher, AF – Agent based Furnisher.

From the results summarized in Figure 5, where levels furnished by CA were tested by artificial agents, it was found that fewer enemies were killed when compared with other level furnishing methods. While this appears to support Aikman's (2015) thesis, these levels had high completion levels across the 3 tested player types.

Additionally, Aikman (2015) evaluates the use of CA for entire level generation for the game GALAK-Z (17-BIT, 2015) and thus was assessing CA's use in a very small context. GALAK-Z (17-BIT, 2015) also appears to market itself towards the "killer" player type, defined by the Green *et al* (2019) study as an agent that prioritizes killing monsters. This may have further decreased Aikman's (2015) confidence in the technique. Contrastingly Green *et al* (2019) were evaluating CA in a broader context than Aikman (2015) and saw an application for CA furnishing. While Aikman's (2015) concerns should not be disregarded, especially around areas not covered by Green *et al*'s (2019) study such as enemy path generation, Green *et al* (2019) prove that it is possible to have furnished levels generated with CA.

Finally, several design issues are created when using CA for level generation. While Johnson, N. Yannakakis and Togelius's (2010) approach for generating cave levels appears to be intuative, it is difficult to completely understand the impact of changing each parameter (Shaker et al., 2016; Aikman, 2015). This means that it is hard for designers to prescribe a specific layout or number of rooms for a dungeon. It also makes it hard to prescribe a difficulty for a given level or section within the design of a level. It is argued by van der Linden, Lopes and Bidarra (2014) that PCG techniques are unpopular within the industry because designers feel that they do not have control over the generation process. The combination of these two factors means that CA may be an inappropriate method for level generation for games that need more subtle control, such as roguelikes.

Binary Space Partitioning

Binary Space Partitioning (BSP) is a spatial partitioning algorithm. Spatial partitioning is the process of subdividing a 3D or 2D space into non-overlapping subsets, sometimes referred to as 'cells', such that any point in the space can only be in one subset (Shaker et al., 2016). BSP follows these principles by dividing space into two subsets recursively until the subset is of the desired size (Wolverson, 2020). As well as the generation of dungeons BSP has been used for image compression (Radha, Vetterli and Leonardi, 1996) and generating forests (Fan, Li and SIsson, 2019).

Firstly, unlike Cellular Automata, BSP creates a very well-structured appearance of a dungeon that has a set of rooms that are well spaced (Shaker et al., 2016; Wolverson, 2020). This allows the

creation of levels that feel more constructed, rather than a natural cave structure. This aligns with Shaker *et al*'s (2016) definition of a dungeon as a labyrinthic environment.

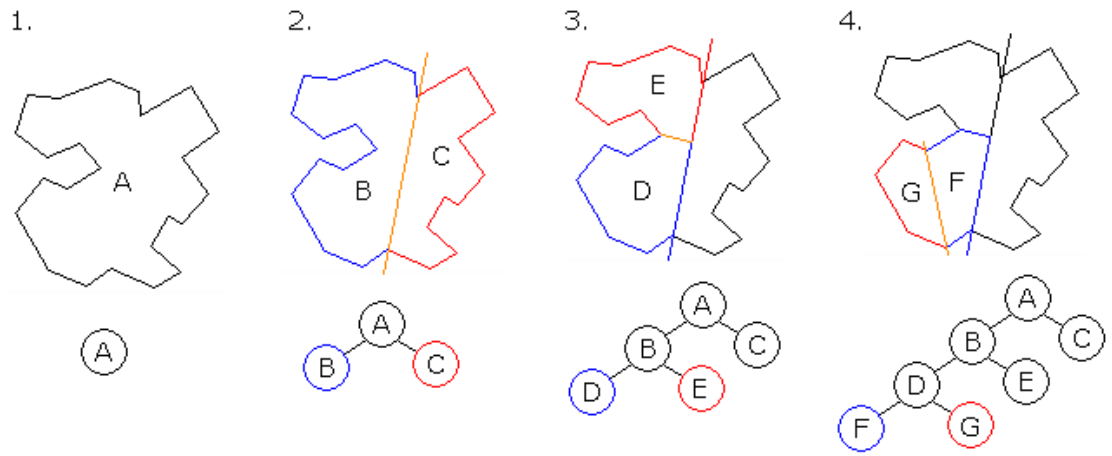

Figure 6: An example of a Binary Space Partition (above) with a Binary Space Partition Tree (below) (Fredrik, 2004)

This structured approach is extended as the method of generating rooms allows the creation of a binary tree, known as a Binary Space Partitioning Tree (BSP Tree) (Shaker et al., 2016). An example space with a BSP Tree is shown in Figure 6. BSP Trees allow the dungeon to be split into different sub-sections, which may have different themes, enemies, or loot types. This is usually accomplished by Voronoi Diagrams, which is a common approach to dividing space into regions (Santamaria-Ibirika et al., 2014), a method that is supported by Wolverson (2020).

BSP graphs allow a better approach for dividing the level into sections than Voronoi diagrams. Using BSP graphs is more efficient than using Voronoi diagrams in this instance because they can be implicitly generated in the BSP room generation process, rather than requiring an additional step to be run. Additionally, because the BSP graph is generated as part of the level generation process it means that entire rooms will be neatly divided into sections, rather than with Voronoi where rooms may be split into 2 sections, or a section could contain zero rooms.

