Matthew L Wheatley

SUPERVISOR: NAME

SECOND SUPERVISOR: NAME

Optimisations of The Wave Function Collapse Algorithm In Procedural Generation

GDEV60001 GAMES DEVELOPMENT PROJECT

Contents

[Abstract 2](#_gjdgxs)

[Introduction 2](#_30j0zll)

[Aims and Objectives 2](#_1fob9te)

[Aim 2](#_3znysh7)

[Objectives 3](#_2et92p0)

[Evaluate the Base WFC Algorithm: 3](#_tyjcwt)

[Develop Optimization Strategies: 3](#_3dy6vkm)

[Evaluate the Optimizations: 3](#_1t3h5sf)

[Literature Review 4](#_4d34og8)

[Procedural Content Generation: 4](#_gprn0spke3ik)

[Rouge 4](#_89u327wvsnf8)

[Minecraft 4](#_1jas2suj0o18)

[Dwarf Fortress: 5](#_2u2ot24cywqu)

[WFC: 7](#_vz6svlr6uggu)

[A Review On PCG Algorithms: 9](#_1nhemxoozxvp)

[Previous Work: 12](#_98c40ntwwsuf)

[Research Methodologies 14](#_2s8eyo1)

[Results and Findings 15](#_17dp8vu)

[Discussion and Analysis 15](#_3rdcrjn)

[Conclusion 15](#_26in1rg)

[Recommendations 17](#_lnxbz9)

[References 18](#_35nkun2)

[Appendices 19](#_1ksv4uv)

[Appendix 1 – xxx 19](#_44sinio)

# Abstract

***An overview of the project***

# Introduction

Procedural Generation Content(PCG) plays a pivotal role in modern game development, offering unique and varied experiences through each playthrough through the generation of terrain, maps, and assets(Hendrikx *et al., 2013)*. Central to this innovation are algorithms that are highly optimized for efficiency. Among these, the Wave Function Collapse (WFC) algorithm, inspired by quantum mechanics and developed by Maxim Gumin in 2016, stands out for its unique approach to generating coherent and diverse patterns from a limited set of inputted rules (Gumin, 2016). Despite its potential, the WFC algorithm's application has been largely theoretical due to its computational demands, particularly its inherent serial processing limitation. This dissertation explores the optimization of the WFC algorithm by transitioning from a single-threaded to a multi-threaded, parallelized process, enhancing efficiency and scalability (Nie *et al., 2023)*. The WFC optimization strategy I will be discussing will be:

Nested WFC: This approach divides the grid into smaller, manageable regions that can be processed in parallel, increasing efficiency by allowing multiple regions to be worked on simultaneously (Nie *et al., 2023)*.

Stitched WFC: Here, the grid is split into separate, smaller regions that are processed individually in parallel and then stitched together with a different version of the WFC algorithm, ensuring continuity and coherence across the entire grid even allowing for more personalization (Watt Designs, 2023).

By addressing these challenges, this work aims to transition WFC from a lesser-used method to a practical tool used in many procedural generations’ games, unlocking new possibilities for procedural content generation within larger scales to create diversity and custom experience even allowing for a plugin of this is widely used in simple to larger scale games.

# Aims and Objectives

## Aim

The primary aim of this dissertation is to critically evaluate the performance of a standard implementation of the Wave Function Collapse (WFC) algorithm and to assess the effectiveness of three specific optimizations—stitching, and nested WFC—in improving its scalability, efficiency, memory usage, and the quality of procedural generation in game development contexts.

## Objectives

### Evaluate the Base WFC Algorithm:

* Objective 1.1: Conduct and evaluate a standard 3D WFC algorithm to establish benchmarks for speed, memory usage, scalability, and output reproducibility.
* Objective 1.2: Identify limitations and then areas for improvement in the base algorithm that can be addressed through optimization.

### Develop Optimization Strategies:

* Objective 2.1: Design and implement the nested optimization to divide the processing grid into smaller more manageable sections, aiming to enhance processing speed and scalability.
* Objective 2.2: Design and implement the stitching optimization by processing separate, smaller regions in parallel and then combining them, focusing on improving scalability while maintaining output coherence and continuity.

### Evaluate the Optimizations:

* Objective 3.1: Assess the impact of each optimization strategy on the processing speed of the WFC algorithm, comparing it to the benchmarks established for the base algorithm.
* Objective 3.2: Evaluate the scalability of each optimization by testing their performance in generating content of varying sizes and complexities.

# Literature Review

## Procedural Content Generation:

Procedural Content Generation (PCG) represents a shift in video game development, leveraging algorithms to automatically generate assets in real-time, thereby reducing manual input and enhancing the variability and uniqueness of each player (Hendrikx *et al., 2013)*. Originating as a niche technique starting with Rouge, PCG has rapidly evolved from being uncommon to being used in AAA games like the god of War Ragnarök’s new DLC Valhalla (Levi Winslow, 2023).

### Rouge

#### Origin and Development

Rogue, developed in 1980 by Michael Toy and Glenn Wichman, marked a significant milestone in video game history(A.I. Design, 1980; Edge Staff, 2009). It introduced the genre of "RogueLike" games and showcased one of the earliest applications of PCG to create vast, explorable dungeons. This innovative approach allowed Rogue to offer endless variations in dungeon layouts, items, and enemies, ensuring a unique experience with every playthrough.

#### PCG Methodology

The PCG in Rogue was relatively simple yet highly effective. Utilising a grid-based system, the game dynamically generated dungeon levels, with each grid cell potentially representing a room, corridor, or wall. This system randomly placed rooms and then connected them with corridors, adhering to a set of predefined rules to ensure navigability and variety(Edge Staff, 2009). Additionally, the game randomised the placement of enemies, treasures, and items, adding to the challenge and depth of each run.

