DELFT UNIVERSITY OF TECHNOLOGY

MASTERS THESIS

Designer Empowerment through mixed-initiative Wave Function Collapse

Author:
Thijmen S.L. LANGENDAM

Supervisor:
Dr. ir. Rafael BIDARRA

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Computer Graphics and Visualization Group
Software Technology

September 2, 2022

"The highest forms of understanding we can achieve are laughter and human compassion."

Richard Feynman

"When people think about computer science, they imagine people with pocket protectors and thick glasses who code all night."

Marissa Mayer

"Random numbers should not be generated with a method chosen at random."

Donald Knuth

DELFT UNIVERSITY OF TECHNOLOGY

Abstract

Electrical Engineering, Mathematics and Computer Science

Software Technology

Master of Science

Designer Empowerment through mixed-initiative Wave Function Collapse

by Thijmen S.L. LANGENDAM

This research has been performed in pursuit of the MSc in Computer Science at Delft University of Technology.

Wave Function Collapse (WFC) is a powerful generative algorithm, able to create locally-similar output based on a single example input. One of the inherent limitations of the original WFC is that it often requires users to understand its inner workings, and possibly make their own ad-hoc modifications to achieve satisfactory results. Besides being distracting from your creative task, this strongly reduces the algorithm's effective usefulness to a small group of technical users.

We propose a novel mixed-initiative approach to WFC, aimed at overcoming these drawbacks. These methods focus on providing intuitive control to its users, in a way that matches their usual creative workflow.

Among its main features, our approach provides (i) interactive navigation through design history, including controlled backtracking, (ii) precise manual editing of the output for direct expression of design intent, (iii) interactive manipulation of the tiles, and (iv) an initial layered approach towards post-processing algorithm output. These methods combined provide the required tools for users to tweak the global appearance of the output.

We evaluated a prototype implementation of our approach among game artists and other creative professionals, and concluded that its features were largely considered intuitive and supportive to express their creative intent.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. ir. Rafael Bidarra, for the support during my bachelors, with the many courses he has taught, as well as during my thesis, with his patience and thousands of lines of feedback, knowledge, and guidance.

Besides my supervisor, I would like to thank the participants of my user tests, for their time and effort they have put into the completion of the questions asked and tasks provided, without which I would not have been able to properly evaluate my work.

I would like to express my gratitude towards Stephen Monaco, for the extensive explanations of the Avalonia Framework, used in this thesis, without which, the developed method for validation, the miWFC application, would've been harder to maintain and less functional.

I would also like to thank my parents in particular, Rob and Gina Langendam, who enjoyed the ups, and have helped me through the downs during this project. Supporting me throughout and have kept me moving forward after encountering setbacks and frustrations.

Finally, I would like to thank my family and my friends, both from the university and from home, who have provided me with countless accounts of moral support, as well as invested interest into this research which motivated me towards this milestone in my life.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Most game level designers and artists do hard and unstructured creative work. Procedural Content Generation (PCG) methods have often been proposed as a powerful tool to assist them [33]. However, to be effectively helpful, such means have to empower artistic users, not only respecting but amplifying their creative freedom [32].

Mixed-initiative techniques propose a type of human-computer interaction in which the computer and the human user alternatively take steps towards the desired goal. Mixed-initiative PCG systems have long been proposed as promising tools for a variety of purposes in game development, from game level [1, 19, 36, 43] to complete game world generation [14, 28, 35]. However, the challenges of combining PCG with manual editing of its output have been pointed out by Smelik et al. [34].

The underlying issue is that the potential for human creativity is frequently being limited, since users (such as designers or players) do not always have the essential creative tools at their disposal [44]. Mixed-initiative interaction methods convert the system into an autonomous computational system that, with the help of human input, explores the possibility space in unique ways, and provides tools to control or restrict its processes [34].

Wave Function Collapse (WFC) is a PCG method that has recently gained widespread popularity [11, 15]. The original WFC partly resembles the model synthesis work of Paul Merrell [21], which was initially geared towards procedurally generating complex 3D models based on one input model. WFC simplified and facilitated its use and application for image synthesis purposes. However, this comes at the cost of providing little control on the generative direction followed by the algorithm.

1.1 Research Questions

This thesis analyses and explores how to adapt the WFC algorithm to support and integrate a number of interactive methods that more appropriately suit the usual creative workflow of game level designers and artists. We discuss the workings of methods designed and present results of the evaluation of their implementation in a prototype application, and aim to answer the following research questions through our evaluation:

Research Question 1: *How can the Wave Function Collapse algorithm and its advanced features be made usable for creative professionals in a responsive and intuitive way, to explore a generative space?*

Research Question 2: *What does it take to convert the Wave Function Collapse algorithm into a mixed-initiative PCG method?*

The first question aims at providing a solution to improving the currently unstructured work of creative professionals, as WFC is a powerful PCG method and a promising basis for creative design.

The second question assists answering the first question, as we believe mixed-initiative WFC is a solution to making its advanced features available. We approach this question by defining methods, and unfolding requirements that make these methods usable, intuitive and effective.

1.2 Mixed-Initiative Interaction

Mixed-initiative interaction provides users with more control, as users themselves collaborate with the computer, working together to take steps towards the desired goal [44, 45]. This idea is noticeably useful for creative professionals, and we want to promote user participation into the creation of the output, keeping the user in the loop.

A few implementations of mixed-initiative approaches for creative design currently exist, and some are prime examples of effectiveness when a mixed-initiative approach is introduced to PCG systems:

Sentient Sketchbook [19] iteratively generates a world level for a strategic game setting. The user is first able to sketch a map in a low-resolution (high-level of abstraction). This is analysed, and the application provides real-time feedback on multiple criteria such as playability and balance to the user. The application then suggests alternatives that are similar to the current input and potentially improve scores defined by criteria.

Actions used in Sketchaworld [35] each apply a different unique set of tasks when they are performed. The user chooses a procedural method and enters the necessary information. The world is changed in accordance with the input using the selected procedural method. This reduces the user's need to perform several tasks to a single activity that they can manage themselves.

Tanagra [36] is a tool for 2D level design. The user is able to put desired features into the world, whether it be physical features or criteria such as play through speed of certain sections in the level. The tool directly adapts the world around it in order to maintain playability. Similar to Sentient Sketchbook [19], the tool is able to provide a multitude of designs to choose from as replacements.

The Evolutionary Dungeon Designer (EDD) [1] introduced Interactive Constrained MAP-Elites [12] for dungeon design. This design tool allows users to create dungeons that dynamically and flexibly select appropriate dimensions of variety. The tool proposes improvement ideas that fit into the user designed level. If chosen, these improvements are iteratively fed back into the tool to further develop propositions for the level.

TaleForge [28] is a tool which, through the suggestion of relevant items to users, helps designers build a narrative world for a particular given story. The tool suggests content through previous knowledge and inter-story associations.

RL Brush (Reinforcement Learning Brush) [6] enhances user-input through suggestions via artificial intelligence. The tool has a selection of agents that each suggest enhancements to the user designed level, or combines other agents to create an aggregate suggestion. The user is able to influence agent suggestions through parameters such as the area visible to agents for suggestions, or the amount of recursive iterations.

Questgram [2] is a prototype tool that extends EDD [1] for the generation of quests using grammars. The tool allows users to choose quests that combine to fulfil an objective. Additionally, it aids the user through suggestions of which quest actions to choose from, based on created dungeons.

Similarly, research by R. van der Linden et al. [42], designed a dungeon generator which converts action graphs, generated by a gameplay grammar, into dungeon levels, allowing users to first design the most important features, around which the generator designs a valid dungeon.

1.3 Thesis Approach

Most creative professionals are not served by the lack of control offered by the original WFC algorithm or its variants. Our work aims at converting WFC into a powerful mixed-initiative creativity tool that makes it much more accessible and usable to a wider, non-technical public. In particular, we address the algorithm's lack of control over each step. We investigate solutions such that WFC can effectively support artists in freely expressing their intent, and exploring the design space. Naturally, this should be done without compromising algorithm strengths.

A typical use case of mixed-initiative WFC might consist of an artist initiating the output by manually creating a specific area of interest at a desired location (something the algorithm would typically not produce on its own), and then utilize the algorithm to fill in remaining areas (Figure 1.1). Such a facility is a good example of assisting users in steering the output in the desired direction. Allowing such manual control at any stage promotes the kind of designer empowerment that drives our research contribution.

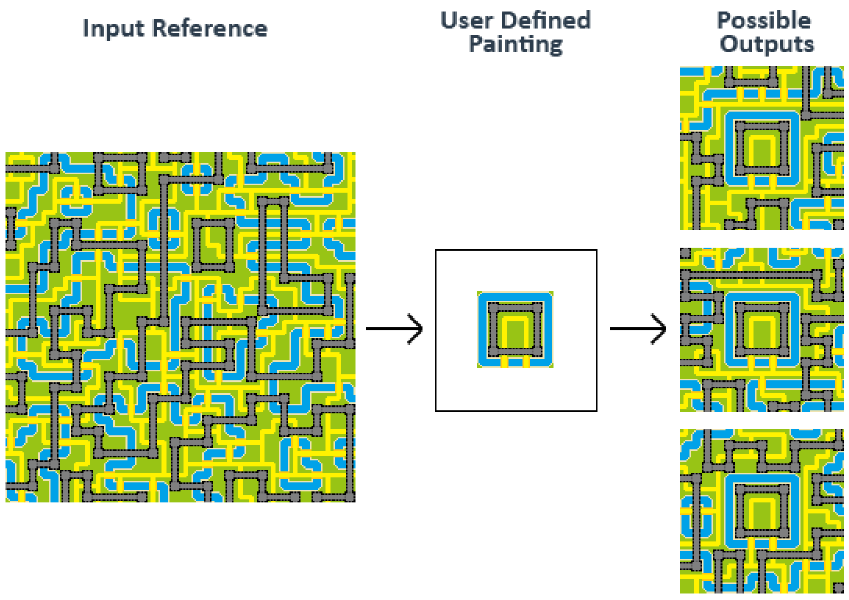


FIGURE 1.1: Manual drawing before running WFC.
(Input image: Castle [11])

2 Background

We discuss the original WFC algorithm, its strengths and weaknesses, as well as related research work, such as modifications and extensions to the original algorithm.

2.1 Wave Function Collapse Algorithm

WFC has several similarities to Paul Merrell's model synthesis work [21], but differs in using a different selection heuristic, to decide where and what to place. The original WFC algorithm [11] generates images locally similar to the input, meaning the output only contains clusters and combinations of pixels present in the input.

To generalize the WFC algorithm, it is convenient to first introduce a few notions as follows:

- A **tile** is a distinct combination of elementary 'space subdivision units' (e.g., pixels, voxels, letters, etc.), and is identified and extracted from the input.
- A **cell** is the basic building block of the algorithm output space; initially, any tile can potentially be assigned to each cell, hence the (quite remote) quantum analogy of WFC: every cell 'simultaneously contains all possible states' (i.e., potential tiles) until you either 'collapse' it (i.e., assign it one concrete tile) or its neighbour cells constrain its allowed 'states'.
- A **pattern** is a unique local configuration of tiles, used in the overlapping model (Section 2.1.2). In this model, constraints are inferred from the input (e.g., an image, constructing a set of unique NxN patterns) [15].

M. Gumin [11] used terms such as “wave function”, “observe” and “collapse” due to the loose connection with the *quantum mechanics wave function collapse*[1]. Here, the *wave function* is in a superposition of all states, which gets reduced to a single state when interacting with the external world (*observing*[2]).

WFC is additionally related to cellular automata (Johnson et al. [13]) as both algorithms have cells influenced by neighbours and randomness. They vary in picking mechanisms: cellular automata updates all cells simultaneously, and repeatedly for a number of iterations, having an exportable image at every step. WFC visits each cell one by one, until the output has been generated in full.

The generic WFC algorithm, see Algorithm 1, initially analyses its input, detecting and extracting from it both a number of tiles (and patterns in the overlapping model) and identifying constraints (e.g., existing adjacencies) between them (1). The output is then initialized fully “unobserved”, meaning that every cell contains all possible states (2). Subsequently, it will iteratively select where to continue (4), what tile to choose (5), and propagate this decision to neighbouring cells (6). Eventually, it

[1]`https://en.wikipedia.org/wiki/Wave_function_collapse`
[2]`https://en.wikipedia.org/wiki/Observer_effect_(physics)`

stops (3) when either all cells have been assigned a concrete value, or some conflict took place. Steps 4 and 5 combine to assign a new value to a cell, also referred to as "observing". Collapsing can originate from both observations and propagations, and is the reduction of possible states to a single state. A full description of the default algorithm can be defined as follows:

Algorithm 1 Generic WFC algorithm

```
1: Process input, find tile relations, store weights based on occurrence
2: Initialize output unobserved
3: while not done or until a conflict arose do
4:     Find cell to collapse                                          ▷ Observation - Step 1
5:     Assign a tile to the selected cell, based on weights           ▷ Observation - Step 2
6:     Propagate collapsed cell to neighbours until no changes occur  ▷ Propagation
7: end while
```

Propagation of the WFC Algorithm can be described as a set of nested loops. The observation of a tile at a cell is communicated to the direct neighbours of this cell, which in turn act upon this observation if influenced, reducing their own "wave function" (i.e., set of allowed states). If altered, this neighbouring cell communicates its change to its own neighbours, and so forth, until no more changes occur and the output space has come to a stable resting state.

2.1.1 Adjacent Model

The adjacent or simple tiled model, is the basic implementation of the algorithm. In this model, input and output are built out of pre-defined tiles. Rules or constraints, used to decide what may be placed where, are provided manually or through a sample input, from which it will generate a set of constraints (Figure 2.1).

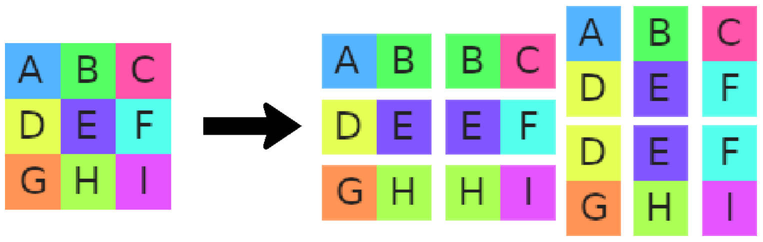


FIGURE 2.1: Constraints extracted from the input by the adjacent model. [3]

Constraints only reference direct neighbours in four cardinal directions (N, E, S and W), hence, no tile has any direct influence on tiles positioned two or more cells away. Additionally, each tile has a weight, influencing the chance of being selected. These weights are manually provided or read from the input sample based on frequency of appearance.

2.1.2 Overlapping Model

The overlapping model is a significant step up from the adjacent model. As described by Gumin, *"The overlapping model relates to the simple tiled model the same way higher order Markov chains relate to order one Markov chains."* [11].

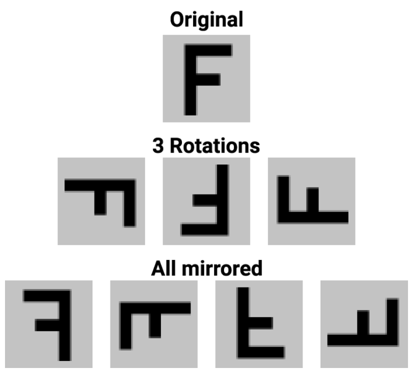


FIGURE 2.2: Transformations of a pattern.

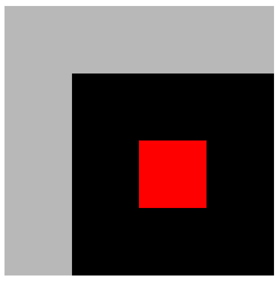


FIGURE 2.3: RedMaze [11], colours adapted.

Cells are assigned a value, such as a colour, based on the pattern formed by the cell itself and its neighbours. For this model, an extra parameter is introduced called the *pattern size*, which defines the amount of tiles in a pattern when processing the input. To obtain patterns, the input is scanned using this defined pattern size N, extracting N x N sized patterns. Similar to the adjacent model, weights are calculated by counting the frequency of each pattern in the input image, including its possible transformations.

Patterns house their own constraints, as two neighbouring patterns must match where they overlap. Patterns can be rotated and flipped to create additional patterns (Figure 2.2). Whether two patterns are allowed to overlap, is pre-calculated during initialization based on the horizontal and vertical offset (x, y) of the two patterns. The amount of overlapping orientations is a function of the pattern size N: $(2(N-1)+1)^2$ (example shown in Figure 2.4).

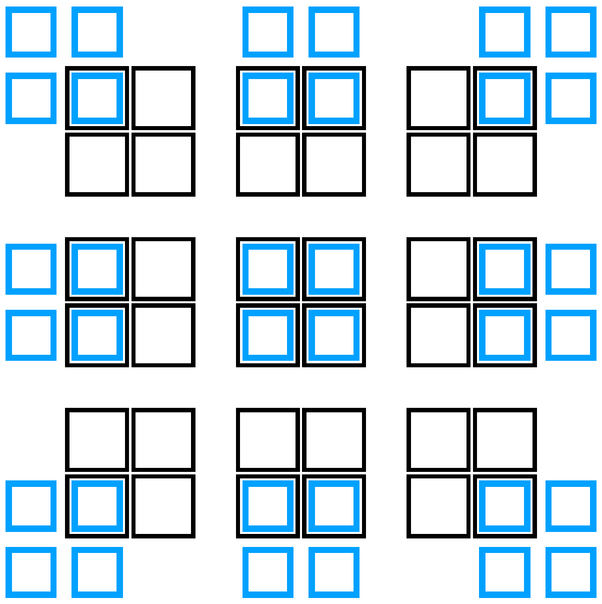


FIGURE 2.4: The 9 overlapping offsets for $N = 2$ [15].

For each pattern, we store for each other pattern whether they are able to overlap, and for which offset. An example of these pre-calculations for the first pattern of the RedMaze (Figure 2.3) input image is shown in Figure 2.5, with the highlighted (outlined in blue) pattern used as an example in the visualization for each overlapping offset (Figure 2.4).

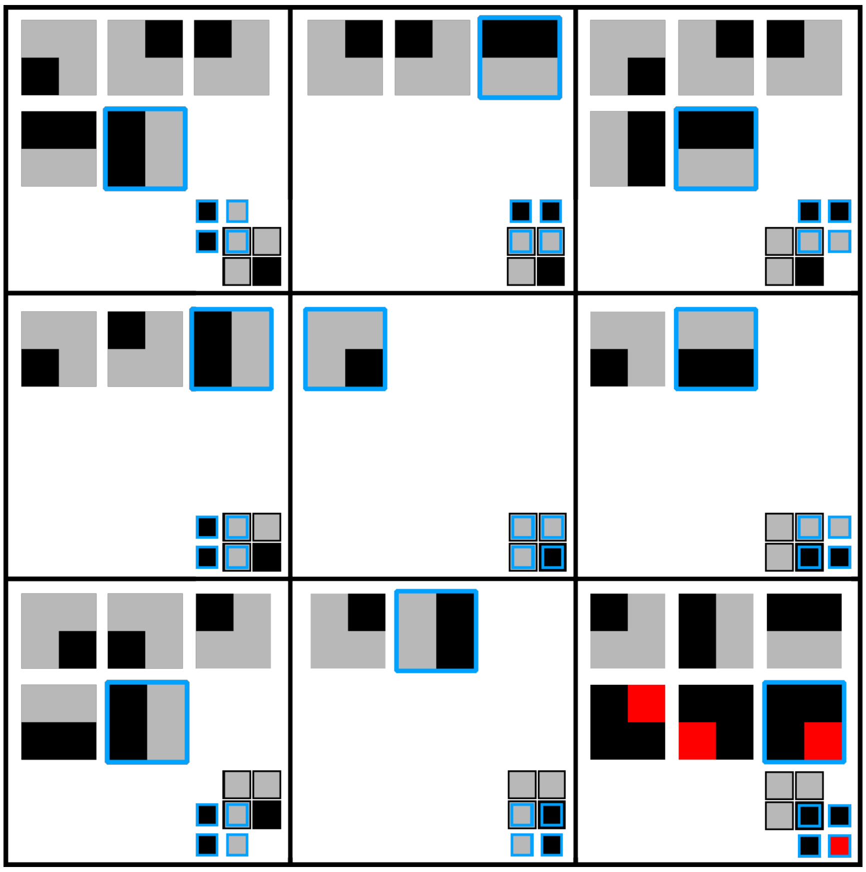


FIGURE 2.5: Allowed neighbouring patterns [15].