Aikman (2015) described BSP for dungeon generation as a "tried and true" process. However, did not choose this for the level generation for the game Galak-Z (17-BIT, 2015). This was because the method had issues when trying to connect rooms. Aikman (2015) states that the problems created

by this method involve creating corridors that are of undesirable angles and connect to rooms in undesirable ways. However, this issue may be directly related to the design of Galak-Z (17-BIT, 2015) as the gameplay trailer (PlayStation Europe, 2016) shows that the game has movement that requires large open spaces, something which is better suited to a CA rather than BSP.

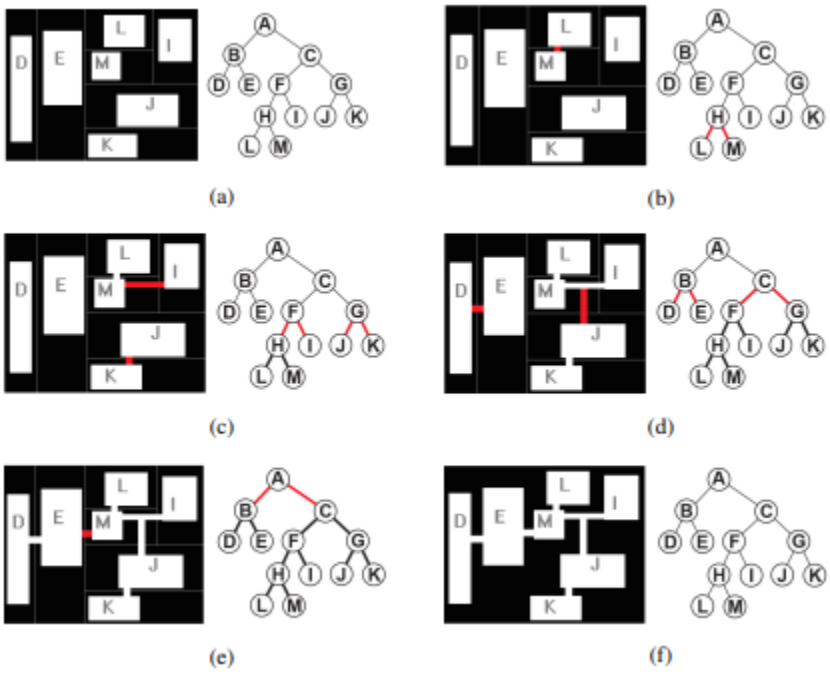

Figure 7: Diagram demonstrating the process developed for connecting corridors using BSP Graphs (Shaker et al. 2016, p. 37, fig. 3.4)

For more traditional roguelike games, Shaker *et al* (2016) suggest an approach that allows the BSP Tree to generate corridors between rooms, this is shown in Figure 7. The process is achieved by randomly choosing the bottom nodes of the graph and connecting them to the parent. The nodes are then considered connected at the parent node (labelled 'H' in Figure 7). It then moves up the tree to connect to other rooms to the newly formed connected pair, as well as any nodes that are not yet connected to a parent. This process is repeated until the entire dungeon is connected.

This approach is supported by the work of Baron (2017) who remarks that this method creates very orderly and linear paths. These paths should create a logical route for players to follow, meaning that a dungeon using BSP for its corridors should create layouts that are easy to navigate.

Finally, Whitehead (2020) has critiqued the use of BSP, claiming that it is difficult to design elaborate room arrangements with BSP since the algorithm requires that all rooms be fully contained within cells: preventing cells from overlapping a cell boundary. This can be mitigated by allowing rooms to cross cell boundaries under certain conditions, however this requires tweaking the technique. This is touched on by Shaker *et al* (2016) who states that the appearance of dungeons is closely linked with how closely the principles of the algorithm are followed, however they do not state how specific alterations will influence dungeon design, aside from leaving some BSP nodes empty which does not address the concerns of Whitehead (2020).

In conclusion, Binary Space Partitioning is an effective method that can be used to create well-structured dungeons, that have the appearance of structures that have been purposefully created, both in the design of how rooms are created and their connecting corridors. As a direct effect of the constructed look, the approach has come under criticism from industry experts who state that the approach creates dungeons that appear too constructed and cannot support complex arrangements. While this does not disqualify BSP as a valid dungeon generation technique it does present problems that require significant steps to mitigate.

ENTITY GENERATION

A problem that has been identified within this paper is the issue of generating the entities of the game world. Generally, these entities include loot chests, where players are rewarded with items, and enemies, that the player must fight to progress. In this section, the paper will analyze the use of prefabs and spawners as well as an agent-based approach.

Prefabs and Spawners

Figure 8: A screenshot from Unite 2014 - Dungeon of the Endless Rendering and Procedural Content (Dubois and Amplitude Studios, 2014) showing a room template.

Blue squares are locations (slots) where props and be spawned.

Dubois (2014) suggests an approach that leads to a more controllable generation of level entities. This approach involves creating pre-made rooms that contain several locations where props and other items can be spawned, as shown in Figure 8. Using this method allows greater control over the design of the level, artistic composition and made the system scalable to different room sizes, at the cost of more recognizable patterns in rooms (Dubois, 2014). However, this requires designers to create the rooms themselves. The spawning of enemies is not covered by this technique, meaning that this exact implementation cannot be used to generate a complete dungeon as roguelike games almost always require enemies to be spawned in the level.