#### Impact and Contributions to PCG:

Rogue's application of PCG was groundbreaking, enhancing replayability and laying the groundwork for future PCG techniques in video games. Its method of creating complex and engaging gameplay experiences with limited computational resources set a precedent for the use of PCG in game design(Edge Staff, 2009). The legacy of Rogue is evident in numerous games within the roguelike genre and beyond, including The Binding of Isaac, Spelunky, and Dwarf Fortress, each building upon the foundations of procedural generation in unique ways.

### Minecraft

#### Origin and Development:

Minecraft, developed in 2011 by Mojang Studios, is a monumental game in the realm of PCG, widely recognized for its innovative use of algorithmic generation to create expansive, dynamic worlds(Mojang Studios and Markus Persson, 2011). At its core, Minecraft utilises a seed-based system to procedurally generate vast, block-based landscapes, which include biomes, structures, and resource(Awiszus, Schubert and Rosenhahn, 2021). Through the use of multiple PCG techniques like seeded generation, Perlin noises, biome selection, and structural generation.

#### PCG Techniques

##### Seeded generation:

At the heart of Minecraft's PCG lies the seeded generation system. This system ensures consistency in world creation by using a seed, a numerical value, to initialize the world's generation process(Awiszus, Schubert and Rosenhahn, 2021). The seed acts as the foundation for all subsequent procedural generation through the use of deterministic programming ensuring that the same seed will always generate the same world layout, allowing players to share and explore identical worlds by simply exchanging seed values.

##### Noise Algorithms:

Minecraft employs noise algorithms, such as Perlin and Simplex noise, to generate the complex terrain of its worlds (Awiszus, Schubert and Rosenhahn, 2021). These algorithms create natural-looking landscapes with varying elevations, from towering mountains to deep oceans, by simulating random yet coherent noise patterns. The choice of noise algorithms enables the creation of a diverse range of terrains, enhancing the exploration experience by introducing varied geographical features.

##### Biome Selection Algorithms:

The game further diversifies its worlds through an intricate biome selection process. Biomes, distinct regions with specific climates, vegetation, and geographies, are generated based on noise-generated temperature and humidity maps(Awiszus, Schubert and Rosenhahn, 2021). This process ensures a rich variety of environments, from arid deserts to lush jungles, each offering unique resources and challenges. The biome algorithm takes into account the generated terrain to place biomes in a way that feels natural and coherent, contributing to the immersive quality of the game's worlds.

##### Structure Generation:

Beyond the natural landscape, Minecraft also features procedurally generated structures, such as villages, temples, and mineshafts(Awiszus, Schubert and Rosenhahn, 2021). These structures are created using a combination of predefined templates and algorithms that adapt the structures to the terrain and biome in which they are placed. This method allows for the seamless integration of human-made elements into the natural world, offering players opportunities for exploration and interaction within the game's environment.

#### Impact and Contributions to PCG:

Minecraft's use of PCG has not only shaped the gameplay experience but also contributed significantly to the field of procedural content generation. By successfully implementing a variety of PCG methods to create a dynamic and engaging game world, Minecraft has inspired further research and development in PCG technologies(Awiszus, Schubert and Rosenhahn, 2021). Its open-ended nature and the use of seeds for world generation have facilitated community engagement, with players sharing seeds of particularly interesting or challenging worlds.

### Dwarf Fortress:

#### Origin and Development:

Dwarf Fortress, developed in 2006 by brothers Tarn and Zach Adams, is a cornerstone of PCG, renowned for its extensive use of algorithmic generation techniques used to create large dynamic worlds(Bay 12 Games, Tarn Adams and Zach Adams, 2006). PGC techniques are used for procedurally generating overworld to a Non-player character-generated background. Dwarf Fortress employs diverse techniques including Perlin noise for natural landscape formation, cellular automata for simulating environmental processes, random walk and diamond-square for terrain variability, Voronoi Diagrams for regional division, agent-based Modeling for NPC behaviour, Lindenmayer Systems for plant growth patterns, and rule-based generation for lore-rich NPC Histories(Martin, 2012).

#### PCG techniques:

##### Cellular Automata:

Dwarf Fortress uses Cellular Automata (CA) for simulating natural processes like water flow, lava movement, and vegetation spread. CA operates on a grid where each cell updates its state based on its neighbours and a set of rules(Martin, 2012). This method allows for the emergence of complex, dynamic patterns from simple interactions, contributing to the game's environmental realism and unpredictability.

##### Rule-based Generation:

The game employs rule-based generation to create intricate lore and NPC behaviours. By establishing a set of predefined rules, Dwarf Fortress generates unique histories, cultures, and personalities for its non-player characters and civilizations(Martin, 2012). This technique ensures a rich, evolving narrative landscape where each playthrough offers new stories and interactions, deeply enriching the player's experience.

#### Impact and Contributions to PCG:

Dwarf Fortress stands as a monumental achievement in PCG, showcasing the potential of algorithmic complexity to create vast, explorable, and ever-changing worlds. Its deep integration of various PCG techniques has inspired a generation of game developers and researchers, pushing the boundaries of what is possible in game design and world-building. The game not only serves as a benchmark for PCG's capabilities but also as a learning tool for those interested in the underlying mechanics of algorithmic generation. Its open-ended nature and the sheer depth of its simulated world highlight the transformative power of PCG in creating immersive and enduring gaming experiences.