Entropy

WFC chooses which cell to observe based on a property called entropy as it iterates across the generating space. A cell with a high entropy has numerous potential states available, and a low entropy indicates a more constrained cell. Entropy is calculated with the Shannon Entropy [7], with parameter w_i being the weight of the tile with index i:

$$entropy = log(\sum w_i) - \frac{\sum(w_i * log(w_i))}{\sum w_i}$$

At each iteration, WFC chooses to observe the cell with the lowest entropy, naturally having the least amount of states available. This method was chosen to minimize conflicts during output generation, as well as to conform to a "humanly" approach; *"I noticed that when humans draw something they often follow the minimal entropy heuristic themselves. That's why the algorithm is so enjoyable to watch"* [11].

2.2 Algorithm Strengths and Weaknesses

WFC's simplicity makes it quite attractive for being applied to generative problems in many domains. Among its strongest advantages, one can point out:

- The algorithm requires a single input, which makes it faster than machine learning approaches that require substantial training data as well as training time and computational power.
- Output can often be generated in milliseconds, giving users quick feedback.

- The iterative nature of the algorithm allows for interception or manual pause at any stage.
- WFC is considerably customizable, allowing for modification and extension at almost any point to shape it to your requirements.

Unfortunately, WFC also has a number of drawbacks, some of which strongly hinder artists in their creative work. Among them, one can point out:

- Notions like 'collapsing a cell', 'entropy-based selection' and 'constraint propagation' are non-trivial, and require understanding to grasp why some output is as it is.
- WFC originally lacks an undo facility, which in turn excludes a trial and error approach, frequently taken by artists. Likewise, WFC is unable to backtrack, a much-needed option for restarting, for example, after a conflict occurs, or when some of the output is not conforming to the user's design goal.
- Propagation is not always logical or obvious: whilst conceptually easy to understand, it all happens behind the scenes, which may often be confusing.
- Tile selection is not easily controllable, because after picking a cell to collapse, tile choice is based on fixed weights, extracted from the algorithm input.
- The algorithm requires intricate understanding for non-default input images to be used and consequently yield desired output.

2.3 Related Work

In this section, we describe work related to WFC, such as current implementations, extensions and variations.

Karth and Smith [15] make a good analysis of the WFC algorithm, placing it in continuity with previous methods: "Both traditional texture synthesis and Markov chain approaches are primarily data-driven and thus accessible to non-programmers.". One can therefore wonder: if these two techniques were made accessible to non-programmers, how could that be done with regard to WFC?

The same authors also discussed how to make the algorithm faster, more efficient and complete [16]. Regarding completeness, one of the answers is the inclusion of backtracking, as it helps cover the generative space, eventually leading to find a solution, if it exists.

To address more flexibility in input usage, Sandhu et al. [31] propose dynamically adjusting tile weights during algorithm execution, which has potential to lead to more satisfactory output, at least if properly controllable. Additionally, they take steps towards adding non-tile concepts to the generation such as items, through a multi-layer WFC approach.

Cheng et al. [4] implemented three mechanisms into WFC: global constraints, multi-layer generation and distance constraints, to provide a base for non-local constraints, inspired by Sandhu et al. [31]. These extensions were added for improved designer control, better playability, and increased similarity to human-designed levels.

Kim and Kang further explore the possibilities of WFC by extending it into a graph-based domain, moving away from a simple grid layout [17]. A graph-based domain allows for a broader definition of combination rules, and ultimately to the generation of a much larger variety of content (from maps to Sudoku levels). However, this comes at the cost of a much less intuitive interaction.

Billeskov and Møller [23] combine WFC with Growing Grids [9] on irregular quadrilateral grids. Growing grids allows shaping of grids to any desired shape, aiming to improve WFC's diversity, level difficulty controllability and traversability. Traversability was solved through chiseling [41], and difficulty was significantly increased as the implementation strayed away from default straight grids. Finally, diversity was improved through growing grids itself.

Newgas created a library and tool, *Tessera* [24], for constraint-based procedural generation. It contains a 'painting tool' for drawing tile adjacencies (rather than extraction from the input), and a number of extensions such as working with pre-placed tiles, tiles spanning multiple cells, more grid types and path graphing constraints. Through these additions, they aim to improve algorithm configurability. In addition, they have made available his WFC C# Library [3], extending the original algorithm with several additional features, including backtracking and non-grid layouts. Our *miWFC*prototype builds upon this library.

Lin et al. [20] take advantage of WFC and Convolutional Neural Networks to design urban spaces with fast prototyping. Here, WFC is used to generate road networks, of which a suitable one is picked, and urban blocks subsequentially extracted by CNNs. Although accuracy was limited (caused by hardware and CNN algorithm), WFC proved to be suitable for urban space synthesis.

Two games developed by Oskar Stålberg [38] are good examples of online PCG[3] through an ad-hoc version of WFC: *Bad North* [39] and *Townscaper* [40]. Bad North customizes basic WFC with an extra constraint which makes sure that agents in the game are able to have a navigable path across levels.

[3]Meaning the algorithm is used during gameplay, rather than using it to generate content before execution.

3 Methods

This chapter introduces and describes designed methods which convert WFC into a mixed-initiative PCG method. Through extension of the default algorithm with advanced features and user-centred design approaches, we solve algorithm drawbacks and provide more intuitive and user-friendly control.

3.1 History Navigation

In the generic algorithm described in Section 2.1 (Algorithm 1), selecting a cell (4), collapsing it to a single state (5), and the subsequent propagation (6), are deterministic processes: these steps always result in the same state, and are therefore revertible. Additionally, the iterative nature of WFC provides a clear and discrete measure for control in stepping through the process, both in its basic forward (generative) direction, and its backward (undo) direction.

To extend controlled stepping through time, allowing to save progress at the current state of the process, and the ability to revert to this state, further lowers the amount of steps required in a "trial and error" approach of converging to a desired output.

By introducing backtracking and manual stepping through the WFC algorithm, and by conveniently bringing these features to an intuitive interface, we allow for more and finer control over the generative process. Because these features relate to the timeline dimension of the algorithm, we call them *History navigation*. This definition also provides the possibility of presenting the user with a timeline to visualize the entire generative process of the algorithm.

The addition of history navigation is the first and most essential step towards improving controllability of the algorithm, being the base for the other methods. These elements work together to address and solve the absence of an undo facility. As a result, the algorithm's process becomes more visible, and its control more accurate and intuitive.

3.1.1 Backtracking

Backtracking focuses on steps 4 and 5 of the algorithm (Algorithm 1), which are deterministic, yet the original implementation does not provide the ability to revert these steps. The issue with non reversibility, is that the algorithm has to restart output generation from scratch, resetting the output by discarding all progress.

This approach is inherently inconvenient for multiple reasons. First and foremost, it takes unnecessary time to generate output, as conflicting decisions require a full new generation. Secondly, users are unable to undo. If they have a section of the output they like, conflicts will wipe the slate clean. Hence, it forces users to apply trial and error at a global level, and provides no control over the output except for providing different input.

In Figure 3.1, the algorithm has three tiles: grass, a straight road and a crossroads. The original algorithm would simply restart after encountering a conflict (bottom branch), whilst with backtracking, we undo the last step and continue from that point, eventually getting a valid solution (top branch).

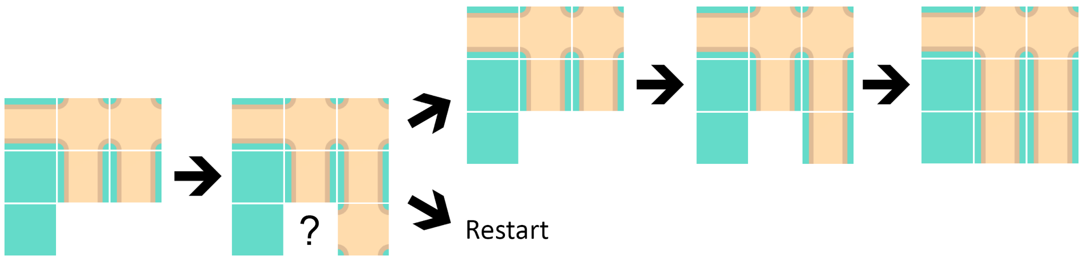


FIGURE 3.1: Process with and without backtracking.

Backtracking keeps track of every step taken. If a conflict (or contradiction) is encountered, we know what to undo to go back to the state prior to the conflict. Alteration of the generic WFC algorithm is required to incorporate backtracking (Algorithm 2). We introduce two stacks to keep track of our decisions. Stack OS has all cell and tile combinations observed, and stack PS contains all propagated decisions.

These stacks can be visualized using Figure 3.2. Each observation (blue) made is pushed to the OS stack (6), with all propagations caused by observation (red) being pushed to the PS stack (8). Then, if any conflict should occur, we backtrack (9 to 11).

Algorithm 2 Generic WFC algorithm with backtracking

```
1: Process input, find tile relations, store weights based on occurrence
2: Initialize output unobserved → Every cell contains all possible states
3: while not done or until a conflict arose do
4:     Find cell $c$ with the lowest entropy
5:     Assign tile $t$ to the selected cell $c$ based on tile weights
6:     Push $(c, t)$ to $OS$
7:     Propagate collapsed cell to neighbours until no changes occur
8:     Push all propagated $(c_p, t_p)$ to $PS$
9:     if conflict at any step then
10:        Backtrack                          ▷ Undo progress caused by observing $t$ at $c$
11:    end if
12: end while
```

We now keep track of all decisions made, and this is done to allow reverting the process one step at a time without causing additional issues. If any conflict arises at any point, we undo the steps taken in this iteration.

Backtracking itself is described in Algorithm 3: we first undo the propagations caused by the last observation (1 to 3). In the WFC algorithm, propagation is performed after observation, hence we undo in the opposite order. We remove (or "ban", 6) the observed tile at the cell it was previously chosen at, as this was the source of our conflict, and ensures this decision does not surface again.

With backtracking, users can now apply trial and error on a more local scale, removing the necessity of restarting upon encountering conflicts.

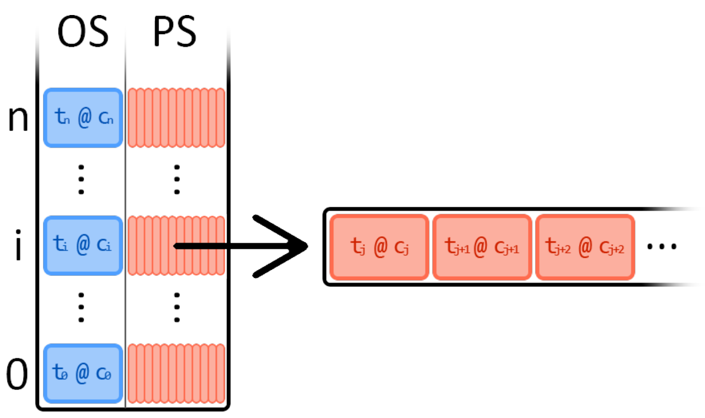


FIGURE 3.2: Visualization of the OS & PS stack.

Algorithm 3 Backtracking function

```
1: for each cell $(c_p, t_p)$ in $PS$ do
2:     Add tile $t_p$ to possible states of $c_p$        ▷ Meaning $t_p$ can exist at $c_p$ in the future
3: end for
4: Pop $(c_d, t_d)$ from $OS$
5: if Backtrack originated from a conflict then
6:     Ban $t_d$ at $c_d$        ▷ Permanently forbid $t_d$ from existing $c_d$
7: end if
```

3.1.2 Markers

Placing markers (or “save points”) allows users to apply fast iterative regeneration of undecided (non-collapsed) parts of the output. After generating part of the output, users can place a marker which they can revert to at a later stage if the generation does not head into the desired direction. On a higher level, markers apply backtracking of multiple steps at once, often referred to as “back jumping” [5].

Placement of a marker is only registered if no other marker exists at the current point in time. If allowed, a pointer to the current location on the OS stack is associated to the new marker (Figure 3.3), and then added onto the stack of markers.

To load markers, the associated pointer to the stacks is retrieved from the marker and used as a reference point when backtracking. Figure 3.3 shows the effect on the stack when loading to *marker 1* (2nd stack), *marker 0* (3rd stack), and finally reverting to the start of the generation. Manual backtracking does not originate from a conflict, hence observations being undone are not being banned (line 5, Algorithm 3).

In algorithm 4, loading of markers is explained. First, we check whether any markers exist (1), and if not, we revert to the start of the generation (2). Otherwise, we find which marker to revert to (4). Finally, we backtrack (6) until we are back at the state when the marker was placed (5).

3.2 Direct Manipulation

By design, the original WFC algorithm is fully autonomous, and chooses the next cell to continue progression through a ‘lowest-entropy’ heuristic (Section 2.1.2). Usually,

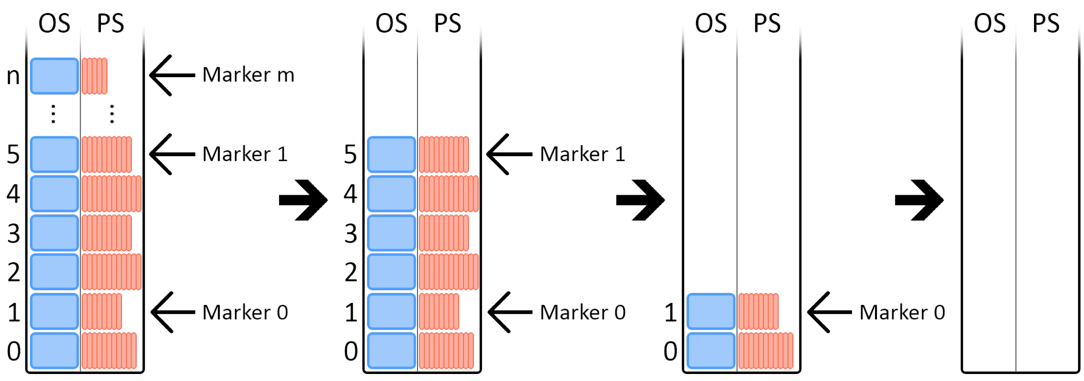


FIGURE 3.3: Visualization of markers and the stacks when loading.

Algorithm 4 Loading to a marker

```
1: if no marker exists then
2:     Reset algorithm and return
3: end if
4: $m(p) \leftarrow$ Marker to revert to          ▷ $p$ is the pointer to the OS Stack
5: while we are not at $p$ do
6:     Backtrack
7: end while
```

the chosen cell is found in the vicinity of collapsed cells, hence the 'flood propagation' appearance of any WFC animation. However, there is nothing preventing us from manually indicating *where* we would like to proceed nor, for that matter, *what* we wish to collapse there.

We designed three interactive methods to directly manipulate this spatial progression: (1) a *pencil tool*, to collapse a given cell at a desired tile, (2) a *brush tool*, to select an area of the output which is meant to be either fully reset (i.e., cleared up) or preserved from a total output reset, and (3) a *templating tool*, to allow reusability of previously created sections (or templates) of the output. These methods are meant to operate directly on cells, and allow the user to materialize their ideas anywhere in the output.

These three interactive methods address the algorithm's weakness (Section 2.2) of the uncontrollable nature of the tile and cell selection through a mixed-initiative solution. Additionally, the user can concentrate on areas of interest with more control, to manually tweak local details while leaving the rest of the output untouched.

3.2.1 Pencil tool: direct manual collapsing

Having tackled manual progression through generation of the output (Sections 3.1 and 3.1.1), and saving and loading progress (Section 3.1.2), we now turn our focus to satisfy the need for the user to say *"I want this over there!"*, without having to rely on randomness and constant trial and error.

Manual selection allows one to steer output in the desired direction, illustrated in Figure 1.1. The pencil tool materializes this idea: by "painting" with a given tile on a chosen cell, overriding the default observation of the WFC algorithm (steps 4-5, Algorithm 1) with a user-defined cell and state.

The pencil tool makes the algorithm more prone to conflicts, as the 'lowest-entropy' heuristic (Section 2.1.2) is ignored. Each cell in the output has a list of all allowed states, which the user directly interacts with through the pencil tool. Shown in Figure 3.4, where each uncollapsed cell has its own set of allowed states in the output space, visualized for a few cells. With this information, we can ignore user input if their selection is not allowed, and feed this information back to the user.

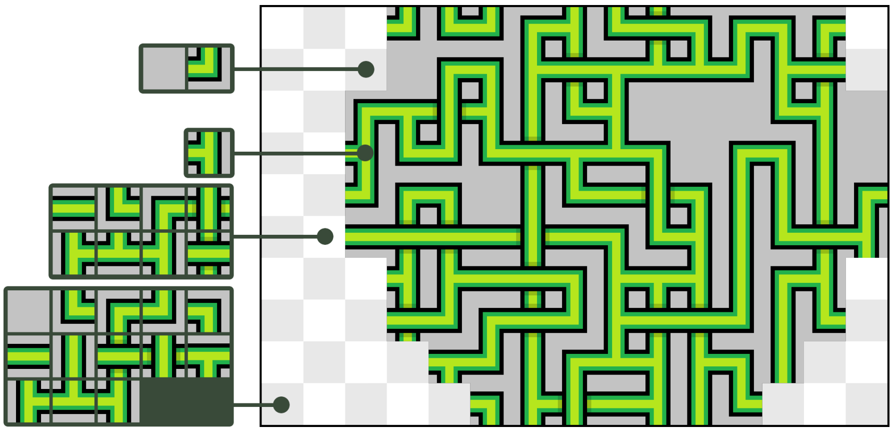


FIGURE 3.4: Some cells and allowed states.
(Input image: Knots [11])

Algorithm 5 Manual observation

```
1:  $c, t \leftarrow$ to be observed tile $t$ at cell $c$
2:  if not overwriting then
3:      if $t$ is not an allowed state of $c$ then
4:          return                                  ▷ Selected state is not possible
5:      end if
6:      Place marker if first time painting
7:      Collapse $t$ at $c$ and Propagate
8:  else
9:      $O \leftarrow$ current output               ▷ For each cell, get the allowed states
10:     Reset cells near $c$
11:     Collapse $t$ at $c$ and Propagate
12:     for each $(c_i, t_i)$ in $O$, descending by distance do
13:         if $t_i$ is an allowed state of $c_i$ then
14:             Collapse $t_i$ at $c_i$ and Propagate
15:         end if
16:     end for
17:     Place irreversible marker                   ▷ Irreversibility explained in the brush tool
18: end if
```

Lines 3 to 7 in Algorithm 5 are the "default" pencil painting procedure. We validate user selection according to the allowed states of the cell they have selected (3), and proceed to collapsing (7). If this is the first time using the pencil tool since any other action was taken, we place a marker (6) in order for users to return to the state prior to painting for fast undoing of their manual work.

A more advanced and error-prone feature is "overwriting" a cell with a state (lines 9 to 17, Algorithm 5). This removes the check of whether a state may exist at a chosen cell, and hence can cause undesired and unexpected behaviour as surrounding cells could be altered to conform to this decision.

In Figure 3.5, an example of overwriting is shown. The user wants to remove the road going through the castle at the centre of the generated image (left). After clicking on the cell with a road-less vertical castle tile, we store current output (9), reset output space (10), and place the user selection (11), illustrated by the middle image. We then retrieve the stored output and re-collapse inwards (12, red arrows) based on descending euclidean distances (gradient, light being far).

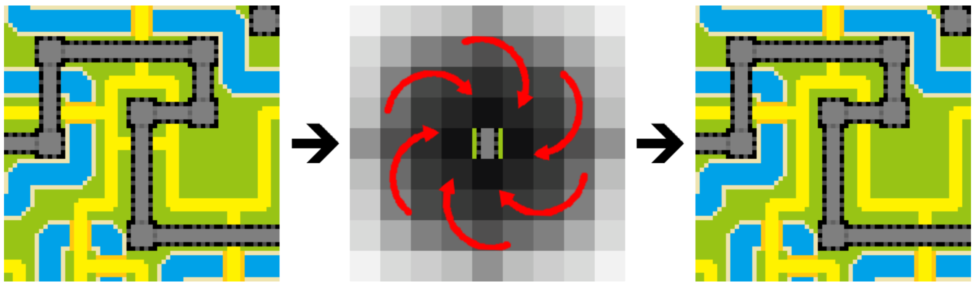


FIGURE 3.5: Process of overwriting.
(Input image: Castle [11])

Figure 3.6 shows how drastic overwriting can be. Where in Figure 3.5, only a trio of tiles changed without a significant difference, now, the entire red area is affected after placing a horizontal river, causing some cells to even allow multiple states again. The upper of the three undecided cells now allows for a horizontal road or T-Junction; the middle can host a horizontal or bridged river; the bottom cell a T-Junction or road corner. Consequently, these undecided cells show up transparent.