This concept can be expanded to allow greater flexibility of the entities spawned and the ruleset used to determine when entities should be spawned. A method prescribed by Pittman (2015) involves having spawners that have a random chance, or a weighed random chance, to spawn an entity. This is then combined with an overseer AI that limits the population of enemies and rare items, as well as ensuring that enemies are not spawned in undesirable locations (such as next to the players' spawn point). The application of this method resolves some of the issues in Dubois' (2014) work by allowing greater control over the difficulty of a given level and allowing the spawning of enemies.

Agent-Based

As the previous method is specific to the game that the developers are creating, they might have a more narrowed outlook on the approach. Cerny Green et al. (2019) present a more academic view of the problems. As well as the previously analyzed Cellular Automata-based furnisher, Cerny Green et al. (2019) demonstrate an agent-based approach to furnishing a room. This involves having each entity in the dungeon move according to a set of rules, as defined in Table 1, over 45 iterations.

Entrance/Exit	Move as far apart from each other as possible
Treasure Chests	Moves closer to goblins in Line-Of-Sight (LOS)
Potions	Move randomly around the map
Portals	Move as far apart from each other, the Entrance, and the Exit
Traps	Move away from other traps and goblins in LOS. Move towards treasure in LOS
Goblins	Move away from other goblins in LOS
Goblin Mages	Move toward goblins in LOS and away from other Goblin-Mages within LOS
Ogres	Move away from Ogres in LOS (within 6 tiles), and move within 4 tiles of Treasure in LOS
Blobs	Move within 4 tiles of other Blobs and Potions in LOS
Minitaur	Move as far away as possible from the Entrance and Exit

Table 1: A table showing the rules for an Agent-based furnisher approach (Cerny Green et al, 2019, p. 4, table 4)

This method generated levels that have a long distance between the start and end points but tended to create dungeons where most of the entities are concentrated towards the start of the level. This can create playability problems as the player may be overwhelmed by enemies or items at the start of the level and then feel that the end of the level is empty and too easy.

SUPPLEMENTAL ALGORITHMS

Several supplemental algorithms can be combined with other PCG techniques to enhance the aesthetics and gameplay quality of the generated level. This section of the paper identifies two such algorithms.

Diffusion Limited Aggregation

The first is Diffusion Limited Aggregation (DLA). DLA is a method developed by Witten and Sander (1981) that involves free particles moving around a space. A fixed central particle is chosen as the start of the structure, when a free moving particle contacts any particle attached to the structure (shown in red in Figure) it is then attached to the structure. This process is then repeated until all particles are attached to the structure. DLA has many applications outside of video games, such as modelling river basin evolution (Wang et al., 2020) and the generation of frost (Liu et al., 2010).

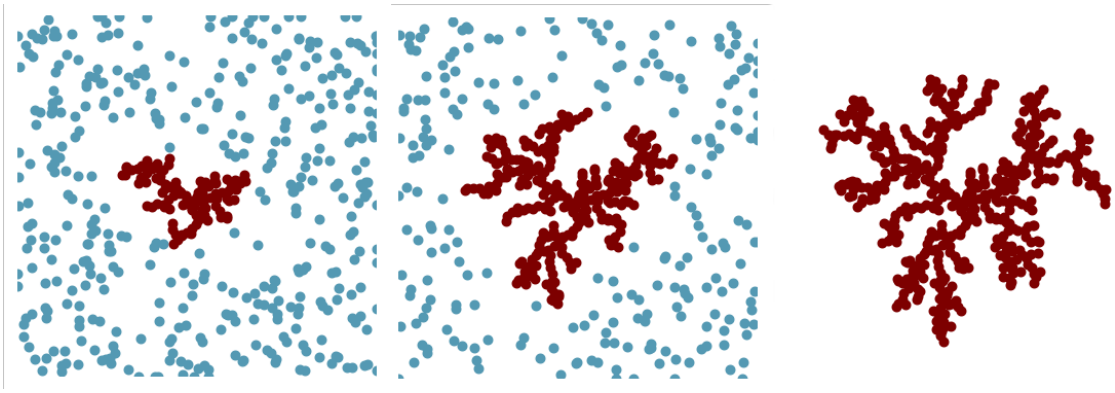

Figure 9: An evolution of Diffusion Limited Aggregration in 2D (Sayam, 2011). Evolution of the algorithm is Left to Right.

DLA can be used to generate entire levels (Wolverson, 2020) and was used to generate the Galaxy in the game EVE Online (Wikidot.com, 2022). While the analysis of DLA as a technique to generate entire levels is beyond the scope of this paper, Wolverson (2020) presents using DLA as an additional step applied to rooms generated via BSP, making the rooms appear less artificial: as if they have been eroded.

Dijkstra Maps

As described by Wolverson (2020), Dykstra Maps are the processes of creating a map of distances between locations in a level based on Dijkstra's (1959) pathfinding algorithm. The process involves creating a 2D map of distances from the start point of the level, ignoring walls. This is shown in Figure 10. After the algorithm has run, any section that has not been assigned a value can be removed from the level, as it is as unreachable from the players' start point. This can be particularly useful with CA generated levels as it solves the issue raised by Cerny Green *et al*

(2019) that CA can generate completable levels due to areas that are separated from the rest of the dungeon.