### WFC:

Wave function collapse is a relatively recent innovation in the world of PCG, developed by Maxim Gumin in 2016(Gumin, 2016), inspired by the waveform collapse phenomenon discovered in quantum mechanics. The PCG method develops on the concept of having a grid with predefined tiles, it identifies the tile with the lowest entropy by measuring how uncertain the tile is and collapsing the tile down, then propagating it to all surrounding tiles. Then repeats the process, gradually increasing the complexity of the grid making patterns(Gumin, 2016).

Within WFC development there are two main variants of WFC algorithm, The first being the type being the paper's main development area, is Tile-Based WFC or also known as Socket-Based WFC(Nie<i> et al.,</i> 2023), and the other type being Pixel-Based WFC(Gumin, 2016).

#### Pixel-Based WFC:

##### Overview and introduction:

Pixel-Based WFC(P-WFC) is a sophisticated variant of WFC developed for the purpose of generating highly-detailed repeating images from a sample image(GridBugs, 2019). By analysing and extracting rules based on the adjacency of pixels in the sample image, P-WFC uses the rules to create coherent patterns that can seamlessly tile across a grid. This process allows the creation of fine grain, tailored and tileable images that can be used to make detailed and complex textures/patterns(GridBugs, 2019).

##### How it works:

The P-WFC process can be encompassed in several critical steps; input image analysis and rule extraction, initialise entropy and the grid, pixel selection and collapsation, propagation of rules and repeat until complete collapse. To break it down further the program takes in an input image that creates a sampler, that is a smaller area of the input image, that samples a region and creates rules from sampled areas by determining the orientation and position of pixel colours in relation to each other and slowly moves around the image moving the sampling area to create more and more of these rules(GridBugs, 2019). Next it creates a grid and updates all of that grid with all the possible pixel states. This is where the repeated section starts with a selection of the lowest entropy, which just meanspixel with lowest possible amount of colours it can be, and proceeds to collapse it down meaning it picks one of the possible pixels at random. Following the rules the pixels surrounding the collapse pixel will update its entropy removing some of the possible colours the neighbouring pixels are allowed to be. Then repeat until all the pixels have collapsed. With the rules that were taken from the sample image, the output image should have a similar outcome(Gumin, 2016).

#### Simple Tile WFC:

##### Overview and introduction:

Simple Tile WFC(T-WFC) also known as Socket-based WFC presents a further evolution from the pixel-oriented approach of P-WFC by tiles and predefined rules to choose the placement of such tiles (Gumin, 2016). Being able to hold more possible information with the possibility of removing the need for a grid and creating more no comformant natural shapes(Kim<i> et al.,</i> 2019). The tiles can be used to create further randomness and variability because of their basic non-identifying features that can be a placeholder for a group of tiles with a multitude of features and presets.

##### How it works:

Like the P-WFC the steps for T-WFC have several similar critical steps; rule creation, initialise entropy and the grid, Tile selection and collapsation, propagation of rules and repeat until complete collapse. T-PWC is very similar to P-WFC there a few differences during initialization instead of giving a list of possible pixels it can be you must give it possible tiles it can make the memory issue more exaggerated at the benefit is more detail. Another difference is that instead of following the rules and checking the pixel positions the tile that is being propagated has to check all of its rules and remove the non-possible ones that don't fit the socket type(Lioret<i> et al.,</i> 2022). Sockets are named as such as they resemble the idea that only one socket type can attach to its matching socket type. Socket A can only be next to Socket A and so on (Gumin, 2016). The final difference is that the outcome is more data-oriented and can be further built off rather than having a complete output. This can be converted to an image but also can be used in further PCG.

#### Challenges and Limitations:

With the advancement of WFC, managing performance, memory, and visual clarity emerges as a significant challenge(Gumin, 2016). The crux of the issue lies in the voluminous information each tile must manage during the collapse process, necessitating a data structure that optimally balances speed and storage efficiency(Gumin, 2016). It must hold all of the possible tiles I can become in some sort of data structure that balances the speed of access and compression. While a small set of rules in a modestly sized grid poses minimal demand, scaling up introduces substantial challenges, particularly for projects aimed at enhancing scalability. This can be mitigated instead of hold many copies of the rules the tile can hold a list of identifiers of the tiles it can save storage of larger rules, also the choice of variable type can change only on a small scale but will add up so using lighter wait things like byte instead of ints will save on more storage and speed(Gumin, 2016). Furthermore with many of the possible optimizations can be applied to help with speed, mitigation to visual noise and distinction needs to be considered.

Another challenge in scaling WFC is optimising the storage and retrieval of the tile grid. As the scale of the grid increases, effectively accessing the singular tiles will become more crucial and the traditional storage methods like arrays have the potential to lead to slower access times and increased size. Advanced storage strategies, such as segmentation and the use of dictionaries, offer promising solutions. Segmentation can reduce the complexity by dividing the grid into smaller more manageable selection, while dictionary allows for a constant time lookup by accessing each tile with a unique identifier with a vector3 or custom tuple. Further exploration of spatial indexing techniques, like QuadTrees or R-trees, might also provide efficient alternatives for managing spatial data within T-WFC. The choice of storage solution has significant implications for both the development process and the algorithm's runtime performance, highlighting the need for careful consideration of the trade-offs involved.

With WFC a large and prominent problem is that the WFC algorithm is very singular-threaded, meaning the process that the WFC algorithm takes must be done one after another without much room for doing multiple things at the same time. This isn't an issue with a smaller scale grid but at the number tiles scale the time to complete the process follows a very equal trend. With modern processors, they are almost all multi-core focused to do multiple calculations at the same time. Without going against the traditional method of WFC the biggest benefit would be the parallelisation of the propagation of rules across the grid after a tile is collapsed. However, since a tile can affect another tile during the propagation there are chances of race conditions and/or use of outdated data. Both of these issues can cause the program to have to run the propagation more than once, possibly negating any benefits of multithreading.