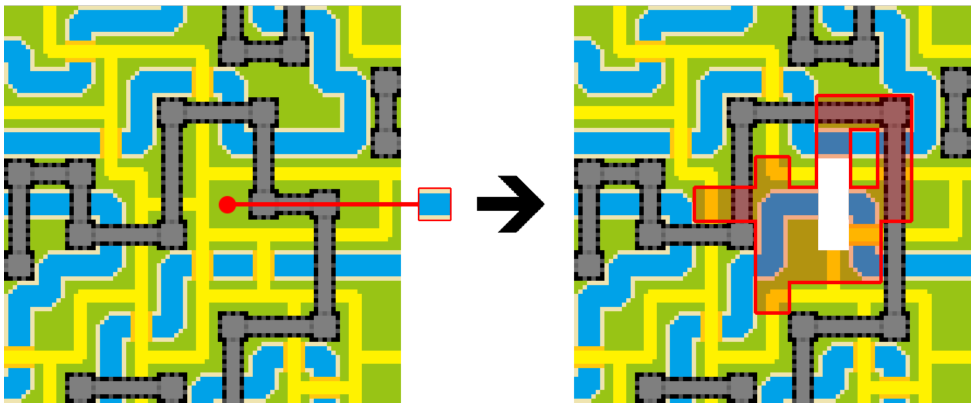


FIGURE 3.6: Overwriting example.
(Input image: Castle [11])

3.2.2 Brush tool: direct manual un-collapsing

An issue still persisting is the inability to reset areas. Currently, you can only keep undoing until all undesired cells have been removed, but this may unintentionally remove sections desired to be kept. To cover this inconvenience, we introduce the brush tool.

This method provides the ability to brush over the output, selecting areas to be either reset (i.e., fully "un-collapse" its cells) or preserved. This way, users can more accurately choose what to keep from the current output. For example, in Figure 3.7, one can define a few small areas to remove (top), or clear up the output except highlighted areas (bottom). In either case, the brushed mask indicates user intent: the actual reset should only take place upon explicit application of the masks.

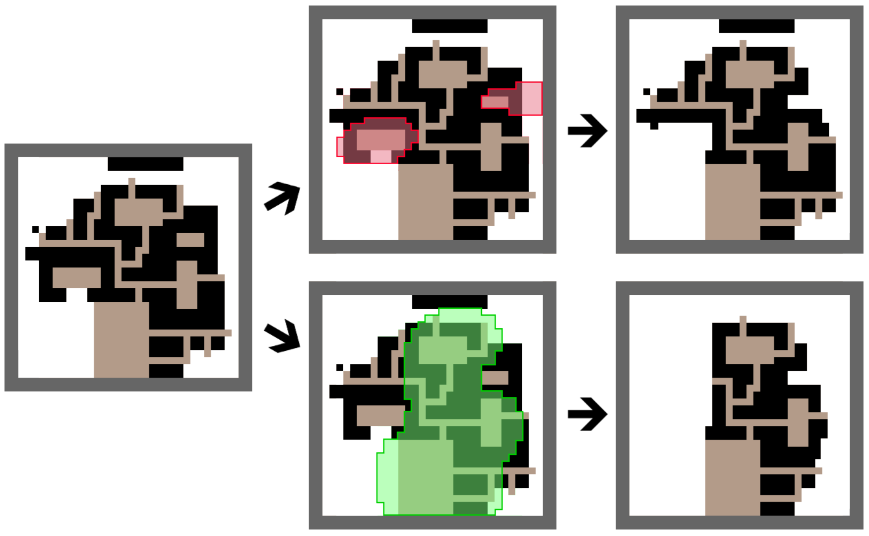


FIGURE 3.7: Manually reset areas (top) or preservation (bottom). *(Input image: LessRooms [11])*

Due to the algorithm being sequential, meaning that each subsequent step is the result of a decision made based on the previous state of the algorithm, we cannot simply reset cells, as they would propagate back to their previous state. Instead, a user-defined mask states the action for each cell, and we manually populate a new output based on this mask and the current output.

In Algorithm 6, we first get the user-drawn mask (1) and current output (2). Then, the output is reset (3) and we iterate over all cells, checking if they are marked for preservation and were collapsed (5), subsequently placed if allowed (6). Since the mask is based on previously generated output, a subset of this output can never yield conflicts, as the whole output inherently conforms to the algorithm constraints.

Algorithm 6 Brush mask application

```
1: $M \leftarrow$ User drawn mask
2: $O \leftarrow$ Current output                              ▷ For each cell, get the state
3: Reset entire output
4: for each $m_i$ in $M$ do
5:     if $m_i$ is marked for preservation and $(c_i, t_i)$ in $O$ is collapsed then
6:         Place $t_i$ at $c_i$ and Propagate
7:     end if
8: end for
9: Place irreversible marker
```

At the end, an “irreversible” marker is placed (9), implying that the user is unable to revert prior to this marker. This is due to the new output being based on a different generative process, not yielded through a WFC-centered approach, but through manual observations based on a mask. This appends marker loading (Algorithm 4) with an additional check if a marker is reversible (line 4). If not, we return. An irreversible marker is also placed when enabling overwriting whilst using the painting tool, as the area around an overwritten cell is reset and tailored to the newly selected state.

An area marked for reset might not be entirely reset, as re-propagation of a previous state can occur on the edge between preserved and reset cells, an example given in Figure 3.8. The output (step 1) has a reset mask applied (step 2). After the mask is applied (step 3), a cell is re-collapsed as this state was the only one available (step 4).

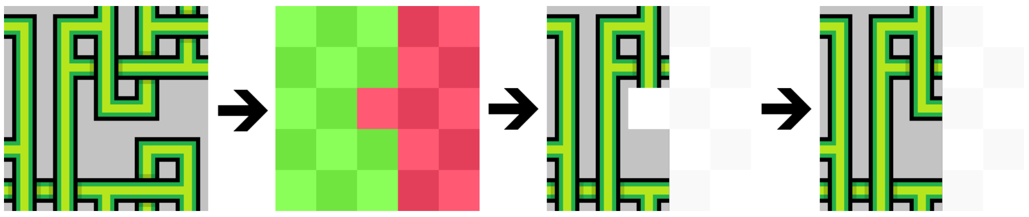


FIGURE 3.8: Cells re-propagating due to neighbouring constraints.
(Input image: Knots [11])

3.2.3 User-Defined Templates

Templates allow for increased workflow speed and fast creation of output through reusing regions of interest. In Figure 1.1, a castle was drawn onto the output, with templating, the user can instead save this castle to stamp onto the output elsewhere (Figure 3.9). The addition of templates strengthens the pencil and brush tools, providing users with capabilities to create mock-ups of regions of interest for quick reproduction, avoiding unnecessary user repetition.

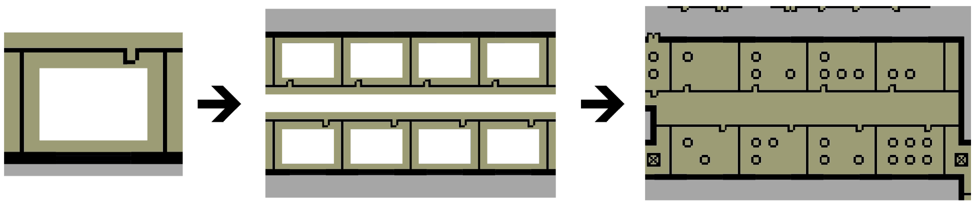


FIGURE 3.9: A template (left), template placed multiple times (middle), final output after filling the blanks (right).
(Input image: FloorPlan [11])

To improve flexibility of template placement, we allow users to paste templates over existing output when matching all overlapping cells. Additionally, we provide the ability to rotate each template through steps of 90 degrees. This functionality is shown in the placement step (middle) of Figure 3.9, having placed four rooms with a default orientation, and four rotated twice (180°).

Placing templates is explained in Algorithm 7. First, the desired rotation is retrieved (1) and applied to the template (2). Then, all tiles in this template are iterated over (3). For each tile, an offset needs to be calculated compared to the clicked position (4), in order to place each tile of the template at the desired location (5), with the middle cell of a template having no offset.

A unique use case scenario for template application arises when a template is created with any input image, say the castle input image, and afterwards altering the tiles to lower the weight of castle tiles to zero. This allows to create unique points of interest

Algorithm 7 Place a template

```
1: r ← user defined template rotation
2: Rotate *template* with $r$                                        ▷ Rotate the template
3: for each $(c_i, t_i)$ in *template* do     ▷ Iterate over the cell, tile combinations in the template
4:     $c_j$ ← offset *clickLocation* with $c_i$
5:     Collapse $t_i$ at $c_j$ and Propagate
6: end for
```

in the output through placement of a castle, being our template, followed by the algorithm generating the remainder castle-less.

3.3 Tile Manipulation

WFC Input processing (step 1, Algorithm 1) identifies all tiles in the input, including rotated and vertically flipped transformations. This may not always be desirable, for certain tiles (e.g., a road corner or chair) it is very useful, whilst others (e.g., a sunflower or compass) might have a single ideal orientation.

At each iteration of WFC, the tile selection mechanism (step 6) selects one tile at random using tile weights calculated at initialization. The more a tile occurs in the input, the higher its weight, and the higher the chance of appearing in the output.

Through the introduction of (i) weight manipulation, (ii) transformation manipulation, and (iii) pattern exclusion, we provide the user with a mixed-initiative approach to control the selection of tiles used by the algorithm, and steering the selection mechanism to more closely resemble their desired outcome.

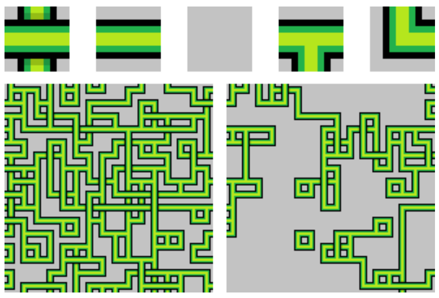
3.3.1 Weight manipulation

Tile selection is ultimately determined by weights and, as long as these are unchanged, WFC generates an output which mostly resembles the input. By providing the ability to manipulate these weights, one can create a much larger variety of output, still inspired by the input, but exhibiting very disparate ratios among their tile occurrences.

Increasing the weight of a tile makes it appear more frequently in the output. Since weights are numeric, they allow for easy manipulation, and can provide intuitive control over relative tile occurrences in the output. This has a significant impact on the algorithm's output, illustrated in Figure 3.10(a), which uses five different tiles representing pipes: Cross, Straight, Empty, T-junction, and a Corner pipe.

To alter weights with spatial variability, that is, based on the cell location, the user can draw over the output (similar to the brush mask, Section 3.2.2) and paint areas with a desired weight. This allows the user to more finely detail and regulate which tiles are allowed to saturate which regions of the output, illustrated in Figure 3.10(b).

The default WFC tile selection gets slightly modified in order to take the location of a cell into account. By providing the user-drawn weight map, as well as the location of the to-be-collapsed cell, we can use this mapping to obtain weights at this location, instead of retrieving a flat weight value.


(a) Knots tile set (top). Example outputs for (left) equal weights, and (right) uniformly increased weight for the empty tile.

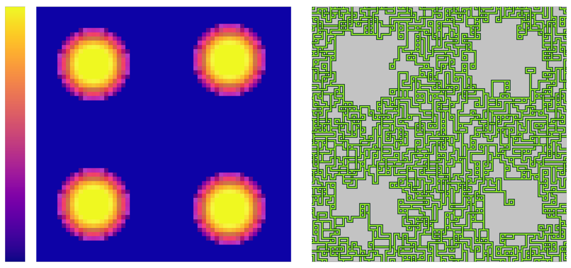
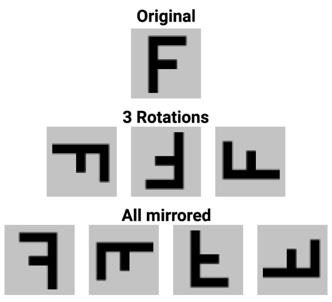

(b) Spatially variable weight manipulation for the empty tile. The olour gradient (left) illustrates the values of weights, yellow being high.

FIGURE 3.10: Weight Manipulation
(Input image: Knots [11])

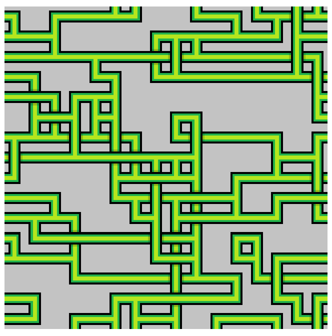
3.3.2 Transformation manipulation

Toggling tile transformations allows for additional control over the appearance of the output. There are a maximum of eight transformations possible for a single tile (Figure 3.11(a)). Not every tile is affected by every type of transformation: for example, the "Empty" tile (Figure 3.10(a)) is unaffected by any transformation, whilst "Straight" tiles are invariant to flipping.

Toggling transformations can have a considerable impact on the output. In Figure 3.11(b), the "Straight" tile has rotations disabled, causing the output to no longer exhibit long vertical pipes. In Figure 3.11(c), the "Corner" tile has all transformations disabled bar the default. As a result, only one corner tile is available, effectively lowering its ratio relative to other tiles, causing it to appear less frequently (and only in its "L" orientation).



(a) Transformations of a single tile.


(b) Only the default "Straight" transformation enabled.

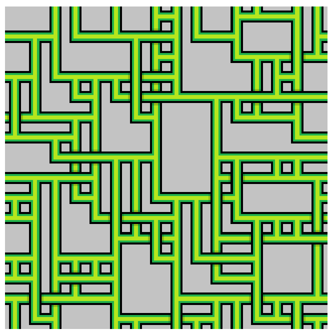

(c) Only the default "Corner" transformation enabled.

FIGURE 3.11: Transformation Manipulation
(Input image: Knots [11])

An unavoidable effect of toggling transformations is the impact on the frequency of the affected tile. If the algorithm was able to pick from a set of four tiles, three of which rotated variants of the first, and in another scenario, only one is available, it inherently alters the chance of selection fourfold. Whether this is desired, is entirely dependent on the design goal of the user. In any case, this effect can be compensated by increasing the weight of the manipulated tile accordingly.

Toggling transformations only affects the results of input processing (1) in Algorithm 1, where we skip application of rotations or flipping of extracted tiles if the user has this disabled.

3.3.3 Pattern exclusion

Patterns are automatically extracted from the input in the overlapping model (Section 2.1.2), yet users have no control over these patterns. By allowing to exclude undesired patterns, control is gained without the necessity of altering input images.

This added control makes the algorithm more vulnerable to conflicts, as we allow tweaking with patterns from a valid and sound input, potentially causing the new subset to be invalid and insufficient to generate an output image.

A less destructive yet potentially undesired effect of this feature is that disabling patterns can cause key interesting features of the input to be lost. For example, if we were to have an input image based on flowers, and excluded the patterns of a flower petal, complete flowers would no longer appear, as the input image does not provide a petal-less flower or an empty floating stem.

On the other hand, this tool can be very powerful if used purposefully, removing undesired features without the need for constant trial-and-error (Figure 3.12).

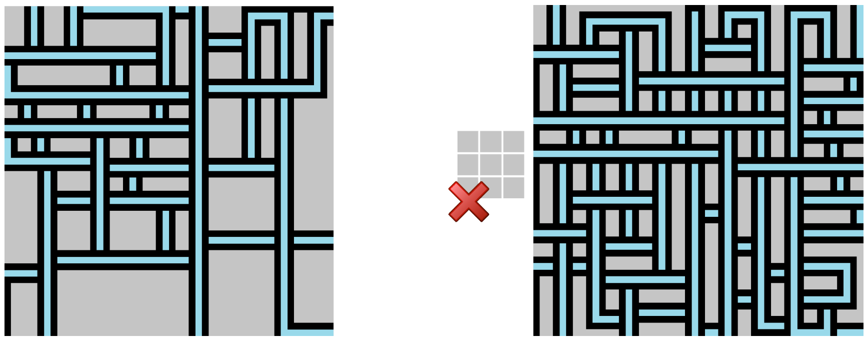


FIGURE 3.12: Two generated images, the right image excluding the grey pattern.
(Input image: TrickKnot [11], adapted colours)

Similar to transformation manipulation, exclusion of patterns is applied during input processing, ignoring a pattern from being processed when excluded by the user.

3.4 Post-Processing

Post-processing refers to operations done on output of a different process [30]. With post-processing, you can, for example, add user defined objects on top of already generated output. This process is slightly disconnected from the WFC algorithm itself, as the output has already been generated, and this layer is not yielded through a WFC approach.

As an example, the user could exploit this method by creating custom spawn locations (NPCs, monsters, animals etc.), define starting and finishing cells or checkpoints, or place items such as potions or weapons in the world.

Post-processing can be seen as a basic application example of what creative professionals can do with WFC-generated images. This functionality is limited, and can be categorized as a proof of concept rather than a full extension of the algorithm. However, the theoretical range of application seems limitless, further discussed in the Future Work.

3.4.1 Object Construction and Placement

The first step towards facilitating our so-called "objects", is to define how they are constructed in order to tailor the final layer to the user's creative preferences.

Through attributes such as a name, quantity, colour, and tiles upon which this object may appear, it promotes creative freedom. Each of these objects is appended to a list, which will be generated into a separate layer on top of the output (Algorithm 8).

Algorithm 8 Placement of user created objects

```
1: for each Object $b$ do
2:     $T \leftarrow GetAllowedTiles(b)$        ▷ Get tiles this object is allowed to exist on
3:     $c \leftarrow$ random cell with any tile in $T$
4:     Place $b$ at $c$
5: end for
```

After the user has defined a set of objects, including all necessary attributes, the grid can be generated. For each object (1), a randomly generated valid location (3) is obtained, which contains a tile this object is allowed to appear on (2), and does not already contain an object. If these constraints are met, we place it (4).

3.4.2 Object Dependencies

Some objects are meaningless without a buddy, e.g., a lock without a key is a dead object. Adding the ability to define whether objects are linked (or dependent) on others, creates more in-depth content.

Additionally, the user might want to define how far dependent objects appear from each other. The user can therefore define a range of distances objects have to appear from each other. In Figure 3.13, "treasure" (gold), always triggers the placement of a "monster" (red) within three cells distance, and both may only appear on white tiles.

The numbers within the object representations in Figure 3.13 indicate dependencies, as a cluster of similar objects could cause the user to lose track which objects were caused to appear by which parent object.

Addition of dependencies changes the behaviour of object placement. This alteration (Algorithm 9) is done through a dependency check (4 to 8) before an object is placed, to make sure that all defined constraints are satisfied (9) before actual addition (10).

3.4.3 Object Region Brushing

A final addition to the constraints on objects is drawing a mask to define where an object may appear, similar to the direct manipulation brush (Section 3.2.2) and spatially variable weights (Section 3.3.1). Figure 3.14 is an example of this locational brushing. In the green area, the user defined where the brown object may appear, combined with the constraint that it may only appear on grass tiles.

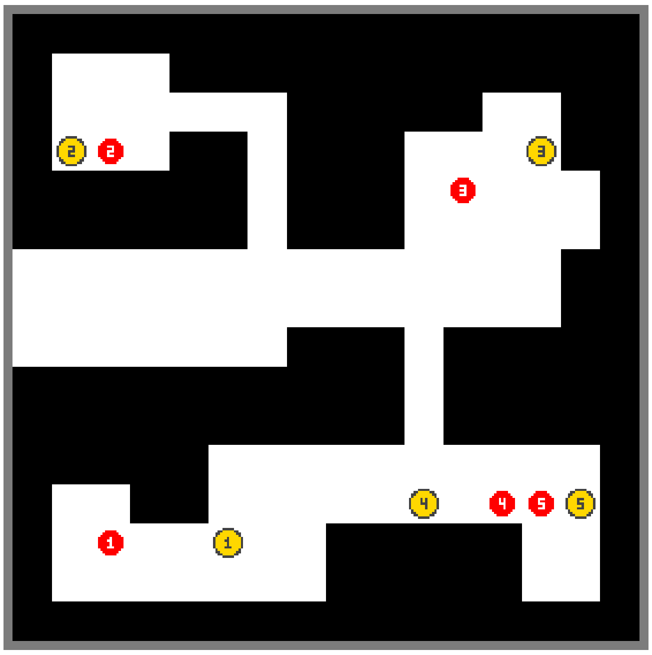


FIGURE 3.13: Dependencies application
(Input image: Dungeon [11])

To implement this, the random location retrieval at line 3 of algorithm 9 is appended with a check to only yield locations marked as allowed by the mask.

This addition sparks exploration in players if the output was, for example, a game level, by forcing objects to appear in regions where the initially generated output would not have been of interest. This mapping further improves the mixed-initiative aspect of object addition.

3.5 Additional Advanced Features and Options

The following features are minor tweaks for communication with the algorithm or to add user-friendliness to the previously mentioned methods. And provide a wider range of control over all methods and the algorithm itself.