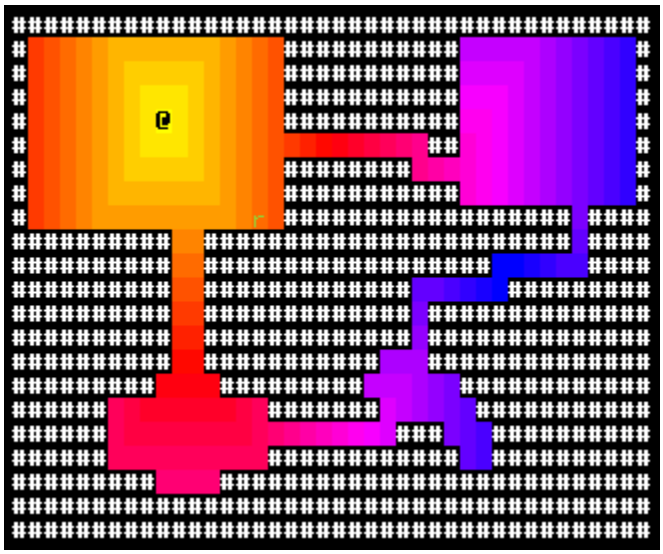

Figure 10: A visualization of a Dykstra map, where the start of the map is the '@' symbol (Dscreamer, 2015)

Wolverson (2020) also suggests that the shortest path through the level can be generated using a pathfinding algorithm (such as Dykstra or A*) and then Dykstra Map can be generated as a distance from the shortest path to generate the most efficient route through the level, this is known as the 'hot path'. This allows an additional step to be completed, such as culling rooms that are not on the hot path or informing a level furnishing system where to place enemies or rewards for players.

Dijkstra Maps are a valuable tool that can be used to create more interesting levels for players, however, they do come with additional performance consideration as one or multiple steps are required to be added to the pathfinding process. Additionally, while there have been no studies into the effectiveness of using Dijkstra Maps within level generation systems, there has been extensive research into the use of Dijkstra's algorithm as a pathfinding algorithm in games (Cui and Hao Shi, 2011; Handy Permana et al., 2018; Rafiq, Asmawaty Abdul Kadir and Normaziah Ihsan, 2020).

SUMMARY

In summary, PCG can offer effective techniques to generate dungeon levels in roguelikes. CA generation techniques create cave-like caverns, however, due to the organic nature of the technique, it is hard to control the output levels, in terms of both gameplay and visual design. In contrast, BSP creates structured levels, formed of constructed rooms. This method appears to create more playable levels as the process is more controlled and it is easier to understand how different parameters affect level generation. BSP has been criticized for creating levels that are too structured and it is harder to create complex room arrangements. On balance, BSP rooms seem to be the most appropriate generation technique for roguelike levels, where gameplay needs to be tightly controlled.

Entity generation is an important factor in the level generation process. Agent-based approaches have been tested, however these create levels that place most level entities at the beginning of the level. Spawners require rooms to be prefabricated, increasing designer workload, however, create better designed levels as the level can appear handcrafted.

Supplemental algorithms can also be added to the previously mentioned techniques. DLA can be used to erode areas of a pre-generated map; this is particularly useful for maps generated with BSP rooms. Dykstra Maps can be used as a separate path on top of the generated level. They can inform entity generation techniques to create more balanced gameplay and trim rooms that will not be visited by players.

Additional research should be conducted into level generation techniques. There is limited research available in the implementation of Dykstra Maps as well as the use of AI systems to generate levels.

Output Design

A practical implementation of the selected techniques will be created to support the research in this paper. This will be in the form of an interactive application that allows users to generate dungeons, with parameters for the size of the generation space and culling of rooms, for Windows and Web platforms. The application will be created using the Unity game engine version 2021.1.12f1 (Unity Technologies, 2021).