For broader possibilities the segmentation of the grid could allow for multiple section to be worked on at the same time collapsing at the same time and propagating at the same time making the potential for multithreading realistic however losing the complete authenticity of the WFC algorithm and even compromising the visual cohesiveness the algorithm is known for. Despite this the WFC algorithm's versatility and its ability to “stitch” together 2 pieces of previously generated sections could be used to repair some of the cohesiveness.

A final foreseeable issue with WFC and inputtable rules is the ability to create a possibly “un-collapsible” grid in which no matter the attempts made not all the tiles will be possible. There are 2 potential solutions behind this, them being preemptive and reactive. The preemptive solution is to make the propagation extremely extensive updating each tile multiple times to make sure there is not a tile that is only kind of possible, this concept can be possible but also very time-consuming and potentially impossible to tell if reached as you would have to fully collapse it to recognize if done. Both solutions present trade-offs between computational efficiency and the likelihood of successfully collapsing the grid. The preemptive approach, with its extensive rule propagation, may offer a more deterministic path to a solution but at the cost of increased processing time and computational overhead. Conversely, the reactive strategy, with its flexibility and adaptability, may expedite the collapse process but at the risk of recursive complexity, however still has the higher potential of increased speed making it the more viable option.  
  
By embracing these adaptations, there's potential not only to maintain the integrity and output quality of WFC but also to significantly enhance its scalability and efficiency. However there may be possible solutions that have not been discussed here, research into other PCG methods and their limitations and solutions could be a valuable resource into what it can develop into.

### A Review On PCG Algorithms:

#### Perlin noise:

Perlin noise, a groundbreaking algorithm developed in 1983 by Ken Perlin, represents an advancement in natural PCG. Perlin noise was developed to produce natural-looking textures and in tern terrains, Perlin noise generated textures from gradient noise, which is used to create smooth and convening visuals characterised by its repeatability, controllability and cohesiveness(Hart, 2001). The Perlin noise algorithm's capacity to generate realistic, lifelike patterns has made it become a staple of any PCG toolkit, becoming applicable for a myriad of techniques from design to development work(Hart, 2001).

A game can utilise Perlin noise to craft richly varied terrains, from towering mountains to sprawling plains, imbuing the virtual world with a depth and realism that mirrors the natural world. The application showcases not only the ability of the algorithm's versatility but also its quality of environments, providing dynamic ever-changing unique experience(Hart, 2001)s.

Despite the acclaim, Perlin noise is not without its limitations and problems. One of these challenges is computation intensity, particularly when larger-scale terrains with high complexity. This in turn creates a problem of slower processing times and this creates performance issues interrupting the player's experience. To mitigate the possible limitations various optimization strategies have been developed and used, in particular multithreading to leverage multicore processors. Even this can be too inefficient to compute on the fly, however the ability to precompute and cache for further generation and creation. These developments have made the Perlin noise algorithm versatile.

#### Cellular automata:

The historical development of CA was significantly propelled forward with the introduction of “The Game of Life” by John Conway in the 1970s. The Game of Life is less of a game but a simulation where cells on a grid live, die or multiply based on the state of neighbours, based on its simple rule set it continues to exhibit complex and dynamic behaviours(Adams and Louis, 2017).

The use of CA in this context showcases the algorithm’s capacity for generating complex systems that CA is known for but the versatility which CA affords. The use of CA allows for easily influenced environments that respond to changes based on other underlying simulated processes, mirroring the very system of natural ecosystems which it wished to copy(Adams and Louis, 2017).

Despite the simplicity of CA, there are chances of several significant shortcomings in simulations like in a lot of PCG or simulations. These include of high computation demand required to simulate each cell in a larger size, this can be mitigated by only simulating the “alive” cells and surrounding cells(Adams and Louis, 2017). Another foreseeable problem would be memory management to efficiently access parts of the grids and a continuation as the grid scales so do the memory requirements. Both can be managed with reasonable methods in terms of computation because all cells are updated simultaneously, and the grid can be broken down into regions and done in parallel reducing real-time time complexity. For memory management instead of storing a whole grid of bools, a custom struct could be used and a dictionary for constant time lookups and only need to hold “alive” cells on the grid.

#### Random Walk:

Random Walk is a mathematical algorithm that simulates a path by repeatedly taking random steps. The origin comes from probability theory, it serves a multitude of purposes from simulating natural processes to the creation of dungeons to explore(Baron, 2017). Random Walk is instrumental in generating vast, intricate underground labyrinths and landscapes, all to enhance the player's experiences and replayability.

Random Walk can help simplify the creation of complex and natural environments by breaking the down into a series of steps, each based on a decision decided by probability(Baron, 2017). This method allows for the detailed and granular generation of caves, labyrinths, dungeons and other features.

While computationally the algorithm does not seem demanding, as the complexity of the walk increases and layers of generation also increase, there is a point where trading the algorithm creates a tradeoff between complexity and performance. Another issue comes from the “randomness” of the algorithm, the outcomes are not always desirable or maybe even repetitive.

A method of mitigation for the initial issue is to further break down the development of the full generation algorithm, instead of using all random walk which by nature is single-threaded, having another noise-based algorithm like Perlin noise, which can be done in parallel, change the path of the random walk it increases perceived randomness. This can reduce the performance hit and keep the high complexity. This can also help with the second issue of “randomness” in PCG, without complexity, an algorithm can become very similar ruining the experience for the player so with the incorporation of higher complexity the versatility will increase.