Algorithm 9 Placement of user created objects with dependencies

```
1: for each Object $b$ do
2:     $T \leftarrow GetAllowedTiles(b)$        ▷ Get tiles this object is allowed to exist on
3:     $c \leftarrow$ random cell with any tile in $T$
4:     for each dependent object $d$ do
5:         $r \leftarrow$ defined range of object $d$
6:         $e \leftarrow$ random cell with any tile in $T$, within range $r$ of $c$
7:         Place $d$ at $e$
8:     end for
9:     if $d$ was succesfully placed then
10:        Place $b$ at $c$
11:    end if
12: end for
```

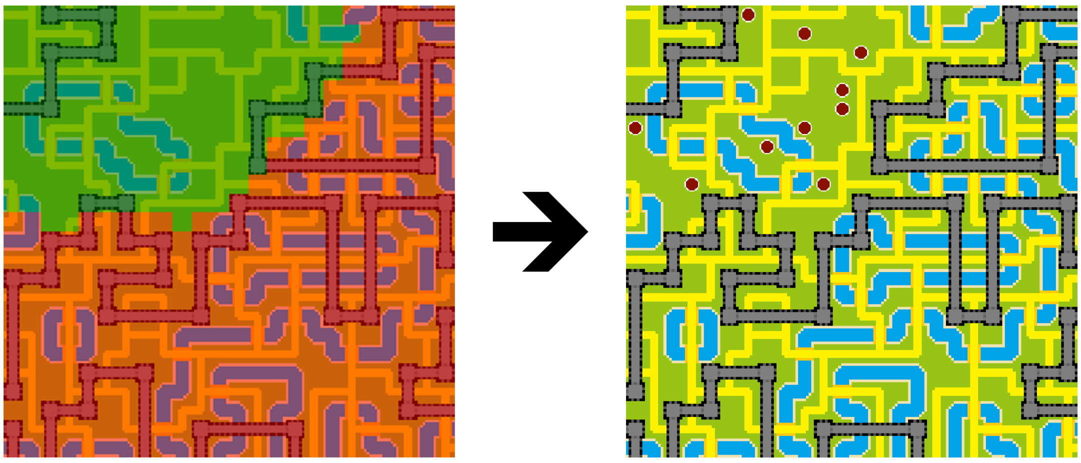


FIGURE 3.14: Object location brushing application
(Input image: Castle [11])

3.5.1 Seamlessness (or tileability)

Toggling whether the algorithm tiles treat output edges as being wrapped allows users to create output images that can be placed beside each other indefinitely. Seamlessness means that when two identical images are placed next to each other, they do not create seams, and appear as a single image (Figure 3.15).

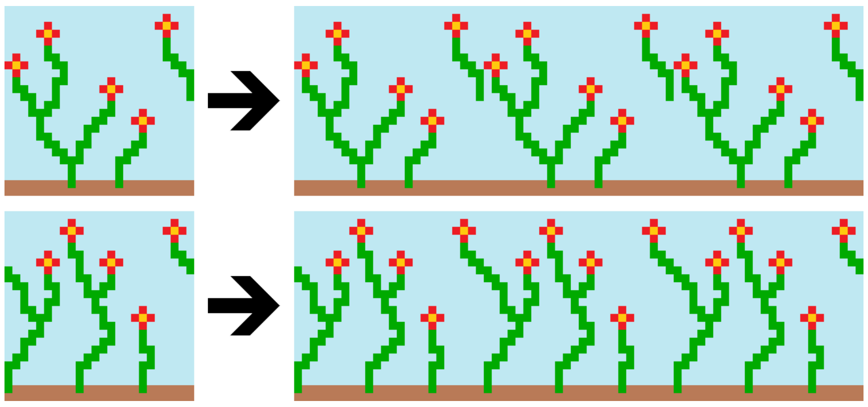


FIGURE 3.15: Seamlessness disabled (top) and enabled (bottom).
(Input image: Flowers [11])

This effect is achieved by defining the edges of the output to be dependent on each other, causing the “left” edge to act on what happens at the “right” edge, and similarly for “top” and “bottom” edges (and vice versa).

3.5.2 Input Pattern Wrapping

In the overlapping model (Section 2.1.2), the user is able to toggle whether input processing may extract patterns by tiling the input image beside itself. The reason to have this setting disabled or enabled is dependent on the input, as the algorithm comes across new patterns (most of the time) with this option enabled (Figure 3.16, top versus middle).

If the user has input with an upright orientation (rather than being flat), such as the flowers in Figure 3.15, enabling input pattern wrapping can yield unexpected and undesired behaviour, as a pattern is also extracted with "sky" below "ground", causing floating layers of ground to appear.

3.5.3 Pattern Size Adjustment

Another feature solely available using the overlapping model is the ability to change the size of patterns extracted from the input. In Figure 3.16, an example is shown with patterns of size 3*x*3 (top and middle) and 2*x*2 (bottom).

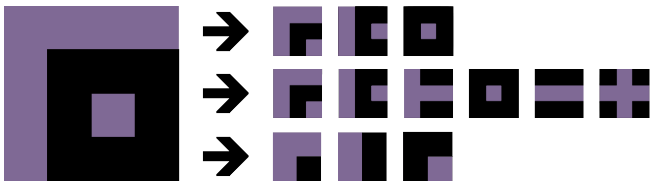


FIGURE 3.16: 3*x*3—sized patterns extracted without (top) and with (middle) input wrapping; 2*x*2—sized patterns (bottom). *(Input image: SimpleMaze [11] with adapted colours)*

Changing the pattern size significantly changes the output, as local features are scaled up or down depending on the pattern size. In Figure 3.17, different pattern sizes affect conservation or loss of local features. A large pattern size showing a closer resemblance to the input.

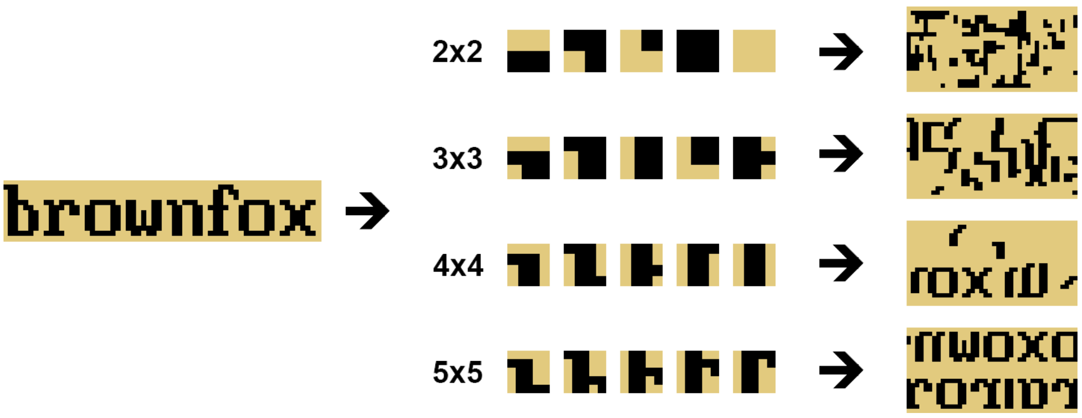


FIGURE 3.17: Various pattern sizes resulting in distinct output *(Input image: BrownFox [11])*

A large pattern size also drastically increases running time and removes the possibility for unique interpretations of the input to exist, such as the unique calligraphy created by the 4*x*4 pattern output in Figure 3.17.

3.5.4 Output Exporting & Importing

Saving output is an unmissable feature of any generator and an intrinsic feature of the original WFC algorithm implementation by Gumin [11]. Importing previously created output is not, and it allows users to save work and come back to it later. Additionally, users can re-touch creations and append it with new ideas. Importing is straightforward, as previously exported output is considered valid, given they were generated by the algorithm itself.

Algorithm 10 is the step-wise description of how an image is imported. After selecting an image to import (1), we reset the output to make room for the new image (2).

Two checks must be passed, the first being whether the imported image conforms to the tile dimensions of the input image (3), for the overlapping model, this check is omitted, as tiles are of size $1x1$. For the adjacent model, the image dimensions must be a multiple of the dimensions of the tile (e.g., *Castle* [11] tiles are of size $7x7$, *Knots* has tiles of size $10x10$). Second, all colours and tiles must match the input image (4). If both checks succeed, we iteratively place each tile onto the output (7).

Algorithm 10 Import an image

```
1: $I \leftarrow$ User to selected image to import
2: Reset output
3: if size is invalid                          ▷ E.g. not conforming to tile size
4:     or cells are invalid then               ▷ May only contain colours/tiles of the input image
5:     Error, cancel import
6: end if
7: for each $(c_i, p_i)$ in $I$ do             ▷ Place each encountered cell onto the output
8:     Place $p_i$ at $c_i$
9: end for
```

Complications emerge when exporting output created with adapted weights (e.g. through weight or transformation manipulation), as users likely desire keeping these weights identical after importing, especially when continuing unfinished work.

Luckily, the PNG data structure [29] used for exporting describes the start and end of its own data, allowing appending of data thereafter without interference, ideal for storing integers, such as weights. Appending data directly, instead of having separate files, cuts down actions required for the user and makes sure weights are always linked to the correct output, as mixing can cause conflicts or inability to import.

3.5.5 Weight Mapping Exporting & Importing

Spatially variable weights (Section 3.3.1) can also call for re-usage. As weights are capped to a maximum of 250, all weights fall within the hexadecimal numbers used for indicating colours (#*RRGGBB*), allowing to export mappings as greyscale image.

Greyscale itself does not take away from readability, as the user can still view the mapping as an image, dark areas being lower and light areas higher weights. This simplifies saving and loading, as no conversion from colours to weights is required.

When exporting, we iterate over all values, converting each weight to a greyscale colour, and exporting this as a *PNG* file. When importing, we convert the greyscale image back to a weight mapping, and assign it to the selected tile.

3.5.6 Custom input images

Using custom images as input unlocks the final step towards complete control. Users can (i) use an existing input image to modify it to their design goal, (ii) export this output, and (iii) use it as input. This essentially turns WFC into a mixed-initiative "evolutionary algorithm", as users are able to filter each generation and reinsert offspring.

Additionally, this allows creating custom input images (through e.g., Paint[4]). This, however, is a more “advanced” feature and requires algorithm understanding. Since WFC scans input to extract patterns, changing a single pixel could dispose of key patterns. In Figure 3.18, the change of a single pixel significantly alters the output, as this yellow pixel prohibits red areas from growing.

Similarly, creating features larger or equal than the pattern size can cause output to be saturated with this pattern, shown in Figure 3.18 (left), where the default *Colored-City* [11] input has a completely red pattern.

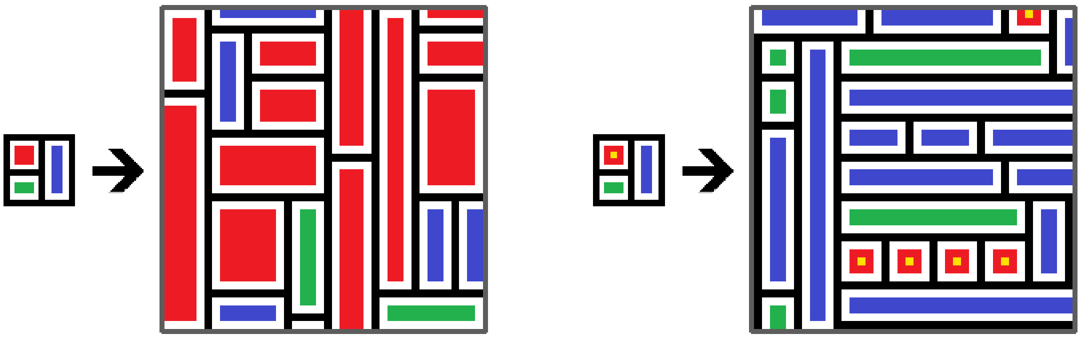


FIGURE 3.18: Default input (left), user adjusted input (right)
(Input image: ColoredCity [11])

A final issue is the accidental inclusion of two visually similar colours, yet differing by a single bit (e.g., *#FD69D*<u>*4*</u> ■ vs *#FD69D*<u>*5*</u> ■, visually similar, numerically diverse). This can cause a pattern to be more constrained than initially desired, shown in Figure 3.19, where the inside of the knot has a slightly darker shade of white, causing output to solely contain closed strings.

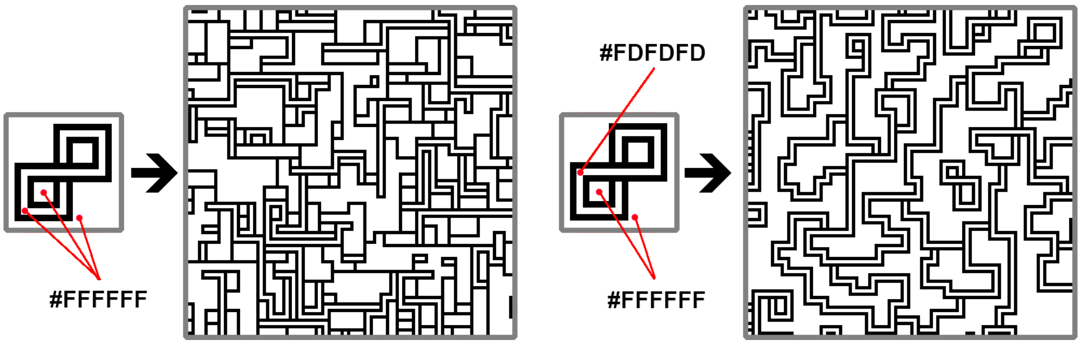


FIGURE 3.19: Effect of visually similar & numerically distinct colours
(Input images: Knot (left) and TrickKnot (right) [11])

Combined with filtering and manipulative methods, requirement of thorough understanding of WFC to create input images is partially diminished, and now provides full control over the algorithm, internally and externally.

[4]Microsoft Paint - `https://apps.microsoft.com/store/detail/paint/9PCFS5B6T72H`

4 Prototype Implementation

In order to evaluate our methods on intuitiveness, we developed a prototype implementation designed to provide an intuitive interface to these methods. This prototype can be seen as a means to approach the research questions (Section 1.1).

First, we present reasoning behind our choice of language and UI Framework. Second, we touch upon the interface structure and rationale. Third, we describe the code architecture used in the prototype implementation. And finally, for each of the designed methods, we provide our design choices and the final design itself.

The prototype has been implemented using *C#*[5] and the open source DeBroglie library by A. Newgas [3]. *C#* was chosen as the default WFC algorithm by M. Gumin [11] was implemented using this language. The choice of using the DeBroglie library over the default implementation was due to (i) backtracking, which the default algorithm does not contain, and (ii) the library being very well documented, where the original algorithm is, as Karth and Smith state, "a black box" [15].

Due to inexperience in UI design, we decided to aid our design process through a UI framework, saving both time and avoiding unnecessarily designing basic UI elements from scratch. Second, we are building a prototype, not a fully fledged product, and serves function over form. Our framework of choice initially landed on Windows Forms[6], later changed to Avalonia UI[7], as it allowed for more advanced features, faster window calculation, and building for multiple platforms.

The *miWFC* prototype can be downloaded at its repository[8], for anyone to experiment with. As an open-source project, its source code is available under the usual conditions, for others to see, modify or perform further developments.

4.1 Interface Layout

Prior to creating the prototype, we decided on a wireframe, establishing what goes where. For this, we split up our application into sub-windows; (i) the main window, handing basic interaction, including history navigation (Section 3.1), tile manipulation (Section 3.3) and additional advanced features and options (Section 3.5), and sub-windows for (ii) direct manipulation (Section 3.2), (iii) spatially variant weight adjustment (Section 3.3.1) and (iv) post-processing (Section 3.4).

The main window additionally is split up into three subsections (Figure 4.1), as most time will be spent here, and logical grouping of identical features would add to intuitiveness. For this, we split the main window into three parts; the input side on the left, algorithm manipulation in the centre, and the output side on the right.

[5]C Sharp - `https://en.wikipedia.org/wiki/C_Sharp_(programming_language)`
[6]Windows Forms - `https://en.wikipedia.org/wiki/Windows_Forms`
[7]Avalonia UI - `https://avaloniaui.net/`
[8]miWFC - `https://github.com/ThijmenL98/miWFC`

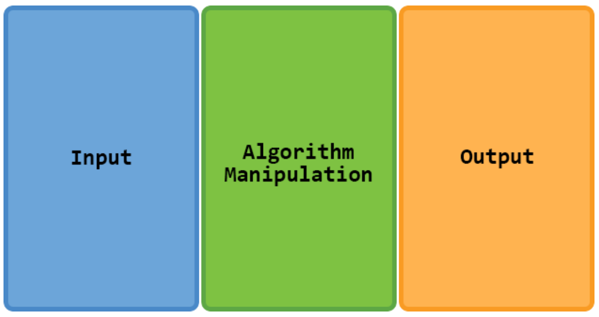


FIGURE 4.1: Three subsections of the main window

Each window always contains a "help" button providing an informative pop-up of the features visible in the current window, and explanations of terms. Finally, all visuals have been tested using colour filters to make sure that people suffering from colour blindness (although unlikely for creative professionals) can also use the prototype. The software used to simulate colour blindness is Color Oracle[9], which applies a full screen filter based on the selected impairment.

4.2 Code Architecture

Avalonia UI uses the Model-View-ViewModel (MVVM) design pattern [22] (Figure 4.2), allowing for separation of display and logic. Each sub-window, as well as the main window subdivisions have their own view model, handling data bindings to their respective parts of the interface, and communicating with the model.

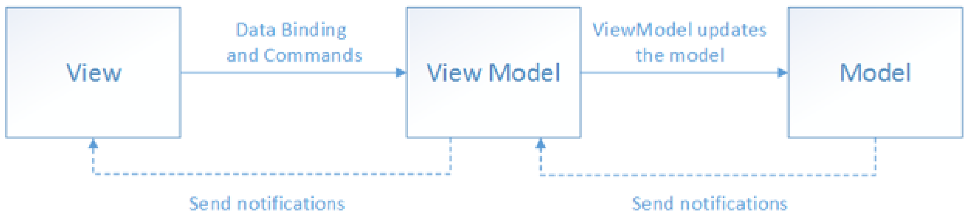


FIGURE 4.2: Model-View-ViewModel[10]

For code maintenance and manageability, we reduced our model into a handful of handlers, each with its own task; The *WFC Handler* handles all communication to and from the DeBroglie library. The *Output Handler* handles all direct manipulation of the output. Finally, the *Interface Handler* encapsulates all other interface handling, such as window switching, pop-ups and changing input parameters.

These handlers all follow the *Singleton Creational Design Pattern* [26], and *Chain of Responsibility Behavioural Design Pattern* through a central delegator, which houses the singleton instances of each handler. Finally, the *Adapter Structural Design Pattern* allows for operation of both the overlapping and the adjacent model within the same application and instance of the DeBroglie *TileModel*[11].

[9]Color Oracle - https://colororacle.org/
[10]*Source: Avalonia UI Documentation* - https://docs.avaloniaui.net/
[11]https://github.com/BorisTheBrave/DeBroglie/tree/master/DeBroglie/Models

4.3 Default Features

In order to have basic communication with the DeBroglie library, and allow usage of the default WFC algorithm without the need for a command line or programming, we need to implement the default features of WFC such that the algorithm can be manipulated. These basic features include (i) selection of input, (ii) defining output dimensions and (iii) the ability to reset the generation.

To improve user experience and reduce searching time required by the user, we categorize all default WFC input images based on appearance (Table 4.1).

Category	Description
Textures	Surface detailing to apply to 2D/3D models
Shapes	Surfaces of multidimensional objects
Knots	Intertwined and/or tangled lines
Fonts	Printable or displayable text characters
Words Side-View	(Game) Worlds as seen from the side
Worlds Top-Down	(Game) Worlds as seen from above
Custom	Images uploaded by the user

TABLE 4.1: Input image categories

In Figure 4.3, these default features have been given a place in the window, a screenshot shown in Figure 4.4. On the input, the user can select between the adjacent model (A) and the overlapping model (O), and choose the category (C) and input image (S) using a drop-down list. This selected input image is shown to the user below (INPUT). In the bottom left the "info" button (I) is located. In the centre of the window, the user is shown the extracted patterns or tiles from the input. On the output side, size of the output (width (W) and height (H)) can be tweaked using a numeric up-down control, accompanied by a reset button (R) to re-start generation of the output.

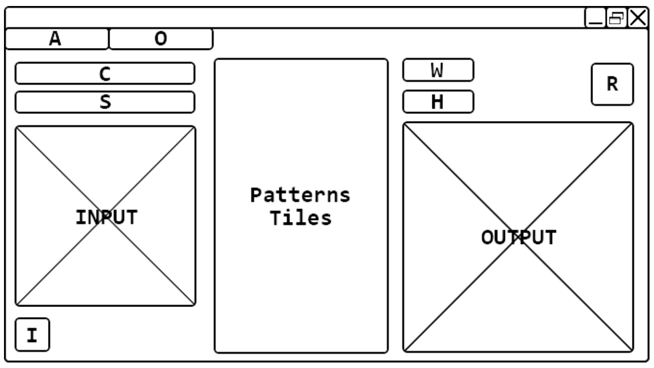


FIGURE 4.3: Default Features interface wireframe

4.4 History Navigation

To progress through output generation, the user requires two buttons for advancing and reversing time. This, is very limited and would take excessive time on larger outputs, as the clicks required to fully generate the output increases with image size.