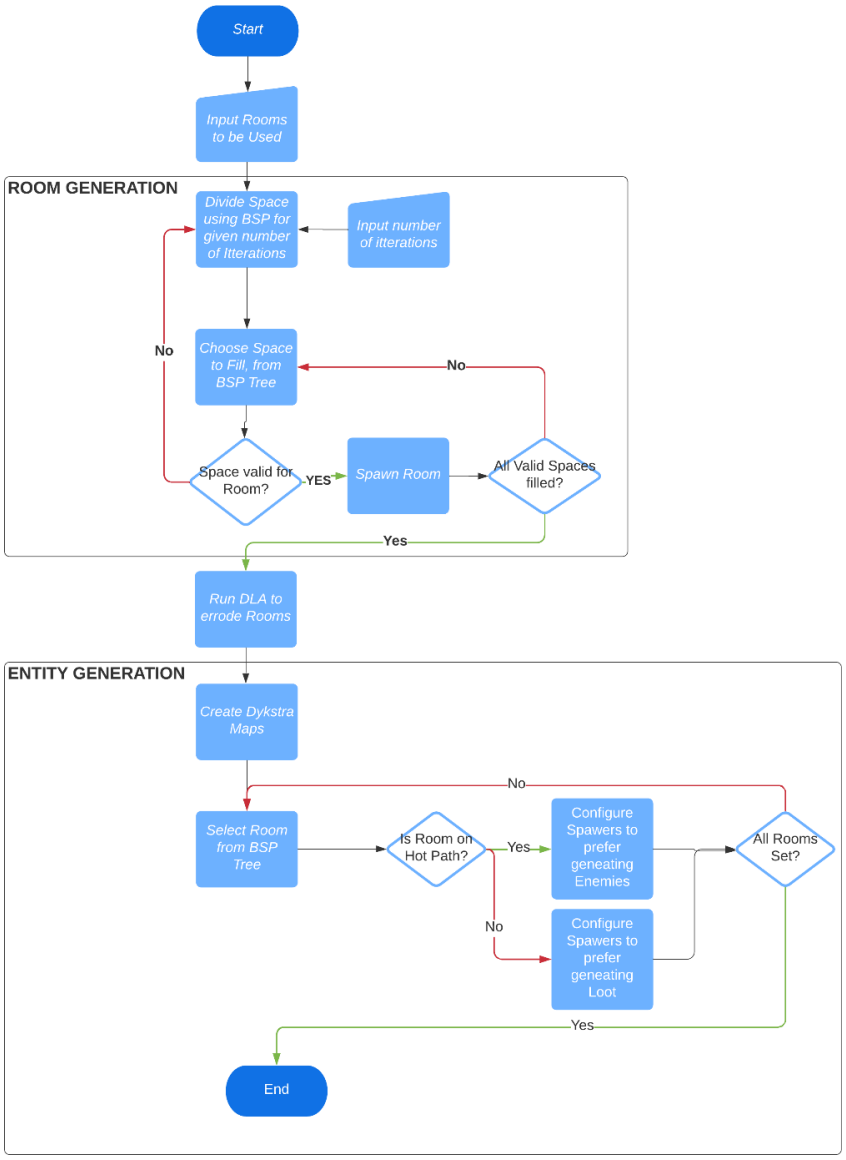

Figure 11: Flowchart showing the level generation process used in the application

BSP Rooms will be chosen for the generation of the game space. The algorithm will determine the appropriate places to generate several prefabricated rooms, as well as using the BSP tree to generate corridors between the rooms, as described by Shaker *et al* (2016). DLA will be used to deform the meshes of these rooms, at generation time, so that rooms will appear to be more naturally eroded and less constructed. This mitigates concerns of the rooms appearing too constructed. Spawners will be placed within these rooms to generate points where enemies and rewards are intended to be spawned: the types of entities that they spawn will be informed using Dykstra Maps. Rooms that are not on the level's 'hot path' will have a greater proportion of rewards, whereas rooms on the direct route through the level will have a greater proportion of enemies. This process is summarized in Figure 11.

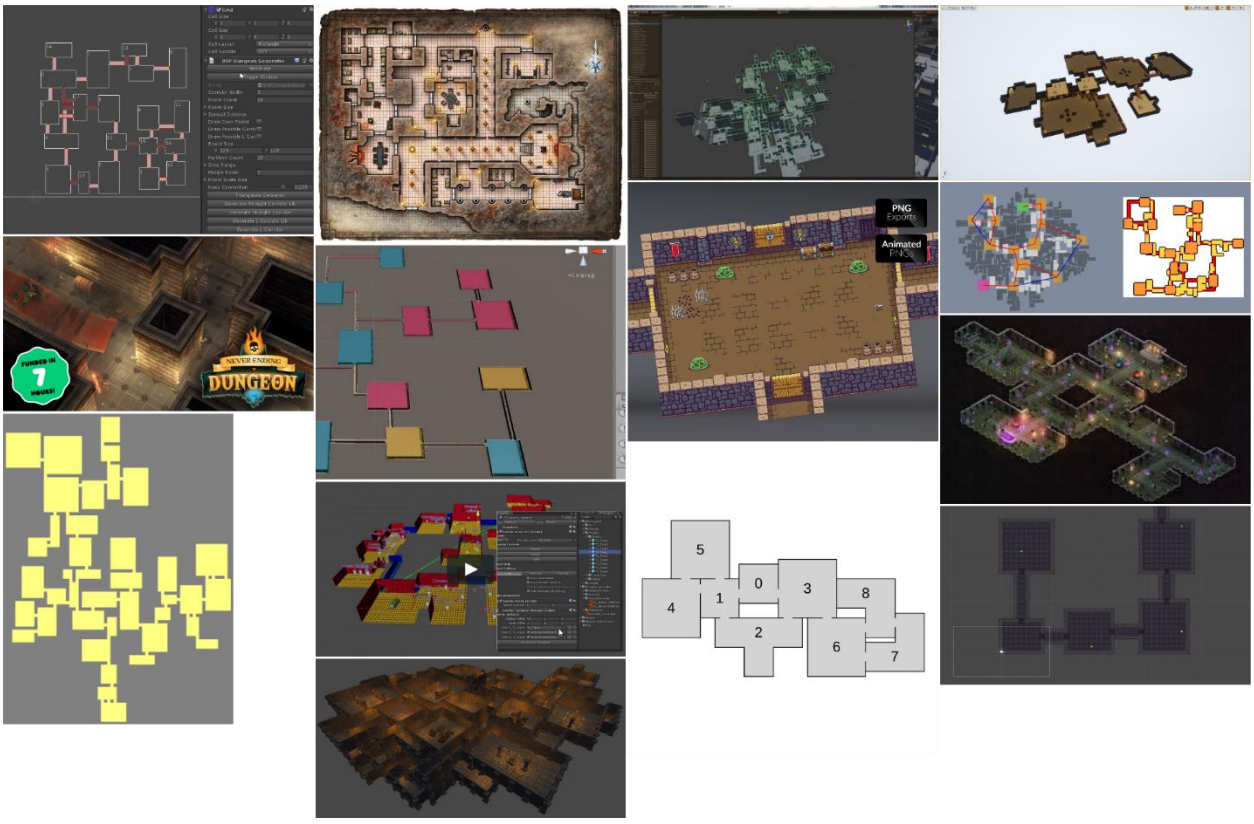

Figure 12: Mood board showing designs for the application's visualization

The created application will generate a 3D level which will be presented from a top-down isometric perspective, with all the rooms generated on the same vertical plane. During the generation process, the application will pause at each generation step to demonstrate to the user how each step impacts the generated level.