#### Diamond-Square Algorithm:

The Diamond-square algorithm is a vaguely unique PCG method, particularly because of its origin in computer graphics, with its primary purpose of simulating natural-looking terrains and height maps for structures like mountains, hills and flat rolling desserts(Archer, 2011). The algorithm excels at creating meticulous detail through the use of a subdivision process, alternating between “diamond” steps, where midpoint values are created, and “square” steps, where it adjusts the values of square midpoints, producing realistic topographically diverse landscapes.

Because of the method of subdivision, the algorithm manages complexity extremely well, breaking a large section of content into manageable steps, allowing for the simplification of complex landscape features into procedural tasks which can be parallelized efficiently(Archer, 2011). This also allows for a more simple algorithmic structure with a complex output. With the ability to make complex natural-looking terrains with simple rules, a simple change of rules creates the possibility for further versatility, like the ability to make other formations like cloud patterns, island formations and planetoids.

Diamond-Square algorithm's possibility of creating these cast visually appealing terrains can become a detriment to the very application as it has the possibility of creating impossible geographical structures limiting its ability to be used in simulations. Another issue comes with scalability even though the algorithm has its built-in ability to divide the problem up making it more manageable, larger landscapes require more processing power and the larger issue of memory because it divides tasks having all memory thread-safe is required.

Possible solutions to the issue of impossible geography are refinement of the rules, or having more than one version of the program to run after the other, creating a broad and narrow outlook. This can create the expected visual definition of the algorithm using the broad version of the algorithm and then move to the narrow version to create more fine details fixing any issue that may occur. To fix memory management issues once scaled higher would use dynamic allocation of memory for reuse of memory. And also mentioned because of its ability to split task hardware acceleration through CPU multithreading or GPU acceleration.

#### Conclusion:

In the world of PCG, exploration and application of algorithms like the ones researched above reveal endless possibilities and creativity. All the algorithms bring their strengths and weaknesses to the table, and overcoming the weaknesses through innovation and research thus broadening the horizons for other algorithms that follow.

Perlin Noise, with its natural-looking textures and terrains, addresses computational intensity and performance issues through techniques like multithreading and precomputation. These strategies ensure that the algorithm remains a versatile tool for creating dynamic, realistic environments without compromising on processing speed or quality.

Cellular Automata, celebrated for simulating complex systems from simple rules, mitigates challenges related to computation demand and memory management by focusing simulations on "alive" cells and employing efficient memory strategies. This approach allows for the simulation of intricate behaviors and environments with reduced computational overhead.

Random Walk, a method known for generating vast, intricate landscapes and structures, tackles its randomness and performance trade-offs by integrating with other algorithms such as Perlin Noise. This hybrid approach enhances complexity and variability, ensuring that generated content remains both high-quality and diverse.

The Diamond-Square Algorithm, adept at creating detailed terrains, overcomes scalability and realism challenges through rule refinement and dynamic memory allocation. By adjusting its subdivision process and leveraging hardware acceleration, it continues to produce realistic landscapes that are both extensive and detailed.

WFC, an algorithm known for its ability to generate bitmaps that are locally similar to a given input bitmap and simple and tileable grids of tiles to create rule-based worlds, stands to benefit greatly from the advancements and optimizations in the aforementioned PCG algorithms. The essence of WFC lies in its constraint-solving approach to content generation, where it attempts to maintain the coherence of the output concerning a set of input rules.

In conclusion, with the innovation and evolution of optimizations in the realm of PCG algorithms will not only help the specific algorithm but can be applied to other algorithms including WFC. By harnessing solutions to the limitations of other PCG algorithms, WFC can be advanced to a higher level of development to be used in large-scale productions. Offering new and unique player experiences by giving creators access to more solutions to the ever-growing production of PCG in video game creation.

### Previous Work:

#### WFC GitHub(maxim gumin):

##### Introduction:

Maxim Gumin not only developed the innovative PCG approach, he created a open-source implementation that demonstrates the algorithms versatility across mediums(Gumin, 2016). By creating a concrete example of how WFC can be utilized, he helped bridge the gap between theoretical and real-world application. With his development of the algorithm he stated his thought on theoretical issues like constrained synthesis, heuristics, Tilemap and bitmap generation and a few more development problems(Gumin, 2016). His application of the WFC algorithm develops the bitmap Pixel-Based generation method already discussed. Being the progenitor of the algorithm a large amount of his work will be used in other implementations of that algorithms.

##### Algorithm method:

1. Reading Input and Counting Patterns

Input Bitmap: The algorithm starts by reading an input bitmap image. This image serves as the sample from which the algorithm will generate larger patterns.

Count NxN Patterns: It counts all unique NxN patterns (tiles) in the input. N is a chosen parameter that defines the size of the patterns. This step may include counting all possible orientations and reflections of each pattern if augmentation is enabled, to enrich the pattern database.

2. Augmenting Pattern Data (Optional)

Rotations and Reflections: Optionally, the algorithm can augment the initial pattern data by including rotations (90°, 180°, 270°) and reflections (mirror images) of each pattern. This increases the variety of patterns that the algorithm can use in the generation process.

3. Initializing the Wave

Output Array ("Wave"): A wave array is created with dimensions corresponding to the desired output size. Each element of this array represents a superposition of NxN patterns from the input, indicated by boolean coefficients. True means the pattern is possible at that location, and False means it is forbidden.

Unobserved State Initialization: Initially, all coefficients are set to True, indicating that any pattern is possible everywhere.

4. Observation Step

Minimal Nonzero Entropy: The algorithm searches for a wave element (NxN region) with the minimal nonzero entropy, meaning the element with the least number of possible states, but more than one.

Collapse: This element is then collapsed into a definite state, chosen based on the distribution of patterns in the input. This means selecting one pattern from the possible options.