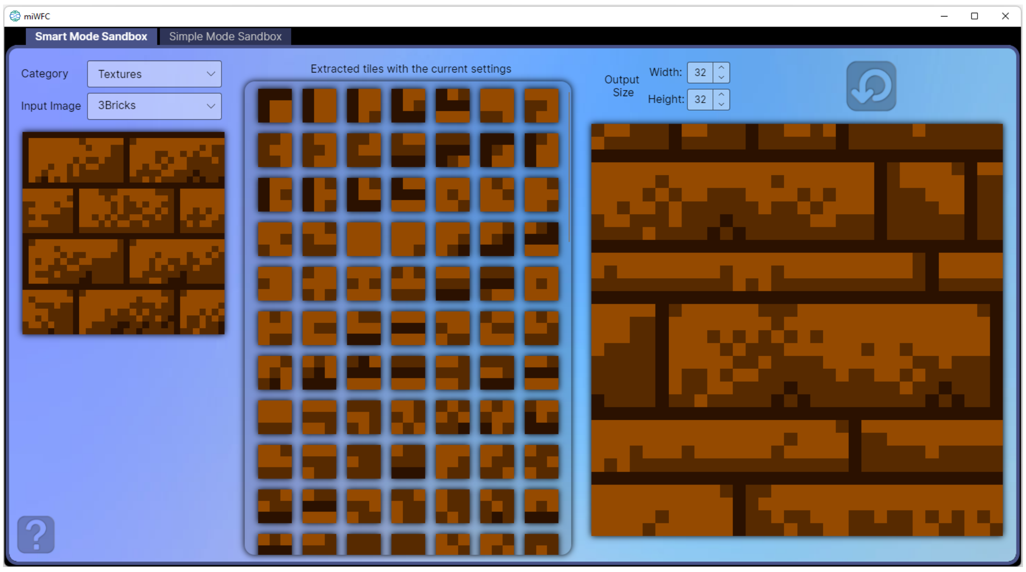


FIGURE 4.4: Default Features screenshot

We introduce a set of tools, which fall under timeline manipulation of output generation; aside from buttons to (i) advance and (ii) reverse time, we also provide buttons to (iii) save progress by placing a marker, (iv) load the latest marker, (v) a slider to alter the amount of steps to take, the ability to (vi) animate generation, and (vii) alter the time between steps of the animation.

As mentioned in Section 3.1, a timeline can be added to visualize the generative process. This timeline can show both the current point in time, as well as each time the user has manually saved progress. This provides a better feeling of the relation between the "now" and placed markers. The positions of both save and the "now" marker along the timeline are calculated by dividing the collapsed cells by the total amount of cells, and using this value to deduce its position.

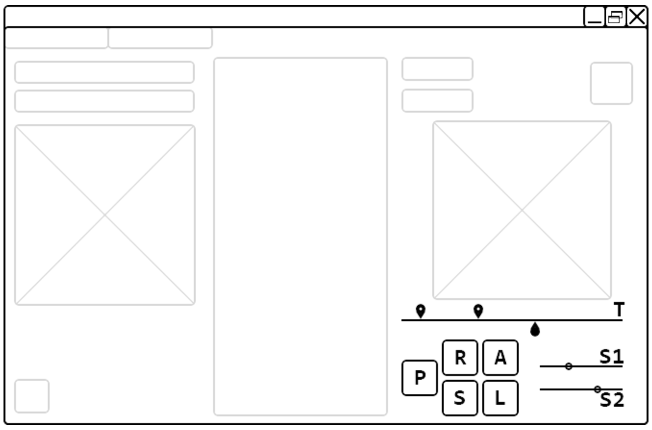


FIGURE 4.5: History Navigation interface wireframe

These new additions are shown in black in Figure 4.5, where existing elements are semi-transparent. The new elements are placed below the output, as history navigation directly influences the output. Directly below the output, the timeline (T) is shown, with save point markers above the timeline, and the current point in time below it. Five buttons handle play/pause of the animation (P), advancing (A) and reversing (R) through time manually, and buttons to save (S) and load (L) markers.

On the right are two sliders for the amount of steps to take (S1) and speed of the animation (S2).

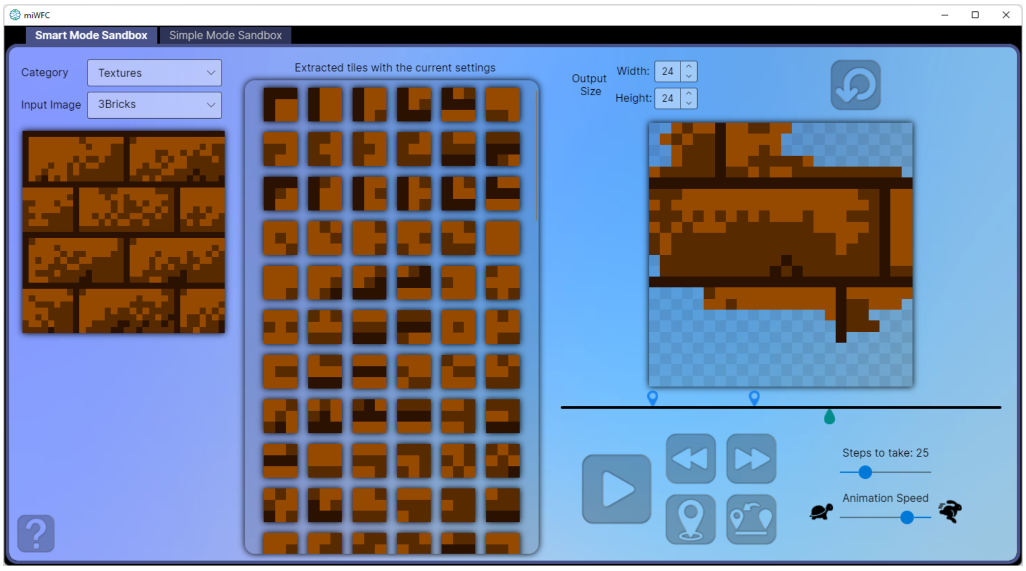


FIGURE 4.6: History Navigation screenshot

4.5 Direct Manipulation

Direct manipulation is a very powerful tool, and all actions taken through these tools have direct and significant impact on the output. Due to this, we separate these tools from the main window, as accidental application can be destructive. For this, we add a button in the main window which takes us to a subwindow, solely housing these tools (layout shown in Figure 4.18).

The main purpose of our new sub-window is to directly operate on the output, and for this, a large working area is beneficial. Additionally, direct manipulation encapsulates multiple tools which cannot be used simultaneously. Focusing on a single tool and suppressing others reduces on-screen clutter and improves manageability.

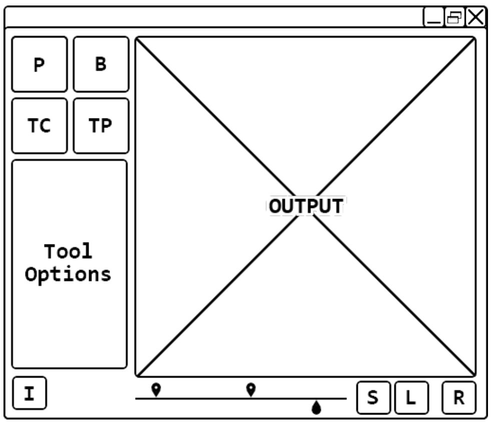


FIGURE 4.7: Direct Manipulation subwindow interface wireframe

In Figure 4.7, the direct manipulation layout is shown, with our earlier mentioned large output image a central feature. In the top left, four buttons allow switching between the pencil (P, Section 3.2.1), brush (B, Section 3.2.2) and templating tools (Section 3.2.3), as well as both template creation (TC) and template placement (TP).

A large area is reserved for additional controls and options associated to the selected tool. Our usual "info" button (I), timeline, both with the save (S) and load (L) buttons, and a button to return to the main window (R) are located at the bottom.

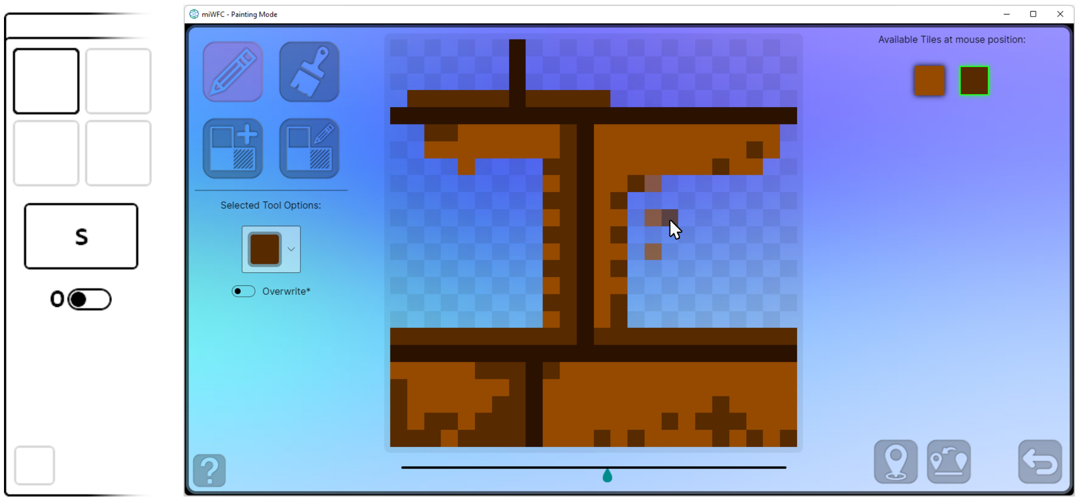
4.5.1 Pencil Tool

The pencil tool in its most basic form requires two pieces of information; what to place, and where. To improve intuitiveness and ease-of-use, we provide two types of informative insight into the algorithm; (i) the allowed states for a cell and (ii) a preview of the consequence when applying the selected action.

By providing insight into the possible states of the cell the user is interested in, we reduce confusion as to why an action might not be allowed. This is possible as WFC always keeps track of all allowed states for each cell. Although this information is normally hidden within the algorithm, we can display it in real-time and on demand when hovering over a cell.

To indicate how the algorithm responds to an action, a preview is introduced to show resulting propagations of this action. This preview is calculated by duplicating the algorithm's state, and attempting to place the user selected value at their hovered cell, showing the resulting state as a semi-transparent layer. This allows users to see the effect of clicking, further reducing confusion, as propagation now becomes knowledge prior to collapsing.

The pencil tool options consist of two parts (shown in Figure 4.8(a)); selection of a tile (S), and a toggle to enable or disable overwriting (Section 3.2.1). A screenshot of the tool menu is shown in Figure 4.8(b).



(a) Wireframe (b) Screenshot

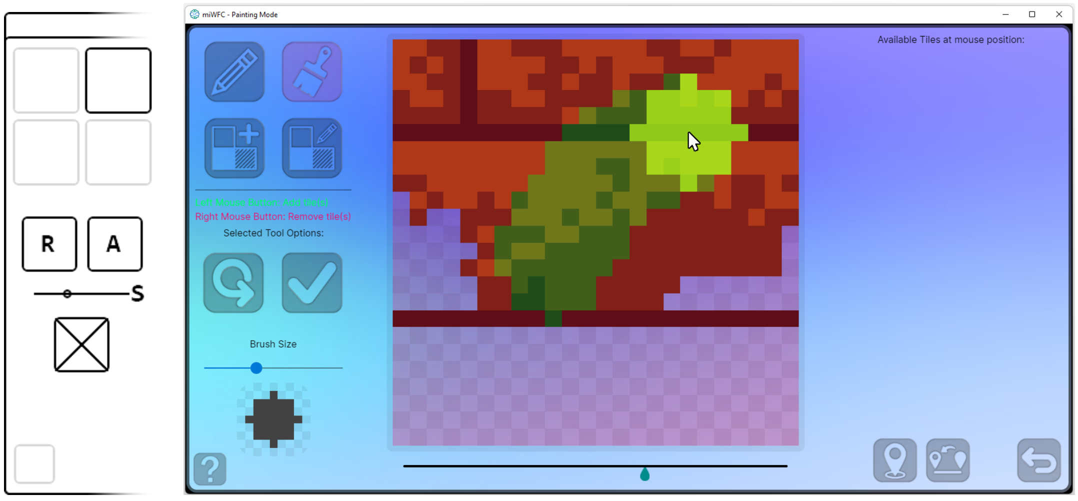
FIGURE 4.8: Pencil tool interface

4.5.2 Brush Tool

To allow for an intuitive brushing experience, we provide the ability to select a brush size. This can be achieved by providing a fixed set of brush sizes, a brush diameter through numeric input or through a slider. This third approach was our design of choice, accompanied by a small visual representation of the selection, to allow to directly see the brush, rather than guess the required size.

Additionally, we provide the user with the ability to apply the drawn mask, rather than applying the brush as the user releases their mouse. Next to this, resetting the mask allows for quick discarding of previous work. If the user were to return to the main window without having the mask applied, they are prompted to either discard or apply the mask upon exiting the subwindow through a pop-up.

The brush tool options (Figure 4.9(a)) contain reset (R) and mask application (A) buttons, as well as the brush size slider (S) and visual representation below. A screenshot of the implementation is shown in Figure 4.9(b).


R
A
S
miWFC - Painting Mode
Available Tiles at mouse position:
Left Mouse Button: Add tile(s)
Right Mouse Button: Remove tile(s)
Selected Tool Options:
Brush Size

(a) Wireframe (b) Screenshot

FIGURE 4.9: Brush tool interface

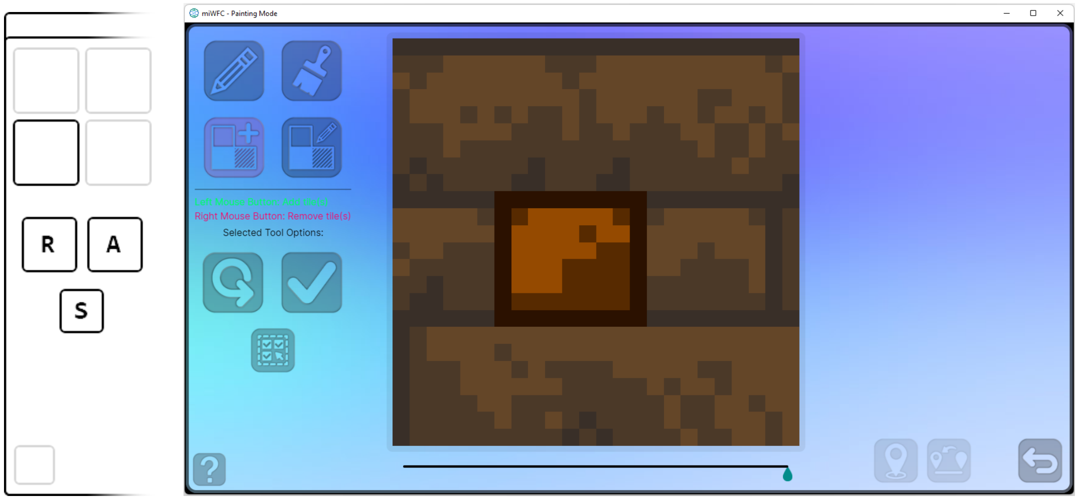
4.5.3 User-Defined Templates

Templates are a two-step process, first creating a template, followed by placement. Since these tools are distinctly different, we decided to keep them separate to maintain focus on one tool at a time.

During template creation, the user is able to directly draw on the output to outline their desired template. Similar to the brushing tool, the user is supplied with the ability to reset the template or apply it. Next to these, a third button is introduced to select all collapsed cells in the output. This is useful when first having used direct manipulation to draw on the output, then with a single click, select the drawn creation all at once. To indicate what is selected, we lower the contrast of cells that are not selected, shown in Figure 4.10(b).

When the user saves the created template, it is stored locally, with the name of the input image being used encoded in the template name. This allows to only show templates associated with the current input image, as a template referenced from a different input image would not be practical.

In the layout (Figure 4.10(a)), mask reset (R) and application (A) buttons, as well as the button to select all non-transparent cells (S) are shown.


R
A
S
miWFC - Painting Mode
Left Mouse Button: Add tile(s)
Right Mouse Button: Remove tile(s)
Selected Tool Options:

(a) Wireframe (b) Screenshot

FIGURE 4.10: Template creation tool interface

With template placement, the user is to select which template to place. In order to allow for a more intuitive experience, we decided that the click location of the user is at the centre of the template (Figure 4.11(b)), rather than the "usual" top-left corner. This also provides the ability to place templates at all edges of the output.

If the user desires to rotate a template, we allow them to do so through 90 degree steps. Rotated cell locations can be calculated by first converting our degrees of rotation to radians (r), with the origin being the mouse position on the output relative to the centre of the template:

$$x_{rot} = x_{origin} + Cos(r) \cdot (x - x_{origin}) + Sin(r) * (y - y_{origin})$$

$$y_{rot} = y_{origin} - Sin(r) \cdot (x - x_{origin}) + Cos(r) * (y - y_{origin})$$

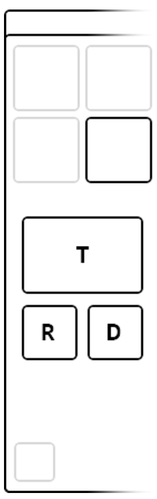
Finally, if the user wishes to dispose of a template, they can delete the selected template from their system with a single click of a button.

Shown in Figure 4.11(a), the template placement options are shown, with the template selection (T), and rotate (R) and delete (D) buttons below.

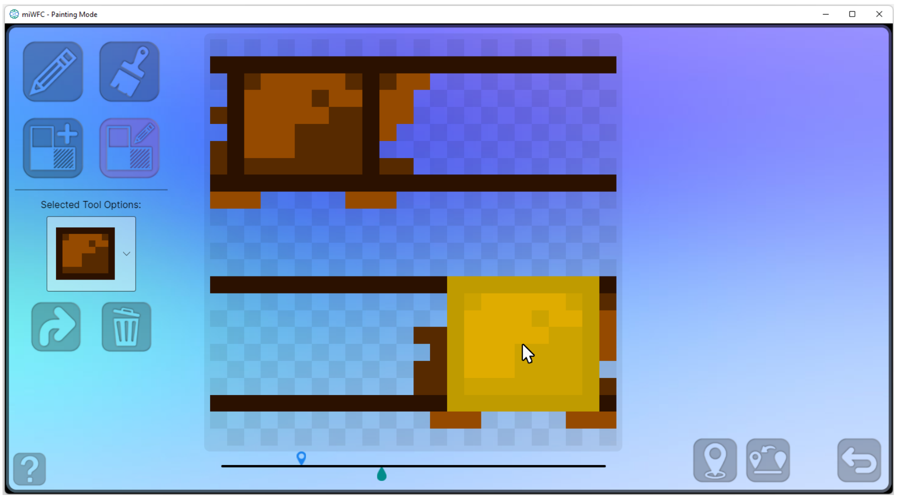
4.6 Tile Manipulation

Tile manipulation is the first in what is considered an "advanced feature". In the main window, the user can enable advanced features through a toggle (Figure 4.18). The reason to add a toggle for advanced features, is that these features are not always used, as some require a more thorough understanding of the algorithm.

With advanced features enabled, users are presented with the layout shown in Figure 4.12(a) for each unique tile in the output (excluding transformations). Along



(a) Wireframe



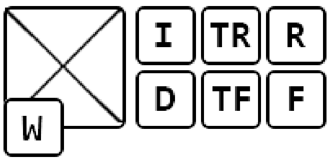
(b) Screenshot

FIGURE 4.11: Template placement tool interface

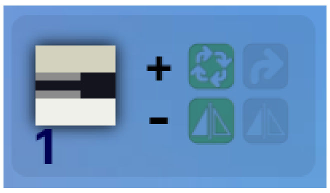
with the tile representation itself, they can see the current static weight (W) associated to this tile, and adjust it by increasing (I) or decreasing (D) it.

We introduced modifier key functionality for changing weights, being the CTRL and *SHIFT* keys. Holding either button will alter the amount of change inflicted on the weight. CTRL adjusts the weight by steps of 10, and SHIFT by 50. Using neither will adjust the weight by 1.