The BSP process will be shown as lines that divide the screen, similar to that of Figure 6. DLA will be shown by visualizing the particles with primitive shapes. The ‘hot path’ of the level, generated from the Dykstra Maps, will be shown by highlighting rooms that are determined to be on the path. The type of entity that is being spawned by a spawner will be represented by an appropriate icon.

After the application has been developed, a poster will be produced to show how the generation algorithm constructs a level, how entities are spawned and how supplemental algorithms enhance the level.

References

17-BIT (2015). *GALAK-Z* [Video Game]. 17-BIT.

Aikman, Z. (2015). *Galak-Z: Forever: Building Space-Dungeons Organically*. *GDC*. Available at: https://www.gdcvault.com/play/1021999/Galak-Z-Forever-Building-Space [Accessed 9 Feb. 2022].

Amplitude Studios (2014). *Dungeon of the ENDLESS* [Video Game]. SEGA.

Baron, J. (2017). Procedural dungeon generation analysis and adaptation. In: *ACM SE '17: Proceedings of the SouthEast Conference*. [online] Kennesaw, GA, USA: Association for Computing Machinery, pp.168–171. Available at: https://doi.org/10.1145/3077286.3077566 [Accessed 15 Feb. 2022].

Braben, D. and Bell, I. (1984). *Elite* [Video Game]. Acornsoft, Firebird, Imagineer.

Bycer, J. (2021). *Game Design Deep Dive: Roguelikes*. 1st ed. Boca Raton, Florida, USA: CRC Press.

Cerny Green, M., Khalifa, A., Alsoughayer, A., Surana, D., Liapis, A. and Togelius, J. (2019). Two-step constructive approaches for dungeon generation. In: *FDG '19: Proceedings of the 14th International Conference on the Foundations of Digital Games*. [online] FDG '19: Proceedings of the 14th International Conference on the Foundations of Digital Games. New York, NY, United States: Association for Computing Machinery. Available at: https://dl.acm.org/doi/10.1145/3337722.3341847 [Accessed 9 Feb. 2022].

Cook, M. (2013). *An Image of a Cave Generated with Cellular Automata in 2D*. [Online Image] *envatotuts+*. Available at: https://cdn.tutsplus.com/cdn-cgi/image/width=600/gamedev/uploads/2013/07/gdt-sim6.png.

Cui, X. and Hao Shi (2011). Direction Oriented Pathfinding In Video Games. *International Journal of Artificial Intelligence & Applications*, [online] 2(4), pp.1–11. Available at: https://www.researchgate.net/profile/Xiao-Cui-12/publication/267405818_Direction_Oriented_Pathfinding_In_Video_Games/links/54fd73740cf270426d125ade/Direction-Oriented-Pathfinding-In-Video-Games.pdf [Accessed 20 Feb. 2022].

Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, [online] 1(1), pp.269–271. Available at: https://doi.org/10.1007/BF01386390 [Accessed 20 Feb. 2022]. Accessed via: http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf.

Dodge Roll. (2016). *Enter the Gungeon* [Video Game]. Devolver Digital.

Dscreamer (2015). *A Visualization of a Dykstra map*. [Online Image] *RougeBasin*. Available at: http://www.roguebasin.com/index.php/File:Dijk_flee.png [Accessed 5 Mar. 2022].

Dubois, S. (2014). *Unite 2014 - Dungeon of the Endless Rendering and Procedural Content*. *Unity*. Available at: https://www.youtube.com/watch?v=zPQOHX9hiLo [Accessed 18 Feb. 2022].

Dubois, S. and Amplitude Studios (2014). *A screenshot of a Level Template from Dungeon of the Endless*. [Video Screenshot from YouTube Video] *Unite 2014 - Dungeon of the Endless Rendering and Procedural Content*. Available at: https://www.youtube.com/watch?v=zPQOHX9hiLo [Accessed 20 Feb. 2022]. Screenshot from YouTube video.

Fan, X., Li, B. and SIsson, S. (2019). Binary space partitioning forest. In: K. Chaudhuri and M. Sugiyama, eds., *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*. [online] PMLR, pp.3022–3031. Available at: https://proceedings.mlr.press/v89/fan19b.html.

Fredrik (2004). *Binary Space Partition*. [Online Image] *Wikipedia*. Available at: https://commons.wikimedia.org/wiki/File:Binary_space_partition.png [Accessed 15 Feb. 2022]. Public Domain.

Gardner, M. (1970). MATHEMATICAL GAMES. *Scientific American*, [online] 223, pp.120–123. Available at: http://www.jstor.org/stable/24927642.

Guerrilla Games (2017). *Horizon Zero Dawn* [Video game]. Sony Interactive Entertainment.