5. Propagation Step

Information Propagation: After a wave element is collapsed, the algorithm updates the state of neighbouring elements to reflect this change, forbidding patterns that no longer fit and adjusting the possibilities based on the new state.

6. Completion

Fully Observed or Contradictory State: The process of observation and propagation continues until every element of the wave is either in a fully observed state (with one possible pattern) or in a contradictory state (with no possible patterns). In the former case, the algorithm outputs the generated pattern. In the latter case, it ends without producing an output.

##### Problems:

In the development of the WFC algorithm mentions that there is a non-zero chance that all the possibilities for a pixel to become zero(Gumin, 2016). With a method called recursive backtracking, in which you keep track of previous states and in what order they were collapsed on the stack and just remove them from the top of the stack when failure is reached.

#### Nested Wave Function Collapse(N-WFC):

#### Introduction:

N-WFC introduces a hierarchical approach, nesting multiple WFC porches with a broader generation framework(Nie<i> et al.,</i> 2023). This structure allows for the division of larger-scale projects into manageable subsections, each processed through its own WFC procedure, thereby enhancing overall performance and flexibility(Nie<i> et al.,</i> 2023).

##### Algorithm Changes:

N-WFC introduces a hierarchical approach, nesting multiple WFC processes within a broader generation framework(Nie<i> et al.,</i> 2023). This structure allows for the division of large-scale projects into manageable sub-areas, each processed through its own WFC procedure, thereby enhancing overall performance and flexibility in content generation.

##### Possible changes:

By allowing for subsection, using a higher WFC algorithm to choose which next region to break down can allow for more efficient collapsation of regions(Nie<i> et al.,</i> 2023). If there are two regions of the same entropy in which they would not affect each other they can be done in parallel with the same outcome.

#### Stitched Wave Function Collapse(S-WFC):

##### Introduction:

S-WFC continues the idea of N-WFC, using the method of N-WFC of splitting the larger grid into subsections, however see that if you created a large amount of regions equally spaced the algorithm can come back and begin to “stitch” the regions back together(Watt Designs, 2023).

##### Algorithm Changes:

The obvious changes in the algorithm is allowing the unstitched region to not affect each other allowing for the parallelisation of this initial process, and with that, the direct section between 2 unstitched regions can be parallelised as well because all the sides should not affect each other(Watt Designs, 2023). That would leave the corners and once again, they should not be affected by each other meaning they can also be done in parallel. This process comes with the benefit of speed but comes with the downside of the increased need for backtracking as the possibilities decrease further as the steps are stepped through.

##### Possible Changes:

A larger change discussed in the video is that this production allows for the creation of multiple WFC algorithms with different tilesets, then a custom WFC algorithm with both tiles sets and custom rules allowing 2 different tilesets to be stitched together. This allows for more creativity and versatility to the already customizable WFC algorithm(Watt Designs, 2023).

### Conclusion:

With the exploration of PCG in video games alongside the concepts behind many staples of PCG including the WFC algorithm, underscores the pivotal role PCG has played in revolutionising games design and development(Hendrikx<i> et al.,</i> 2013). PCG facilitated creation of dynamic, vast and unique experiences and worlds. PCG has shown the potential to create intricate lore, ecosystems, and challenges in which can be adapted to every playthrough.

However, through PCG algorithms it illuminates the challenges inherent in balancing computational demands with creative ambitions(Hendrikx<i> et al.,</i> 2013). Constant Issues of scalability, memory management, and the complications that come with optimisation, the technical problems that come with the inherent need of computational speed, and yet many algorithms through the work of many people have found their way to the point of constant use in video games(Hendrikx<i> et al.,</i> 2013). With this development it allows for other PCG algorithms to follow suit, constantly driving forward innovation to push the boundaries of computation and function.

Through the use of subsection, parallelisation, heuristics and precaching development of PCG algorithms have come along and WFC being in the earlier stages of development could benefit from the the discussed optimisations. Specifically, subsections, parallelisation, and precaching in which this paper wishes to develop.

# Research Methodologies

## Research Methodology

This section outlines the systematic approach employed to investigate the performance of the Wave Function Collapse (WFC) algorithm under varying complexities and dimensions and the possible optimisations. The methodology ensures a robust and replicable framework for assessing the algorithm's scalability and efficiency in procedural content generation within a controlled environment. All data will be quantitative, with removed values only comparing between data to remove the worry of variance between hardware.

## Framework Development

### Baseline Establishment:

A foundational WFC algorithm is developed to serve as the baseline for all experiments. This ensures consistency in the core algorithmic structure, allowing for the isolated examination of performance impacts due to the varied optimisations proposed.

### Program Structure:

All programs are run multiple times to get an average to remove outliers of data, to further develop this a percentage will be taken out of the sampling as they are to be removed from real data points.

## Complexity and Dimension Parameters

### Initial Tile Set:

The experiment begins with an initial tile set designed for simplicity, featuring matching tiles with only 2-3 exits for 2D configurations and 2-5 exits for 3D configurations. This setting provides a starting point for complexity escalation allowing for failure and the need for backtracking to represent what a real-life scenario might look like.

### Complexity Escalation:

starting at only 2 types of exits complexity is systematically increased by introducing an additional socket type on the exits, progressing from 2 up to 5 types. This stepwise increase simulates real-world application scenarios with varying degrees of complexity.

## Experimental Setup

### Spatial Configurations:

The study benchmarks both 2D planes and 3D grids, with initial grid sizes set at a minimum of 5x5 for 2D increasing in size following squares, the 3D grids will follow the pattern but cubed instead. The size will increment in 5 till the dimensions reach 25.