To the right of these, one can alter the allowed transformations of a tile. If the tile is invariant to rotations or flipping, the invariant feature(s) will not be shown. Otherwise, the user can toggle rotations (TR) or flipping (TF), and lock the transformation of the tile rotationally (R) or its flipped representation (F). Our prototype representation is shown in Figure 4.12(b).



(a) Wireframe



(b) Screenshot

FIGURE 4.12: Static tile weight manipulation interface

For spatially variable weights, the weight value (W) in Figure 4.12(a) doubles as a button to alter the weight of the associated with tile spatial variability. This menu can also be opened through a different button, less "hidden", atop the algorithm manipulation section, shown in Figure 4.18. If a weight has spatially variable weights, the weight value will instead show as "D" for dynamic.

When altering a weight with spatial variability, the user is shown a subwindow (Figure 4.13). In this menu, the user is able to paint on a large canvas of similar size to the output. After selecting which tile to alter (T), done automatically if opened by clicking on the weight of a tile, the user can select the brush size (BS) to work with.

Furthermore, the user can decide between a hard or soft brush (BS). The hard brush will paint everything with the selected value (W), whereas the soft brush will gradually interpolate outwards. Furthermore, a handful of buttons are provided to reset the mapping (R), import (IM) or export (E) a mapping, and the usual buttons to return (R) and the "info" button (I).

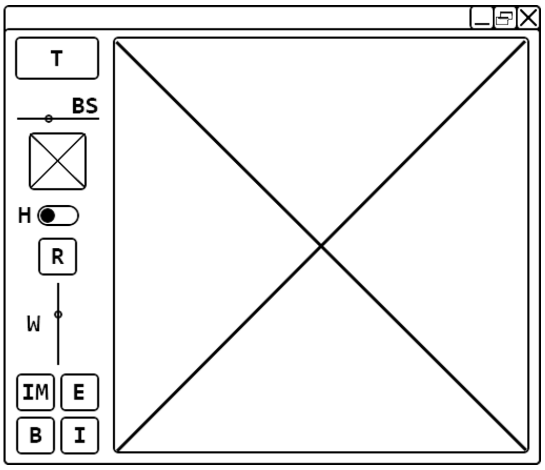


FIGURE 4.13: Spatially variable weight manipulation wireframe

Weights are indicated using the plasma colour gradient[12] (Figure 4.14), which is (i) colourful, allowing to easily see differences, (ii) perceptually uniform, values close to each other have similar-appearing colours and values far apart have more different-appearing colours across the entire range of values, and (iii) robust to colour blindness, as the previous two properties hold regardless of colour deficiency.

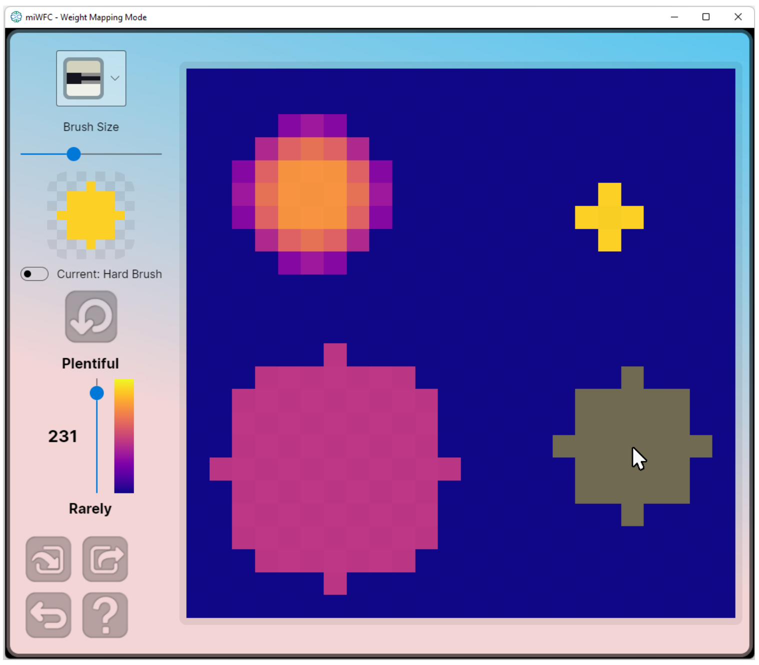


FIGURE 4.14: Spatially variable weight manipulation subwindow screenshot

[12] https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html

Both spatially variable and static weights can be reset to their default values with a button click in the main window, located next to the spatially variable weight button, both of which shown in Figure 4.18.

4.7 Post-Processing

Similar to direct manipulation and spatially variable weight mappings, post-processing is implemented through a separate sub-window to keep categorically distinct tools separated. Below the button to proceed to direct manipulation, an additional button is placed for post-processing (layout of which later shown in Figure 4.18).

In Figure 4.15, the post-processing layout is defined. On the left, a large representation of the output is provided both as an insight for the user, and to directly show objects on top of the output. The usual "info" button (I) is located at the bottom left. Furthermore, a handful of buttons are provided for the addition of a new object (MA), to edit (E) or delete a previously created object (D).

All objects created by the user are shown in a table format. Finally, a button is added to regenerate placement of all objects (R), as well as a button to return to the main window (B). A screenshot shown in Figure 4.16.

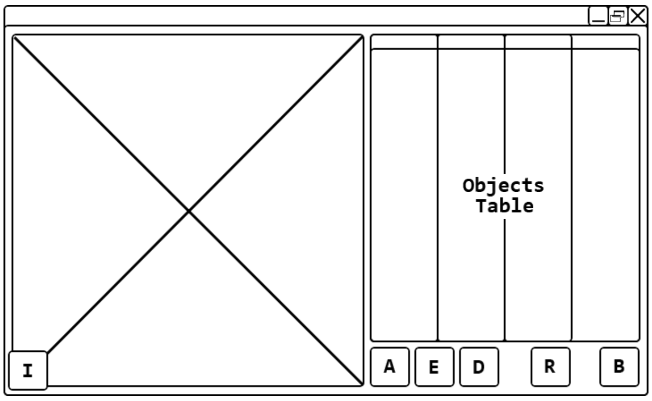


FIGURE 4.15: Post-Processing wireframe

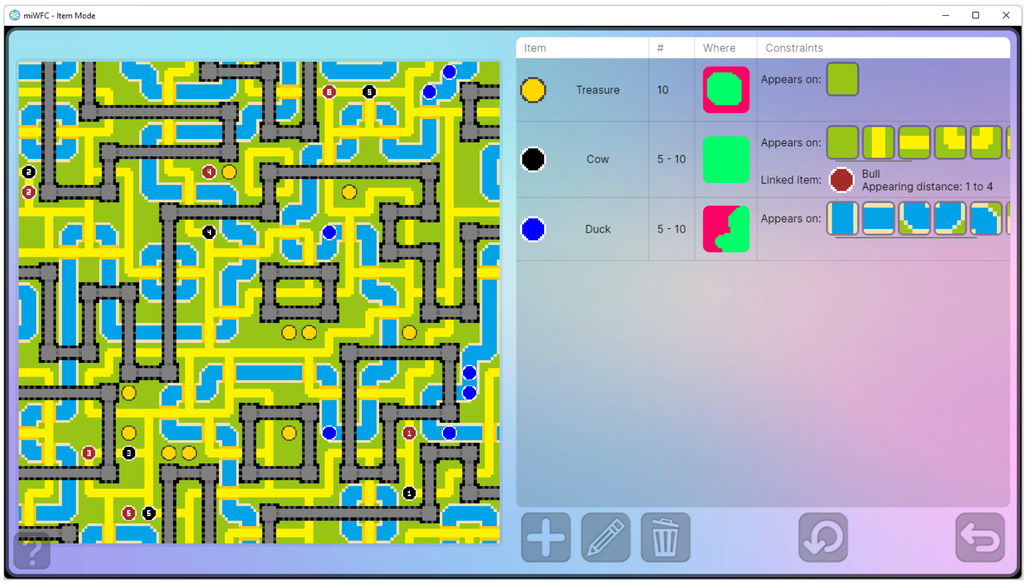
Within this subwindow, either after creating a new object, or editing an existing one, the output image is replaced with a set of controls (Figure 4.17(a)). With these, the user is able to define; (i) what the object is, (ii) how many should exist, and (iii) where it may appear. Additionally, the user is able to define a secondary object linked to the first, causing both to appear inseparably.

At the top of this interface, the user is able to define the name (MN) of the object, as well as the amount (A), whether it be fixed or defined by a range. For more customizability and object separation, the user can define a colour (MC) either through CSS keywords[13] or with a hexadecimal value.

To define where objects may appear (W), the user can select to make it appear anywhere or on a selection of tiles. They can additionally constrain where objects may appear by using a mapping (M) to define for each cell if it may contain the object.

At this point, the object can be added (A) or the user can choose to define a linked object (L), define its name (LN), colour (LC) and distance of appearing from the main

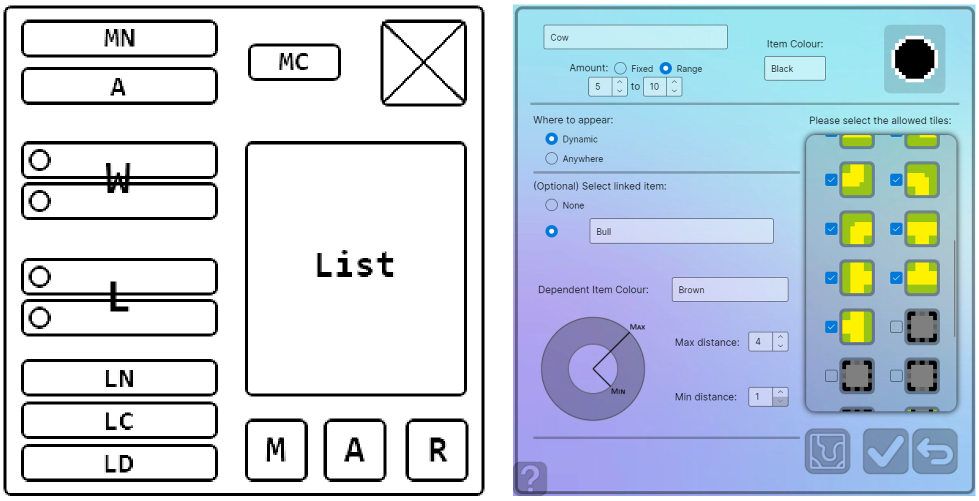
[13]CSS colour keywords - https://www.w3.org/wiki/CSS/Properties/color/keywords


miWFC - Item Mode

Item	#	Where	Constraints
● Treasure	10		Appears on:
● Cow	5 - 10		Appears on: Linked item: ● Bull Appearing distance: 1 to 4
● Duck	5 - 10		Appears on:

FIGURE 4.16: Post-Processing subwindow screenshot

object (LD). Finally, the user can exit this menu (R) and all progress will be discarded. A screenshot of object creation is shown in Figure 4.17(b).



Cow

Item Colour: Black

Amount: Fixed [✓ Range] 5 to 10

Where to appear: [✓ Dynamic] Anywhere

(Optional) Select linked item: None [✓ Bull]

Dependent Item Colour: Brown

Max distance: 4

Min distance: 1

Please select the allowed tiles:

(a) Wireframe (b) Screenshot

FIGURE 4.17: Object creation interface

4.8 Additional Features and Options

The additional features and options are covered by a single control each, as seamlessness and input wrapping are a toggle, and pattern size a value. Importing and exporting of both algorithm output and spatially variable weight mappings are handled through a system dialogue to allow selecting or save files independently.

These additional advanced features have been added to the interface in Figure 4.18, a screenshot of the final window shown in Figure 4.19. On the input side, the user can toggle all advanced features (A), showing toggles for input wrapping (W) and pattern size (PS) (if in the overlapping model), and the seamlessness toggle (S) on the output side. Buttons for importing (I) and exporting output (E) are placed on the top right of the output side.

Buttons for opening the spatially variable weights window (V) and to reset weights (R) are added above the algorithm manipulation section. Finally, buttons to switch to direct manipulation (D) and post-processing (P) are added on the bottom right.

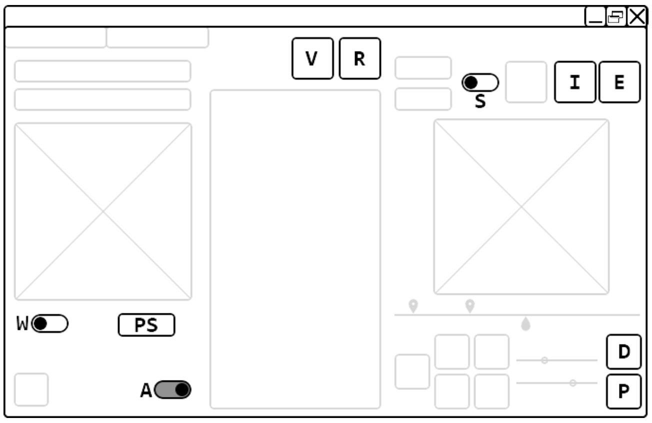


FIGURE 4.18: Final main window additions

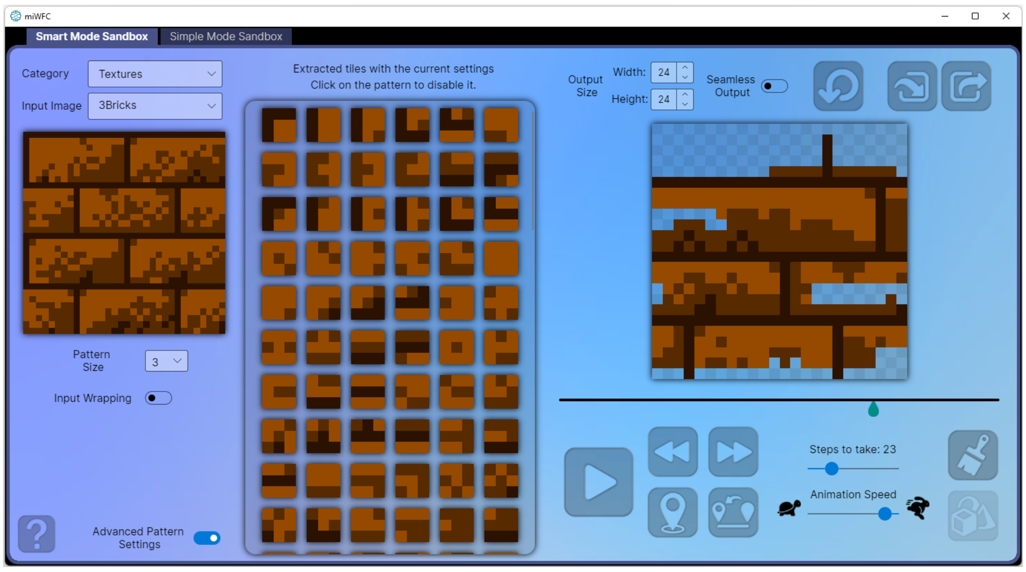


FIGURE 4.19: Final main window screenshot

5 Evaluation and Results

In this chapter, we describe how our evaluation is structured in order to assess the intuitiveness of our designed methods. Afterwards, we discuss the results and findings of each of the conducted tests. The goal of these tests is to validate whether participants are able to use the designed methods to reach goals set by a number of tasks, with a minimum amount of prerequisite knowledge.

5.1 Methodology

To test our methods for intuitiveness, we conducted four user tests in total. This was decided upon after concluding on the first user test, performed over the course of three weeks. Four was deemed a sensible amount, both in order to have enough time for participants to respond, and to properly process and implement changes based on feedback provided.

For these tests, we approached creative professionals from online platforms (e.g, Discord[14]), and contacted game studios, asking for game designers and artists. The decision to invite creative professionals rather than the public was due to these individuals being the most probable target users for the methods designed.

A lot of designers prefer to use macOS over Windows and Linux based systems [37] (with Linux being very unpopular), as Windows renders for readability, whilst macOS renders for visual appearance [18]. Due to this, we would like to have the ability for macOS users to participate in user tests as well. For this reason, we provided a prototype for both operating systems.

User Test Structure

For each test we compiled executables and recorded an explanatory and instructive video on the new features. Each test contained up to four tasks to complete. Tests were done remotely, and did not require our presence, as an explanatory video was provided. If presence were required to provide guidance, it would obstruct exploration, and trial and error. Additionally, designers and artists do generally not like being told what to do[15], and the tasks designed therefore contained little constraints.

During each test, we asked demographic questions to “warm up” participants for the user test [27] and help us explain potentially unexpected results [8] as each participant can approach a question differently (Appendix section A.1.1).

The designed interaction techniques were evaluated on intuitiveness through a set of tasks. We asked participants to score each method’s intuitiveness from 1 (very contrived) to 5 (perfectly intuitive) [25], followed up by an open question to elaborate on the score, providing insights into why a method was (not) deemed intuitive.

[14]Discord – `https://discord.com/`

[15]`https://helloartsy.com/8-comments-artists-hate/`

Finally, each test concluded with debriefing questions (Appendix section A.1.2). Through these, we extracted opinions and ideas on missing functionality.

5.2 First User Test – History Navigation

The first test focused on navigational controls without direct manipulation, as these methods were not yet developed. All questions can be found in Appendix section A.2.

In the first task, we asked participants to create a top-down representation of a city using the navigational tools (Section 3.1) provided. The existence of bugs in this first prototype was a likely scenario, hence we asked feedback on whether participants were unable to perform certain tasks, either due to unintelligibility, not being properly explained, or broken.

The second task regarded seamlessness (Section 3.5.1) of the output to create satisfactory game textures that could be used on a large surface.

A final (optional) "sandbox" task was included for the exploration of features without constraints, and provided the ability to upload creations and the incentive behind them.

5.3 Second User Test – Direct Manipulation

The second test was centred around direct manipulation and tile manipulation. To encourage exploration and creativity, participants were given large freedom, and were only restricted to using a specific input image in order to validate their understanding of the task. All questions can be found in Appendix section A.3.

The first task revisited city generation from the first test, approaching it with more controllability. Participants were asked to steer generation towards their desired arrangement with the use of direct manipulation (Section 3.2).

Tile manipulation (Section 3.3) was approached in the second task using any input image. Participants were allowed to play with weights extracted from the input. They were then asked to correlate the appearance of their generated output with the values they had assigned, promoting exploration of the effect of different values.

Finally, instead of a regular "sandbox" task where participants could do whatever they wanted, we wanted to touch upon methods and designs that were changed since the first test to evaluate whether these changes were considered an improvement. Altered methods were input wrapping, pattern size, seamlessness, placing and loading of markers, and the associated timeline.

5.4 Third User Test – Tile Manipulation

The third and penultimate test focused on pattern manipulation, and static and spatially variable tile manipulation. Similar to previous tests, we designed our questions and tasks around providing creative freedom with minimal restrictions, and all questions can be found in Appendix section A.4.

Our first task regarded static tile manipulation (Section 3.3), with a revised set of controls compared to the second test. The second task was similar to the first, but instead asked participants to alter weights using spatial variability.

In the third task, we tasked participants with toggling tile symmetries. This feature is situational, and asked participants on the intuitiveness of both toggling and locking symmetries independently.

The fourth task asked participants to toggle patterns in a given input image. We additionally explained why and how this feature can cause problems if used carelessly. This was done after the participant had completed the task, to reflect on their understanding of this edge case.

The "sandbox" task did not have tools requiring re-evaluation and was included for the exploration of features without constraints, similar to the first test.

5.5 Fourth User Test – Customizability

The fourth and final user test centred around customizability, such as the creation and placement of patterns, objects, and custom input images. This test provided participants with the least constraints and also evaluated all methods combined. The questions can be found in Appendix section A.5.

The test was composed of four tasks, with the first focused on templating (Section 3.2.3). We tasked participants with creating and placing a template of a castle, followed by multiple placements of this created template.

Secondly, we tasked the participants with creating custom input images. After creating their own image through an image editing application, we asked whether the output, using their input image, was yielded as intended, or had undesired features. This was followed up by reflecting on these undesired features.

The third task centred around placing custom objects on generated output. After asking for elaboration on why they placed each object, we asked whether the algorithm created the object layer as intended.

Finally, to evaluate all developed methods and their intuitiveness, we tasked the participants with an unrestrained task, open to interpretation; *"Please create an image that describes you as a person, your character, your environment, or your area of expertise."*

5.6 Results and Discussion

In this section, we will review and discuss participant feedback from each of the conducted user tests individually, closing with an overview of the series of user tests as a whole.

5.6.1 First User Test – History Navigation

The first user test was performed by 13 individuals of varying professions and expertise. All responses can be found in Appendix section B.1, accompanied by graph representations of the data for improved readability (Figure C.1).

The majority of participants rated history navigation positively (Graph 5.1 - Task 1), with an average score of 3.54 on our 1 to 5 scale, which is above average. Some minor improvements were added, such as (i) a timeline to visualize algorithm progress including placed markers, (ii) the ability to place more than one marker, and to (iii) provide more control to participants, all of which were requested and added in following tests.