Handy Permana, S.D., Yogha Bintoro, K.B., Arifitama, B. and Syahputra, A. (2018). Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game. *IJISTECH (International Journal Of Information System & Technology)*, [online] 1(2), p.1. Available at:

https://www.researchgate.net/profile/Budi-Arifitama/publication/325368698_Comparative_Analysis_of_Pathfinding_Algorithms_A_Dijkstra_and_BFS_on_Maze_Runner_Game/links/5b36489d4585150d23e1d802/Comparative-Analysis-of-Pathfinding-Algorithms-A-Dijkstra-and-BFS-on-Maze-Runner-Game.pdf [Accessed 20 Feb. 2022].

Johnson, L., N. Yannakakis, G. and Togelius, J. (2010). Cellular automata for real-time generation of infinite cave levels. In: *PCGames '10: Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. [online] PC Games 2010, the Workshop on Procedural Content Generation in Games. New York, NY, United States: Association for Computing Machinery. Available at: https://dl.acm.org/doi/abs/10.1145/1814256.1814266 [Accessed 9 Feb. 2022].

Liu, Y., Liu, Z., Huang, L. and Sun, J. (2010). Fractal model for simulation of frost formation and growth. *Science China Technological Sciences*, [online] 53(3), pp.807–812. Available at: https://link.springer.com/article/10.1007/s11431-009-0189-y [Accessed 20 Feb. 2022].

Maerivoet, S. and De Moor, B. (2005). Cellular automata models of road traffic. *Physics Reports*, [online] 419(1), pp.1–64. Available at: https://www.sciencedirect.com/science/article/pii/S0370157305003315?casa_token=xro-HuXvzf4AAAAA:WQgpwD9WZ3nVZW_nZIYhT2g1KYJEa8kktVdY0ZNXOsZatwcXB4G6s9GhbVPRnzOswMIb50itEyo [Accessed 9 Feb. 2022].

McMillen, E. and Himsl, F. (2011). *The Binding of Isaac* [Video Game]. Edmund McMillen.

Mojang Studios (2022). *An image of generated terrain from Minecraft*. [Online Image] *What is Minecraft?* Available at: https://www.minecraft.net/content/dam/games/minecraft/screenshots/explore.png [Accessed 7 Mar. 2022].

Mojang Studios (2022). *Minecraft* [Video Game]. Mojang Studios.

Mossmouth (2013). *Spelunky* [Video Game]. Mossmouth

Nexon, devCAT studio and NEXON Korea (2004). *Mabinogi* [Video Game]. Nexon, devCAT studio, NEXON Korea.

Nexon, devCAT studio, NEXON Korea and Rydian (2017). *Image of a Dungeon from a novel in Mabinogi*. [Online Image] *https://wiki.mabinogiworld.com/view/Dunbarton_Collection_Book*. Available at: https://wiki.mabinogiworld.com/images/e/eb/MabiNovel_Background_Rabbie_Dungeon.png [Accessed 18 Feb. 2022].

P. A. Macedo, Y. and Chaimowicz, L. (2017). Improving procedural 2D map Generation based on multi-layered cellular automata and Hilbert curves. In: *SBC – Proceedings of SBGames 2017*. SB Games 2017. SBCGames, p.495.

Pittman, D. (2015). *Level Design in a Day: Procedural Level Design in Eldritch*. *GDC*. Available at: https://www.gdcvault.com/play/1022110/Level-Design-in-a-Day [Accessed 20 Feb. 2022].

PlayStation Europe (2016). *Galak-Z | Gameplay trailer | PS4*. *YouTube*. Available at: https://www.youtube.com/watch?v=YQAfZK7dc9g [Accessed 15 Feb. 2022].

Radha, H., Vetterli, M. and Leonardi, R. (1996). Image compression using binary space partitioning trees. *IEEE Transactions on Image Processing*, [online] 5(12), pp.1610–1624. Available at: https://www.researchgate.net/publication/37414113_Image_Compression_Using_Binary_Space_Partitioning_Trees [Accessed 25 Feb. 2022].

Rafiq, A., Asmawaty Abdul Kadir, T. and Normaziah Ihsan, S. (2020). Pathfinding Algorithms in Game Development. *IOP Conference Series: Materials Science and Engineering*, [online] 769(1), p.012021. Available at: https://iopscience.iop.org/article/10.1088/1757-899X/769/1/012021/meta [Accessed 20 Feb. 2022].

Rampe, J. (2000). *Visions of Chaos* [Computer Program]. Softology.

Roguebasin.com. (2018). *IRDC 2008 - RogueBasin*. [online] Available at: http://www.roguebasin.com/index.php/IRDC_2008 [Accessed 22 Feb. 2022].

Santamaria-Ibirika, A., Cantero, X., Huerta, S., Santos, I. and Bringas, P.G. (2014). Procedural Playable Cave Systems Based on Voronoi Diagram and Delaunay Triangulation. In: *2014*

International Conference on Cyberworlds. [online] International Conference on Cyberworlds. IEEE, pp.15–22. Available at: https://ieeexplore.ieee.org/abstract/document/6980738 [Accessed 15 Feb. 2022].

Sapin, E., Bull, L. and Adamatzky, A. (2007). A Genetic approach to search for glider guns in cellular automata. In: *2007 IEEE Congress on Evolutionary Computation*. [online] 2007 IEEE Congress on Evolutionary Computation. Manhattan, New York, U.S.: IEEE, pp.2456–2462. Available at: https://ieeexplore.ieee.org/document/4424779 [Accessed 9 Feb. 2022].