### Execution Runs:

Each configuration is subjected to a minimum of 10 execution runs, assuming each run completes within 5 minutes. To account for variability, the top and bottom 10% of results are removed as outliers.

### Performance Analysis:

The average performance across runs, excluding outliers, is calculated to determine the algorithm's efficiency. This analysis focuses on the average time per tile placement, offering insights into scalability across different complexities.

## Technical Environment

### Development Platform:

The Unity game engine, chosen for its flexibility and widespread use in game development, hosts the experimental framework. The experiments proceed without the burst compiler or GPU acceleration to standardize testing conditions.

### Visual Simplification:

To eliminate potential bottlenecks associated with mesh generation and initialization, the study utilizes simple blocks and colours for tile representation. Rendering is disabled during test runs to focus purely on algorithmic performance, with post-test rendering enabled for outcome verification.

### Hardware Specifications:

Tests are conducted on a system equipped with an Intel i7-11700 processor, 32GB of RAM at 3200 MHz and an RTX 3080 12GB, ensuring a high-performance baseline suitable for the computational demands of the study.

## Data Collection and Analysis

### Comparative Visualization:

Results from different techniques and configurations are graphically represented on a unified chart. This visualisation facilitates a direct comparison of performance impacts due to complexity and dimension variations.

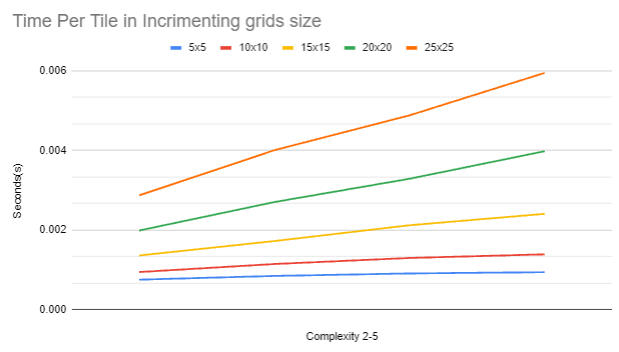
### Optimization Exploration:

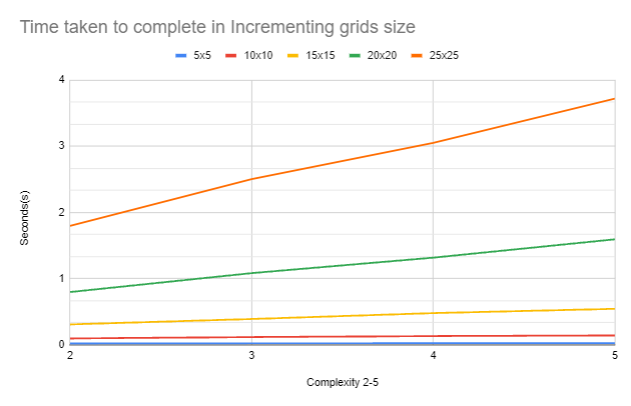
The study aims to identify optimal configurations and variables that influence technique efficiency. Additional experimentation when a subsection is possible will be conducted to discover the optimal subsection size following the same methods. Through experimentation, the research seeks to uncover grouping strategies that maximise performance under the tested conditions.

## Conclusion

This methodology ensures a comprehensive and methodical exploration of the many WFC algorithms' performance, providing a framework for understanding its scalability and efficiency in procedural content generation. The structured approach, from baseline development to complex scenario testing and performance analysis, lays the groundwork for significant insights into the algorithm's application in game development and beyond.

# Results and Findings

(Fig 1)

(Fig 2)

(Fig 3)

A graph of different colored bars

Description automatically generated with medium confidence

In Fig 1 as the data shows that in increasing grid sizes if you simultaneously increase the complexity the speed of generation per tile will begin to slow, compared to the lowest size grid 5x5 the 25x25 grid takes large steps in decreasing speed.

In Fig 2 as the data shows that as the idea that having an increased sized grid size will change time of completion assuming similar complexities.

In Fig 3 spreading the data into the complexity only the data shows are the grid sizes increase the time per tile increases at a larger rate.

In the complete data set in Appendix 1, the difference between 100% to 80% is shown to be no more than different from the 15% difference. All of this data can be used in the next steps of WFC to determine what would be best for the subsection.

With the use of Figs 1-3 the data supports that using 5x1x5 sub-grid will be the more paramount option as to build higher level girds. Both through speed and less variability.

As shown in the

# Discussion and Analysis

***How has the project gone?***

## Interpretation of the Findings:

The investigation has demonstrated that optimizing the WFC algorithm through nesting and stitching strategies significantly enhances its scalability, efficiency and without compromising the quality of procedural generation. These findings align with the initial hypothesis that transitioning from a complete grid system to a subsection system with particularly its computational demands and serial processing bottleneck.

## Achievement of Objectives

The study achieved its primary aim by critically evaluating the performance of the standard WFC algorithm and assessing the effectiveness of the proposed optimizations. The results indicate that both nesting and stitching optimizations offer substantial improvements in processing speed and scalability. This supports the objective of transitioning WFC from a theoretically interesting method to a practical tool in game development.

## Support and Contradiction of Existing Literature

The findings corroborate the assertions by Nie et al. (Nie<i> et al.,</i> 2023) and Watt Designs (Watt Designs, 2023) about the potential benefits of parallelized processing in WFC. However, they also challenge some existing beliefs about the necessity of preserving the serial integrity of the WFC algorithm for coherence and quality of output, suggesting that with careful design, parallel processing can maintain to a large extent.

## Plausible Explanations for Findings

The observed improvements in algorithm performance can be attributed to the inherent advantages of grid division, which allows for simpler computations to be done. The nested and stitched approaches effectively decompose the problem space into smaller, manageable units, which are processed independently, leveraging multi-core processing capabilities to reduce overall computation time.