Seamlessness was regarded as a useful feature (scoring 3.54 on average as well, Graph 5.1 - Task 2) even though hidden behind a button with complicated text. This was later changed to a toggle button as per participants' suggestions.

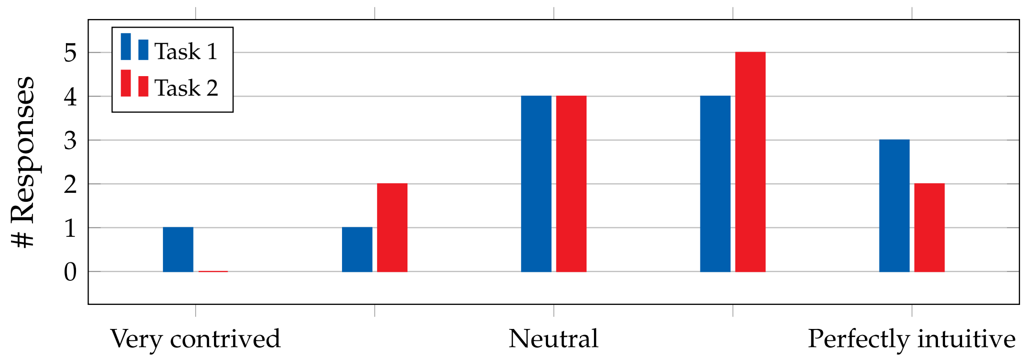


FIGURE 5.1: First User Test Task Results

Albeit optional, only one participant did not participate in the sandbox task, a good sign of interest sparked among participants. In general, the prototype was very well received by the participants, who gave valuable feedback and expressed interest in participating in subsequent test sessions.

5.6.2 Second User Test – Direct Manipulation

In the second test, 11 people participated. Appendix Section B.2 includes responses and graphs of the data (Figure C.3).

The direct manipulation tools were received very well, averaging a score of 3.64 (Graph 5.2 - Task 1). Participants indicated better intuitiveness if effects from each tool were able to be previewed whilst hovering over the output (added in the third test). Additionally, participants asked whether it was possible to allow for painting over already collapsed areas, which later evolved into "overwriting" (Section 3.2.1).

Tweaking tile weights and its effect on the output was quickly grasped by participants. Overall, these tasks yielded mostly positive feedback (Graph 5.2 - Task 2), even though this question was accidentally asked in a binary fashion, rectified in the third user test. Most feedback regarded the UI representation, as the amount of clicks required to alter weights was deemed excessive, adjusted in the fourth test to a single click with modifier keys.

Participants were particularly enthusiastic about markers on the timeline (Graph 5.2 - Sandbox), improving our previous grade (3.54) to a 4.40.

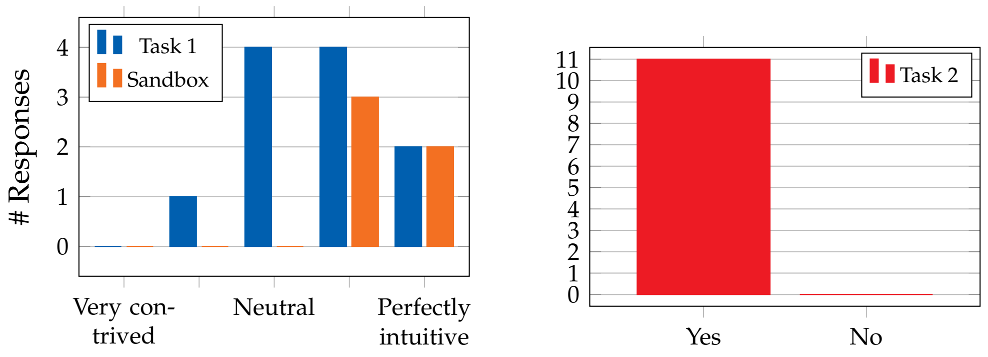


FIGURE 5.2: Second User Test Task Results

5.6.3 Third User Test – Static and Dynamic Tile Manipulation

The third user test was performed by 17 individuals, and focused on four types of tile manipulation provided by *miWFC*. The results of this user test can be found in the Appendix, both in written (Appendix B.3) and graph format (Figure C.5).

We re-touched and revised controls for tile manipulation, and participants went on to experiment with their finer control. Overall, tile manipulation was considered sufficiently intuitive, with mostly positive results, averaging a score of 3.76 for static weight manipulation (Task 1, Graph 5.3), and 3.82 for spatially variable weight manipulation (Task 2, Graph 5.3).

Interestingly, spatially variable weight manipulation scored higher compared to static weight manipulation, even though the textual feedback provided by the participants stated that the feature was somewhat hidden. This, however, deemed to be less obstructive compared to the mechanic of tweaking weights (setting the amount of change manually, followed by actually changing the weight), later changed to a single click with key modifiers.

Pattern symmetry was additionally received very well, averaging a score of 3.94 for toggling symmetries (Task 3 – Q1, Graph 5.3), and a score of 3.71 for defining the tile orientation itself (Task 3 – Q2, Graph 5.3). Whilst some participants directly correlated the effect of each button in the output, others were slightly puzzled by the button arrangement or representations prior to using them, but confessed that using them produced clarity.

Finally, pattern toggling was received well, averaging a score of 4.71 (Task 4, Graph 5.3). The only feedback received on this feature was regarding prediction of the effect on the output, or not understanding why visually analogous patterns did not yield a similar result when toggled independently.

5.6.4 Fourth User Test – Customizability

The fourth and final user test was completed by 11 individuals, the decrease in participants likely linked to the holiday season. Nonetheless, the test was fruitful, and centred around multiple intricate types of customization, such as templating, post-processing and creation of custom input images. All responses can be found in Appendix section B.4, accompanied by graph representations of the data for easy readability (Figure C.7).

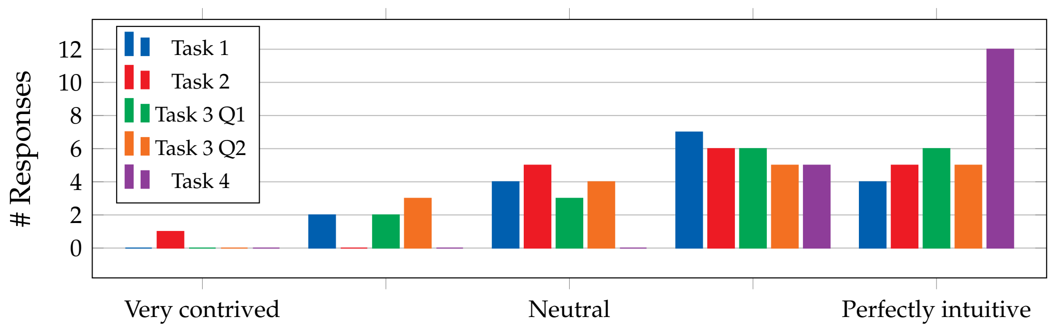


FIGURE 5.3: Third User Test Task Results

Pattern creation and placement were deemed powerful and intuitive features, and scored an average of 3.73 (Task 1, Graph 5.4). Participants deemed single cell selection tedious, originally intended for accuracy. Brushing and a clearer indication of the created template were most commonly requested. A trend among feedback were comments on other methods, used to complete this task, likely having influenced the score.

Creating custom input was received well, even though being fairly complicated, scoring an average of 3.91 (Task 2, Graph 5.4). Most negative feedback originated from disregard of task restrictions; maximum image size and amount of colours, as increasing these can yield long generation times. For these restrictions, a follow-up question was included about the arduous nature of custom image creation, to which participants explained their difficulties.

Even though Post-Processing was praised in the feedback, it scored an average of 3.64 (Task 3, Graph 5.4). The reason this feature received lesser scores, was due to the unrefined nature, as it was open to improvement and extension. Specific feedback included expected features such as: retrying a single item instead of all or multiple linked items.

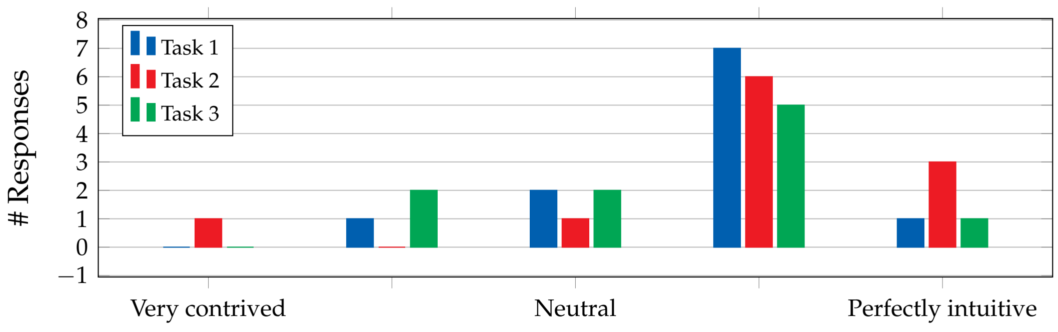


FIGURE 5.4: Fourth User Test Task Results

The final task was ungraded, as it was a subjective task requiring the participants to use all methods available to them, and feedback overall was positive, participants even creating entire stories around their final generations, or trying to generate real-life art such as Piet Mondriaan's Compositions[16].

[16]Composition II in Red, Blue, and Yellow, 1930, Kunsthaus Zürich - https://w.wiki/5cvv

5.6.5 General Overview

Overall, results yielded from the user tests were positive, with negatives often pointing out features that were being developed, but not yet present in the current version. As shown by participant feedback in Appendices B and C, features that were updated in subsequent tests predominantly increased scoring.

As each test was completed by a different number of participants, we have converted the responses of our user tests to percentages for each of the five scores, to provide a better analysis of the evaluation as a whole, shown in Figure 5.5. Across all tests, 68.7% of replies were *intuitive* (4) or *perfectly intuitive* (5). There is certainly room for improvement, but most feedback reassured us that the mixed-initiative techniques were considered useful by the community of testers.

Pattern manipulation was the most popular feature, illustrated by the large green peak on the right of the histogram. This feature was deemed as a very powerful tool to manipulate the algorithm output, with very little effort required. On the opposite side of the spectrum, the blue bars in the histogram show the first iteration of history manipulation tools and were then considered less intuitive, most likely related to the initial design, with a very programmer-centred perspective on design, using complicated terms, and non-self-explanatory interface elements.

Issues and negative feedback on interface design were mostly related to a lack of UX-design experience, sometimes designed from a programmer perspective instead of a user perspective, such as the use of complicated terms. Thankfully, participant feedback steered us into the right direction. Through multiple iterations, most interface elements eventually landed on a satisfactory and intuitive final design.

We can deduce that the designed methods successfully provided intuitively control, seldom causing confusion to the point of inoperability. More often than not, stated usability issues were caused by neglection of explanatory videos, later confessed when asked *"After watching the introductory video, was anything unclear?"*.

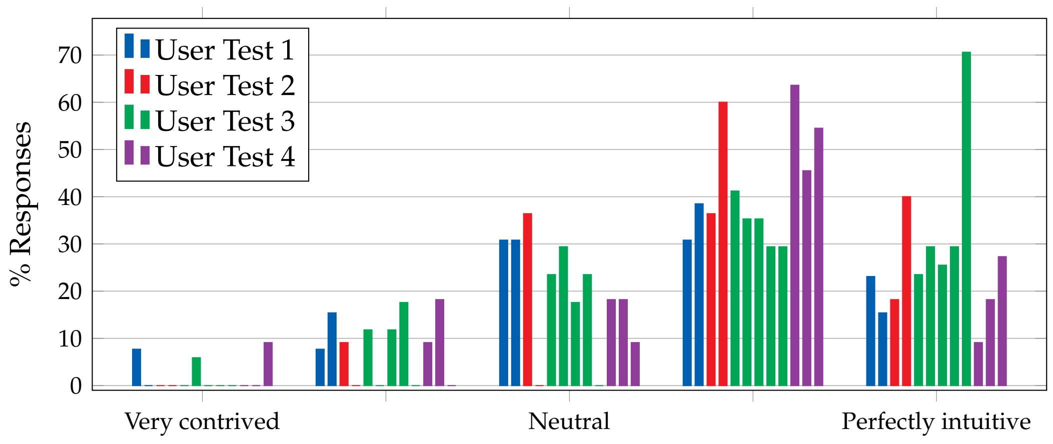


FIGURE 5.5: All User Test Task Results in Percentages

6 Conclusion

Wave Function Collapse is a powerful and promising PCG algorithm, but lacks interactive features and intuitive control, indispensable for the work of most creative professionals. In this thesis, we presented methods that convert WFC into a powerful mixed-initiative PCG method. This approach provided abundant interactive features and intuitive control mechanisms for artists and designers, to effectively assist them in their creative work.

Among other features, *history navigation* explores the iterative nature of the WFC algorithm, and animates its "flood progression", promoting trial-and-error experimentation over a timeline, by means of an intuitive undo mechanism. These features were deemed sufficiently intuitive and got even more so in following iterations.

Direct output manipulation overrules, whenever desired, the automatic WFC cell and tile selection step, replacing it by explicit user selection, and allowing for an effective steering of the generation towards the actual design intent. Based on user feedback, this feature was most prominent in providing control over the algorithm process.

Tile manipulation permits overruling both weights and orientations of tiles identified during algorithm input. *Pattern manipulation*, in turn, provides powerful filtering control over the patterns extracted from the input. Both of these provide users with fine-grained methods for tweaking the global appearance of the output, steering it away from the input in a controlled manner.

Although weight manipulation was initially deemed the least intuitive feature, subsequent iterations converted it into one of the most simple-to-understand features. Pattern manipulation was found to be the least complicated feature to use overall, whereas transformation manipulation, whilst useful, proved to be a less frequently requested modification of the original algorithm.

With *Post-Processing*, an initial application example of WFC output was provided. Although open to numerous extensions and improvements, it sufficiently provides the ability to generate custom objects. Even though being more of a detached addition than an extension, user review validated our expectations of it needing more polishing, but the concept behind it was praised.

Finally, various additional features and options provided the user with wider control over the parameters used, and granted users the final steps towards complete control over the generative process. The most prominently praised feature was custom input images, equipping users with complete control of algorithm output.

Through evaluation with a group of possible end-users, we can conclude that we successfully turned WFC into a mixed-initiative PCG method, making it usable for non-programmers to assist them in their work, and proved to be an effective tool to explore the generative space of the WFC algorithm.

7 Future Work

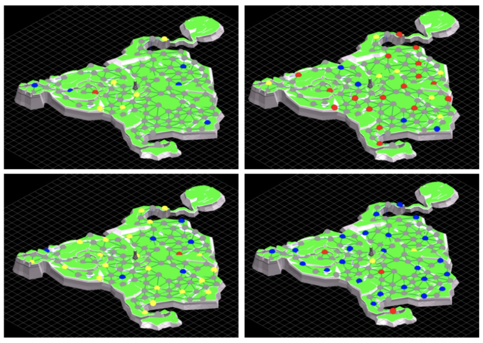
Various other features could provide significant improvements towards facilitating creative expression of artists and designers. These features are deemed future work, as there was not enough time to extend the thesis with these ideas, or the scope of these features is large enough to be considered a project on its own.

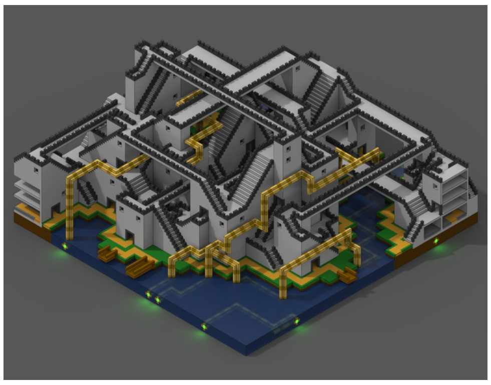
7.1 Non-Regular Grid Extension

The basic implementation of WFC is designed around a grid. Research conducted by Kim and Kang [17] extend WFC into the graph-based domain, allowing an infinite set of layout possibilities. Graph based domains differ from regular grids as each cell does not have exactly four neighbours, but a different or irregular amount. An example implementation by Kim and Kang is shown in Figure 7.1(a).

7.2 Three Dimensional Extension

Addition of a third dimension has already been introduced by many existing WFC applications such as *TownScaper* [40] and *Bad North* [39]. Adding this third dimension does introduce additional hurdles, such as clicking on the output in 3D space, as a screen remains two-dimensional. An example is shown in Figure 7.1(b).


(a) WFC with a graph domain. [17]


(b) 3D application of WFC by M. Gumin. [10]

7.3 Negative and Additive Input Images

Karth and Smith [16] introduce negative input. Their implementation uses multiple "positive" input images to support more exotic output without altering the original input image. Patterns are then altered by excluding patterns yielded from "negative" input images (Figure 7.1).

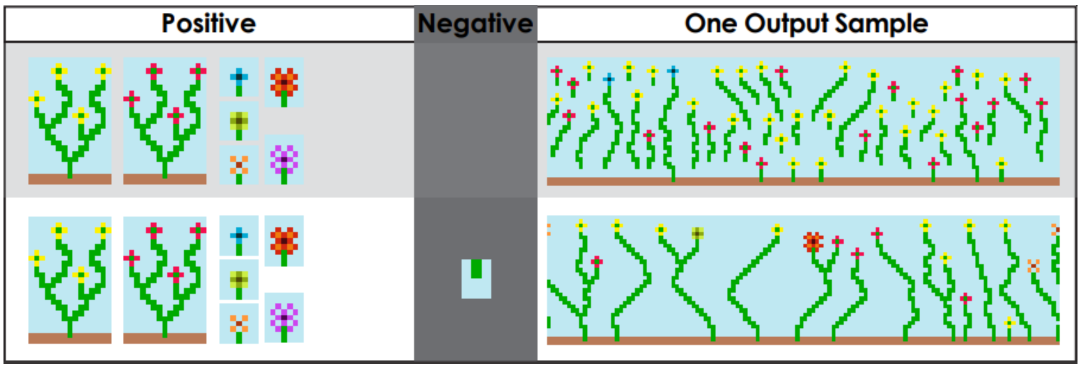


FIGURE 7.1: Input examples, one with added negative input. [16]

7.4 Smart Constraint Learning

A very interesting and creative but potentially harmful feature, is introducing smart constraint learning. This concept would allow the user to paint every tile anywhere, even if extracted constraints would originally not allow the user to do so.

With this option, user created adjacencies will be added to the existing constraints. This, however, can potentially be troublesome, as these created constraints could yield unwanted output further in the process. However, when used with caution, it could allow appending to the input, where the current tools only allow filtering.

7.5 Additional post-processing detail

The current implementation of post-processing is only touching the surface of what is possible. Of course, this was purely added to show what could be done with algorithm output, but would not add to the scope of the project if refined further.

When thinking beyond the scope, a few additional features come to mind, such as a ratio between linked objects, e.g., for each 5 trees added, an axe will appear. This can also be added in reverse, where we add 5 birds if a bird's nest has been added. Also, multiple linkages could be a powerful feature, e.g., when adding a princess to a castle, both a dragon and a treasure will be added to said castle.

7.6 Connectivity between two or more points

Whilst some top-down input images generate exactly what is requested, some lack the possibility for connectivity between a theoretical starting and ending position in the output. Whether it be an actual road being able to transport a player from location A to B, defining that a mountain ridge should be unobstructed between two points, or a river that has to follow a desired route.

By selecting two or more cells in the output, including a connectivity constraint (e.g., a straight or curved line or A. Newgas' chiseling technique [41]) and defining which tiles should be incorporated, the user will be able to achieve this goal.

7.7 Multi-layer generation of natural features

When generating output using a top-down world as input, some natural terrain objects could add to the feel of the generated level. Whilst being more general than this definition, it was already touched upon by [31] and implemented by [4].

The user is currently able to place objects that they define, which usually influence the game level when actually being played. Natural objects, not influencing the game level (trees, rocks, flowers...), could be placed during generation of the underlying world, e.g., each time the WFC algorithm collapses a tile, randomness, or user defined chances can cause trees or flowers to appear.

7.8 Nested WFC

Nested generation could introduce the possibility to generate very large worlds, without the downside of the level being composed of small regions of interest, based on the size of the patterns used.

Instead, if we were to generate a "raw" world layout, for example defining biomes or countries, we can then "zoom in" on a group of cells, and expand this selection to be its own output grid of cells, generate this region with a second level of detail, e.g., subdividing it into grass, forest, hill and mountain tiles.

This can be iterated by zooming in and actually placing trees, fallen logs, ponds, flowers et cetera. Whilst this example is based on a three-layer world (Figure 7.2), the actual amount of options is indefinite, spanning from world generation to house creation and furnishing.