Sayam, H. (2011). *Diffusion-Limited Aggregation: A Real-Time Agent-Based Simulation - Wolfram Demonstrations Project*. [online] Wolfram.com. Available at: https://demonstrations.wolfram.com/DiffusionLimitedAggregationARealTimeAgentBasedSimulation/ [Accessed 20 Feb. 2022].

Shaker, N., Togelius, J., Liapis, A., Lopes, R. and Bidarra, R. (2016). Constructive generation methods for dungeons and levels. In: F. Pachet, P. Gervás, A. Passerini and M. Degli Esposti, eds., . *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. [online] Cham, Switzerland: Cham Springer International Publishing, pp.38–55. Available at: http://pcgbook.com/ [Accessed 15 Feb. 2022].

Togelius, J., Kastbjerg, E., Schedl, D. and Yannakakis, G. (2011). What is procedural content generation?: Mario on the borderline. In: *PCGames '11: Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*. [online] PCGames '11. New York, NY, United States: Association for Computing Machinery. Available at: https://dl.acm.org/doi/abs/10.1145/2000919.2000922?casa_token=4BXO7ybO_DMAAAAA:TvUZZO-_n_PeKqakcBAcUMLSEY6929kK3YUOe7_R2-EiqwWqhe24YSrPkRDVw4gOrBZTkH6f9QrB2g [Accessed 25 Feb. 2022].

Togelius, J., Shaker, N. and J. Nelson, M. (2016). Introduction. In: F. Pachet, P. Gervás, A. Passerini and M. Degli Esposti, eds., *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. [online] Cham, Switzerland: Cham Springer International Publishing, pp.38–55. Available at: http://pcgbook.com/ [Accessed 15 Feb. 2022].

Toy, M., Wichman, G., Arnold, K., Lane, J. and A.I. Design, 1980. *Rouge*. Epyx.

Unity Software (2022). *Unity* [Computer Program]. Unity Technologies

Unity Technologies (2020). *Unity Asset Store - The Best Assets for Game Making*. [online] Unity Asset Store. Available at: https://assetstore.unity.com [Accessed 26 Feb. 2022].

Unity Technologies (2021). *Unity 2021.1.12f1* [Computer Program] Unity Technologies

van der Linden, R., Lopes, R. and Bidarra, R. (2014). Procedural Generation of Dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, [online] 6(1), pp.78–89. Available at: https://ieeexplore.ieee.org/abstract/document/6661386 [Accessed 9 Feb. 2022].

van Muijden, J. (2017). *GPU-Based Run-Time Procedural Placement in "Horizon: Zero Dawn." GDC*. Available at: https://www.gdcvault.com/play/1024120/GPU-Based-Run-Time-Procedural [Accessed 25 Feb. 2022].

Vichniac, G.Y. (1984). Simulating physics with cellular automata. *Physica D: Nonlinear Phenomena*, [online] 10(1-2), pp.96–116. Available at: https://www.sciencedirect.com/science/article/abs/pii/0167278984902537 [Accessed 9 Feb. 2022].

Wang, S., Ji, H., Li, P., Li, H. and Zhan, Y. (2020). Growth diffusion-limited aggregation for basin fractal river network evolution model. *AIP Advances*, [online] 10(7), p.075317. Available at: https://aip.scitation.org/doi/full/10.1063/5.0011624 [Accessed 20 Feb. 2022].

Whitehead, J. (2020). Spatial layout of procedural dungeons using linear constraints and SMT solvers. In: *FDG '20: International Conference on the Foundations of Digital Games*. [online] FDG '20: International Conference on the Foundations of Digital Games. Bugibba, Malta: Association for Computing Machinery. Available at: https://doi.org/10.1145/3402942.3409603 [Accessed 15 Feb. 2022].

Wikidot.com. (2022). *EVE Online - Procedural Content Generation Wiki*. [online] Available at: http://pcg.wikidot.com/pcg-games:eve-online [Accessed 20 Feb. 2022].

Witten, T.A. and Sander, L.M. (1981). Diffusion-limited aggregation, a kinetic critical phenomenon. *Phys. Rev. Lett.*, [online] 47(19), pp.1400–1403. Available at: https://link.aps.org/doi/10.1103/PhysRevLett.47.1400.

Wolverson, H. (2020). *Herbert Wolverson - Procedural Map Generation Techniques*. *Roguelike Celebration*. Available at: https://www.youtube.com/watch?v=TlLIOgWYVpI [Accessed 9 Feb. 2022]. This talk is from the 2020 virtual Roguelike Celebration: https://roguelike.club/event2020.html.

Zapata, S. (2017). *Santiago Zapata - What is a Roguelike*. *YouTube*. Available at: https://www.youtube.com/watch?v=wwc7pZqs9UA&t=5s [Accessed 22 Feb. 2022]. This video is from the 2017 Roguelike Celebration: https://roguelike.club/event2017.html.

Ziegler, P. and von Mammen, S. (2020). Generating Real-Time Strategy Heightmaps using Cellular Automata. In: *FDG '20: International Conference on the Foundations of Digital Games*. [online] FDG '20: International Conference on the Foundations of Digital Games. New York, NY, United States: Association for Computing Machinery. Available at: https://dl.acm.org/doi/abs/10.1145/3402942.3402956 [Accessed 9 Feb. 2022].