## Possible Criticisms of the Investigation

One potential criticism could be the reliance on Unity as the development platform, which may not be the most optimized environment for algorithmic research. Additionally, the complexity and variability of procedural content generation tasks mean that results might vary under different conditions or with different input parameters, possibly limiting the generalizability of the findings.

## Conditions Affecting Results

Under different conditions, such as with more complex input rules or larger grid sizes, the benefits of the optimizations might vary. Exploring these variables proved deeper insights like the reliance on smaller entropy counts and developed into discussing the possibility of further limitations of the scalability and adaptability of the optimizations.

# Conclusion

This discussion highlights the significant potential of nested and stitched WFC optimizations for enhancing the efficiency and scalability of procedural content generation in game development. While the study supports the feasibility and benefits of these approaches, it also opens avenues for further research, particularly in exploring the impact of different computational environments and more complex procedural generation scenarios. The findings represent a step forward in making WFC a more viable and practical tool for game developers, promising richer and more diverse gaming experiences.

# Recommendations

Optimization and Language Choice: The recommendation highlights that choosing a programming language or environment that is more amenable to optimization could yield better results for similar research. While Unity was chosen for its ability to produce visual outputs that facilitate easier debugging, it's suggested that other environments might offer superior optimization opportunities, leading to enhanced performance.

Use of Unity: The decision to use Unity was primarily for the convenience of visual debugging. This choice was not driven by performance considerations but by the ease of debugging and visual analysis, it offers.

Research Focus: The research was not primarily concerned with absolute performance but with relative performance improvements, which are measured in percentage gains. This approach allows for the evaluation of enhancements without necessitating the most optimized environment for the initial research.

Precaching and Tile Sets: A significant concept discussed is the idea of precaching or precomputing parts of larger tile sets, akin to techniques used in generating Perlin noise. By running a version of the Wave Function Collapse (WFC) algorithm to generate smaller grids that can then be assembled into a larger grid, it's possible to increase the efficiency of the algorithms involved. This approach could enable the creation of larger grids at the same computational speed, potentially speeding up the entire process.

In summary, the recommendations suggest that while Unity provides a convenient platform for visual debugging and development, future research might achieve better performance by selecting a more optimization-friendly environment. Additionally, leveraging techniques like precaching and using smaller precomputed grids to construct larger ones could further enhance the speed and efficiency of the computational algorithms being researched.

# References

A.I. Design. (1980) *Rogue*[0].

Adams, C. and Louis, S. (2017) *Procedural maze level generation with evolutionary cellular automata.*IEEE, pp. 1.

Archer, T. (2011) *Procedurally generating terrain.*pp. 378.

Awiszus, M., Schubert, F. and Rosenhahn, B. (2021) *World-gan: a generative model for minecraft worlds.*IEEE, pp. 1.

Baron, J.R. (2017) *Procedural dungeon generation analysis and adaptation.*pp. 168.

Edge Staff (2009) *The Making Of: Rogue.*Available at: [https://web.archive.org/web/20120815191124/http://www.edge-online.com/features/making-rogue](https://web.archive.org/web/20120815191124/http:/www.edge-online.com/features/making-rogue)

GridBugs (2019) *Procedural Generation with Wave Function Collapse.*Available at: <https://www.gridbugs.org/wave-function-collapse/>

Gumin, M. (2016) 'WaveFunctionCollapse', *GitHub repository,*.

Hart, J.C. (2001) *Perlin noise pixel shaders.*pp. 87.

Hendrikx, M., Meijer, S., Van Der Velden, J. and Iosup, A. (2013) 'Procedural content generation for games', *ACM transactions on multimedia computing communications and applications,*9(1), pp. 1-22 Available at: 10.1145/2422956.2422957.

Kim, H., Lee, S., Lee, H., Hahn, T. and Kang, S. (2019) *Automatic generation of game content using a graph-based wave function collapse algorithm.*IEEE, pp. 1.

Levi Winslow (2023) *God Of War: Valhalla DLC Just Got Harder, With One Big Exception.*Available at: <https://kotaku.com/god-of-war-ragnarok-valhalla-update-0503-patch-notes-1851108886>

Lioret, A., Ruche, N., Gibiat, E. and Chopin, C. (2022) 'GAN applied to Wave Function Collapse for procedural map generation', *ACM SIGGRAPH 2022 Posters,*, pp. 1-2 Available at: 10.1145/3532719.3543198.

Martin, P. (2012) 'Landscape and gamescape in Dwarf Fortress', *Proc.Philosophy of Computer Games,*.

Mojang Studios and Markus Persson. (2011) *Minecraft*[0]. Available at: (Downloaded: .

Nie, Y., Zheng, S., Zhuang, Z. and Song, X. (2023) *Extend Wave Function Collapse Algorithm to Large-Scale Content Generation.*IEEE, pp. 1.

Togelius, J., Yannakakis, G.N., Stanley, K.O. and Browne, C. (2011) 'Search-Based Procedural Content Generation: A Taxonomy and Survey', *IEEE transactions on computational intelligence and AI in games.,*3(3), pp. 172-186 Available at: 10.1109/TCIAIG.2011.2148116.

*A new way to generate worlds (stitched WFC)*(2023) Directed by Watt Designs. Available at: <https://www.youtube.com/watch?v=dFYMOzoSDNE>

# Appendices

## Appendix 1 – 2D WFC values



## Appendix 2 – 2D Nested-WFC values



## Appendix 3 – 3D WFC values



## Appendix 3 – 3D Nested-WFC values