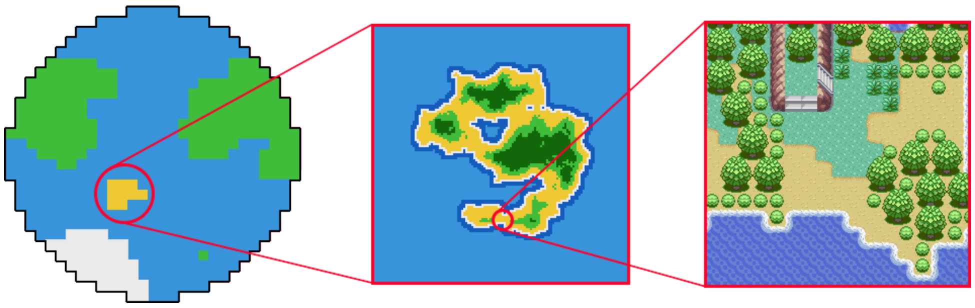


FIGURE 7.2: Nested world generation.

Bibliography

[1] Alberto Alvarez et al. "Fostering Creativity in the Mixed-Initiative Evolutionary Dungeon Designer". In: *Proceedings of the 13th International Conference on the Foundations of Digital Games*. FDG '18. Malmö, Sweden: Association for Computing Machinery, 2018. ISBN: 9781450365710. DOI: 10.1145/3235765.3235815.

[2] Alberto Alvarez et al. "Questgram [Qg]: Toward a Mixed-Initiative Quest Generation Tool". In: *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*. FDG'21. Montreal, QC, Canada: Association for Computing Machinery, 2021. ISBN: 9781450384223. DOI: 10.1145/3472538.3472544. URL: https://doi.org/10.1145/3472538.3472544.

[3] Boris the Brave. *DeBroglie Documentation*. 2022. URL: https://boristhebrave.github.io/DeBroglie/index.html.

[4] Darui Cheng, Honglei Han, and Guangzheng Fei. "Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm". In: *International Conference on Entertainment Computing*. Springer. 2020, pp. 37–50.

[5] Rina Dechter and Daniel Frost. "Backjump-based backtracking for constraint satisfaction problems". In: *Artificial Intelligence* 136.2 (2002), pp. 147–188. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(02)00120-0. URL: https://www.sciencedirect.com/science/article/pii/S0004370202001200.

[6] Omar Delarosa et al. "Mixed-initiative level design with rl brush". In: *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*. Springer. 2021, pp. 412–426.

[7] *Entropy (information theory)*. 2022. URL: https://en.wikipedia.org/wiki/Entropy_(information_theory).

[8] Arlene Fink. *How to ask survey questions*. Vol. 1. Sage, 2002.

[9] Berndt Fritzke. "Growing Grid — a self-organizing network with constant neighborhood range and adaptation strength". In: *Neural Processing Letters* 2.5 (1995), pp. 9–13. DOI: 10.1007/bf02332159.

[10] Maxim Gumin. *Procedural voxel models synthesized with the wave function collapse algorithm, and rendered in #MagicaVoxel*. 2022. URL: twitter.com/exutumno/status/781837058486726656.

[11] Maxim Gumin. *Wave Function Collapse Algorithm*. Version 1.0. Sept. 2016. URL: https://github.com/mxgmn/WaveFunctionCollapse.

[12] Toby Howison et al. *Reality-assisted evolution of soft robots through large-scale physical experimentation: a review*. Sept. 2020.

[13] Lawrence Johnson, Georgios N. Yannakakis, and Julian Togelius. "Cellular automata for real-time generation of infinite cave levels". In: (Sept. 2010). DOI: 10.1145/1814256.1814266.

[14] Daniël Karavolos, Anders Bouwer, and Rafael Bidarra. "Mixed-Initiative Design of Game Levels: Integrating Mission and Space into Level Generation." In: *FDG*. 2015.

[15] Isaac Karth and Adam M. Smith. "WaveFunctionCollapse is Constraint Solving in the Wild". In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*. FDG '17. Hyannis, Massachusetts: Association for Computing Machinery, 2017. ISBN: 9781450353199. DOI: 10.1145/3102071.3110566. URL: https://doi.org/10.1145/3102071.3110566.

[16] Isaac Karth and Adam M. Smith. "WaveFunctionCollapse: Content Generation via Constraint Solving and Machine Learning". In: *IEEE Transactions on Games* PP (May 2021), pp. 1–1. DOI: 10.1109/TG.2021.3076368.

[17] Hwanhee Kim et al. "Automatic Generation of Game Content using a Graph-based Wave Function Collapse Algorithm". In: Aug. 2019, pp. 1–4. DOI: 10.1109/CIG.2019.8848019.

[18] Austin Knight. *Why Do Designers Prefer Macs?* 2022. URL: https://austinknight.com/writing/designers-prefer-macs.

[19] Antonios Liapis, Georgios N. Yannakakis, and Julian Togelius. "Sentient Sketchbook: Computer-Aided Game Level Authoring". In: *Proceedings of the 8th Conference on the Foundations of Digital Games*. 2013, pp. 213–220.

[20] Bo Lin, Wassim Jabi, and Rongdan Diao. "Urban Space Simulation Based on Wave Function Collapse and Convolutional Neural Network". In: *Proceedings of the 11th Annual Symposium on Simulation for Architecture and Urban Design*. SimAUD '20. Virtual Event, Austria: Society for Computer Simulation International, 2020.

[21] Paul Merrell and Dinesh Manocha. "Model synthesis: A general procedural modeling algorithm". In: *IEEE Transactions on Visualization and Computer Graphics* 17.6 (2010), pp. 715–728.

[22] Microsoft. *MVVM Design Pattern*. 2022. URL: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10).

[23] Tobias Møller and Jonas Billeskov. "Expanding Wave Function Collapse with Growing Grids for Procedural Content Generation." PhD thesis. May 2019. DOI: 10.13140/RG.2.2.23494.01607.

[24] Adam Newgas. "Tessera: A practical system for extended WaveFunctionCollapse". In: *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*. 2021, pp. 1–7.

[25] Jakob Nielsen. *Usability heuristics for user interface design*. 10.

[26] OODesign. *Design Patterns | Object Oriented Design*. 2022. URL: https://www.oodesign.com/.

[27] Cindy Passmore et al. "Guidelines for constructing a survey". In: *FAMILY MEDICINE-KANSAS CITY-* 34.4 (2002), pp. 281–286.

[28] Mijael R. Bueno Perez, Elmar Eisemann, and Rafael Bidarra. "A Synset-Based Recommender Method for Mixed-Initiative Narrative World Creation". In: *Interactive Storytelling – Proceedings of ICIDS 2021*. LNCS 13138. Cham: Springer, 2021, pp. 13–28.

[29] *PNG (Portable Network Graphics) Specification*. 2022. URL: https://www.w3.org/TR/PNG-Structure.html.

[30] *Post-process | Meaning & definition for UK English*. 2022. URL: https://www.lexico.com/definition/post-process.

[31] Arunpreet Sandhu, Zeyuan Chen, and Joshua McCoy. "Enhancing Wave Function Collapse with Design-Level Constraints". In: *Proceedings of the 14th International Conference on the Foundations of Digital Games*. FDG '19. San Luis Obispo, California, USA: Association for Computing Machinery, 2019. ISBN: 9781450372176. DOI: 10.1145/3337722.3337752. URL: https://doi.org/10.1145/3337722.3337752.

[32] Ruben M. Smelik et al. "A declarative approach to procedural modeling of virtual worlds". In: *Computers & Graphics* 35.2 (2011), pp. 352–363.

[33] Ruben M. Smelik et al. "A survey on procedural modelling for virtual worlds". In: *Computer Graphics Forum* 33.6 (2014), pp. 31–50.

[34] Ruben M. Smelik et al. "Integrating procedural generation and manual editing of virtual worlds". In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. 2010, pp. 1–8.

[35] Ruben M. Smelik et al. "Interactive creation of virtual worlds using procedural sketching." In: *Eurographics (short papers)*. 2010, pp. 29–32.

[36] Gillian Smith, Jim Whitehead, and Michael Mateas. "Tanagra: A mixed-initiative level design tool". In: *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. ACM. 2010, pp. 209–216. DOI: 10.1145/1822348.1822376.

[37] Jon Sorrentino. *Why 99.9% Of Graphic Designers Choose Mac VS PC*. 2022. URL: https://www.wellfedcreatives.com/article/why-graphic-designers-use-mac.

[38] Oskar Stålberg. *Art by Oskar Stålberg*. 2022. URL: oskarstalberg.tumblr.com/.

[39] Oskar Stålberg. *Bad North - A Minimalistic, Real-Time Tactics Rogue-like with Vikings*. 2022. URL: https://www.badnorth.com/.

[40] Oskar Stålberg. *Townscaper - WebGL interactive demo*. 2022. URL: https://oskarstalberg.com/Townscaper/.

[41] Boris The Brave. *GitHub - BorisTheBrave/chiseled-random-paths: Generates random tile-based paths with a simple novel algorithm*. 2022. URL: https://github.com/BorisTheBrave/chiseled-random-paths.

[42] Roland Van der Linden, Ricardo Lopes, and Rafael Bidarra. "Designing procedurally generated levels". In: *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2013.

[43] Sean P. Walton, Alma As-Aad Mohammad Rahat, and James Stovold. "Mixed-Initiative Procedural Content Generation using Level Design Patterns and Interactive Evolutionary Optimisation". In: *ArXiv* abs/2005.07478 (2020).

[44] Georgios N. Yannakakis, Antonios Liapis, and Constantine Alexopoulos. "Mixed-initiative co-creativity". In: *FDG*. 2014.

[45] Jichen Zhu et al. "Explainable AI for Designers: A Human-Centered Perspective on Mixed-Initiative Co-Creation". In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. 2018, pp. 1–8. DOI: 10.1109/CIG.2018.8490433.

A Questionnaire Questions

A.1 General Questions

The general questions were asked prior to, and after every user test.

A.1.1 Prior to the user test

What is your relation to content generation?

- Programmer
- Artist
- Designer
- Other…

How many years of experience do you have in this/these field(s)?

- <1
- 1-2
- 2-4
- 4-6
- 6-10
- 10-15
- 15+

How have you been contacted to participate in this project?

- Employee Game Development Company
- Online platforms such as Discord
- Friends or Family
- Other…

Which platform will you be using? *(Asked from the second user test onwards)*

- Windows
- MacOS

A.1.2 After the user test

After watching the introductory video, was anything unclear, that could not be solved by using the info button?

Open Question

Do you have any other feedback on the current interaction methods?

Open Question

A.2 First User Test

A.2.1 Task 1 - Navigation & human interaction

Task Description: create a part of a plausible city map, by controlling the steps of the generator (hence and forth) until you are satisfied with the result.

How much sense did the navigation controls make to you?

- 1 (Very unclear)
- 2
- 3
- 4
- 5 (Very clear)

What issues did you encounter when using the navigation controls, if any?

Open Question

In case the controls were not fully satisfactory, which improvements would you suggest (e.g., providing different functionality of individual controls, better naming, using different visuals,...)?

Open Question

A.2.2 Task 2 – Game texture generation

Task Description: create a satisfactory seamless game texture that you would use on a large surface.

Was it logical that toggling the boundary padding would result in a seamless texture? If not, why?

Open Question

How intuitive did you find the visual aid (image) for toggling boundary padding?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

Would you like to elaborate your given score? E.g., how you'd like to see it improved.

Open Question

A.2.3 Sandbox (optional)

Task Description: This third task allows you to explore all functionalities of the tool in a sandbox. For this reason, this task can take as much or little time as you desire.

Would you like to perform the third task?

- Yes
- No

Play around in the Sandbox (e.g., selecting an interesting input, regenerating at will, adjusting settings, ...) and create some output texture of your liking.

Why did you create the shared images? What was the incentive to create these?

Open Question

A.3 Second User Test

A.3.1 Task 1 – Direct steering of the output

Task Description: Create a part of a plausible city map, by controlling the steps of the generator, as well as using the painting editor. You can export the generated city using the export button.

How intuitive did you find the painting editor?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

Would you like to elaborate your given score? E.g., how you'd like to see it improved.

Open Question

A.3.2 Task 2 – Tweaking pattern occurrence

Task Description: Play with the numbers found associated with the extracted patterns, generating an output for various combinations of values.

Is it immediately apparent what effect these values next to the patterns have on the output? (Try setting very large values) If not, why?

Open Question

Was it easy to understand how to tweak a value?

- Yes
- No

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, ...)

Open Question

A.3.3 Sandbox (optional)

Task Description: This third task allows you to explore all functionalities of the tool in a sandbox. For this reason, this task can take as much or little time as you desire.

Would you like to perform the third task?

- Yes
- No

Play around in the Sandbox and create some output textures of your liking. Please also use the following features:

- Input Wrapping
- Seamless Output
- Placing and loading of markers, and the timeline associated

---

Why did you create the shared images? What was the incentive to create these?

Open Question

Was it logical that Input Wrapping, Seamless Output & Pattern Size are located under the "Advanced features", if not, why?

Open Question

How intuitive did you find the placing & loading of markers, and the timeline associated?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

Would you like to elaborate your given score? E.g., how you'd like to see it improved. And any issues you encountered?

Open Question

A.4 Third User Test

A.4.1 Task 1 – Tweaking pattern occurrence

Task Description: On the bottom left, enable "Advanced Pattern Toggling". Then, through the use of the arrows presented with each pattern, increase the frequency value of the green grass tile slightly (e.g., 5), and generate a complete output image. Then, significantly increase the value of the grass tile (e.g., 50), and again regenerate the output.

How intuitive did you find the tweaking of the values?

- 1 (Very contrived)
- 2
- 3
- 4

- 5 (Perfectly intuitive)

Would you like to elaborate your given score? E.g., how you'd like to see it improved.

Open Question

Was the appearance of the output image easily relatable to your changes of the frequency value? Why (not)?

Open Question

A.4.2 Task 2 – Tweaking pattern occurrence

Task Description: To the top-right of this pattern section, press the button to reset the patterns to default. Now click on the "1" to the bottom left of the green grass tile. This new window helps you set these frequency values by painting on the output space: yellow means high values, blue means low values. Try to draw a mapping that has areas of very high values, and areas where the value is (almost) 0. Once you have drawn the mapping of the green grass tile, return to the previous screen and re-generate the output to use this mapping.

How intuitive did you find the ability to create a mapping for pattern occurrence values?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, ...)

Open Question

A.4.3 Task 3: Tweaking pattern symmetry

Task Description: As you might have noticed, there is a new button to the right of the tiles, a green button with arrows. As well as some buttons to the right of these coloured buttons. Try clicking on one or more of these buttons and regenerate the entire output to see what they do.

How intuitive did you find the ability to toggle the use of tile symmetries in the output?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

How intuitive did you find the ability to set a tile's orientation, when symmetries are turned off?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, ...)

Open Question

A.4.4 Task 4: Pattern Exclusion

Task Description: Upon toggling the Advanced Pattern Settings in the bottom left of the application, you are now able to click on patterns that you want to exclude from the output. Please disable ALL patterns with one or less "light" sand pixel, and generate a complete output.

How intuitive did you find the ability to exclude patterns from the output?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

Did you encounter infinite generation or inability to generate output after excluding one or more patterns? If you did, did you understand why this happened?

Open Question

A.4.5 Sandbox (optional)

Task Description: This last task allows you to explore all functionalities of the tool in a sandbox. For this reason, this task can take as much or little time as you desire.

Would you like to perform the third task?

- Yes
- No

Please head to either of the tabs, and play around with the miWFC application

---

Why did you create the shared images? What was the incentive to create these?

Open Question

A.5 Fourth User Test

A.5.1 Task 1 – Templating

Task Description: Please open the application and head on to the tab "Simple Mode Sandbox". After having generated (part of) an output, please create a template of a satisfactory castle. After doing so, reset the output to be empty again, and place your template at least 3 times in the output, followed by filling in the rest of the image as you see fit.

How intuitive did you find the creation and placement of templates?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

Would you like to elaborate on your given score? (e.g., any issues you encountered, how you'd like to see it improved, . . .)

Open Question

Subtask: In the pattern creation tool, there is an extra button to select all generated parts, which can for example be used after solely drawing over the output,

to quickly select your drawn creation. Please locate and use it if you haven't encountered it.

Do you find this option a useful addition?

Open Question

A.5.2 Task 2 – Custom Input Images

Task Description: In Microsoft Paint, or any other image editing software, create your own custom input image with at most 4 colours. Then, in the application, select the category "Custom" and press the folder icon to open the directory where you can add your own input images. If your input image is oriented, meaning it has an up and down, place it in the "SideView" folder, else, place it in the "TopDown" folder. Once you have drawn and uploaded your very own input image, play with it, and see how the algorithm interprets your input image.

How intuitive did you find the ability to create and use your own input images?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, . . .)

Open Question

What did you learn from creating and using your own input images, was it hard to create input images that caused the output to appear as you intended?

Open Question

A.5.3 Task 3 – Post Processing

Task Description: Please select any "Top-Down World" or "Custom" input image, and generate an output. Afterwards, please click on the item customization button (bottom right), and add some items to the generated image. Afterwards, please export the image with the items added (regular export button). Note that an item

does not necessarily need to be a "key" or "sword", but can also be a "spawn location" or "checkpoint". Your imagination is the only limitation.

Please elaborate which colour of items represent what type of item or object, and why you decided to add it, including its frequency, location, and possible linked items.

Open Question

How intuitive did you find the ability to add a new layer of custom items or objects into the world?

- 1 (Very contrived)
- 2
- 3
- 4
- 5 (Perfectly intuitive)

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, . . .)

Open Question

A.5.4 Sandbox – Final Task

Task Description: You are now very familiar with the prototype application, and have used every tool to your disposal. Please create an image that describes you as a person, your character, your environment, or your area of expertise. The sky is the limit regarding interpretation! This is a very subjective task, and is centred around the definition of art, being: "An experience consciously created through an expression of skill or imagination".

Why did you create the shared images? What was the incentive to create these?

Open Question

B Questionnaire Results

B.1 First User Test

B.1.1 General Questions – Prior to the user test

What is your relation to content generation?

The first user test accidentally had this question set to allow multiple answers, rather than one, hence the percentages adding up to over 100%.

Option	Responses	Percentage
Programmer	6	46.2%
Artist	4	30.8%
Designer	5	38.5%
Other…	1 (Student)	7.7%

How many years of experience do you have in this/these field(s)?

Option	Responses	Percentage
<1	0	0%
1-2	3	23.1%
2-4	3	23.1%
4-6	2	15.4%
6-10	3	23.1%
10-15	0	0%
15+	2	15.4%

How have you been contacted to participate in this project?

Option	Responses	Percentage
Employee Game Development Company	5	38.5%
Online platforms such as Discord	5	38.5%
Friends or Family	3	23.1%

B.1.2 Task 1 – Navigation & human interaction

How much sense did the navigation controls make to you?

Option	Responses	Percentage
1 (Very contrived)	1	7.7%
2	1	7.7%
3	4	30.8%
4	4	30.8%
5 (Perfectly intuitive)	3	23.1%

What issues did you encounter when using the navigation controls, if any?

Responses
I didn't understand how I could control the output. Output seemed only to determined by a random seed? Not sure how I could create a satisfying city with it. (In fact I couldn't)
too little control over what the program is doing. Kept trying to fill the field with red buildings, and the only thing I could do was. Go back a step and hope going forward again would not generate a red building. Reducing the step size also helped, but that was found by quite a lot of trial and error.

In case the controls were not fully satisfactory, which improvements would you suggest (e.g., providing different functionality of individual controls, better naming, using different visuals,...)?

Responses
Within the example of generating a city, to have more freedom to generate a specific group of buildings (Red, Green, Blue) would be helpful.

While small, I suggest adding a colour guide in the info section of the program (ex: red, blue, green). It is listed in previous description, but this would help to quickly refresh the mind when constructing the layout. If the program were to be expanded to more in depth creation, it would be useful to work on specific segments at a time, such as the upper right-hand corner. If splitting into, say, 9 segments this can allow for better customization within the algorithm. While this can be useful, I only suggest as an option and not as a mandatory approach.
There is little control over the output. For example, I wanted less red buildings but because there is a 3×3 red square the algorithm favours it. Being able to remove it from the patterns or even lower its probability would allow for a bit more control
A history of marker that you can switch to would be nice
Controls were clear, info button was helpful as well. Perhaps the only thing is to better explain / show the place new marker?
Make it clear somehow that by navigating it generates something new, instead of it iterating over something already generated (Maybe different icons, slider with highlights, etc)
I like adding the "checkpoints". But I fell like it would be nice to revert the generation and create a checkpoint at different points. For example, I found that it was making almost my whole map "red", but it would make more sense to me if most building were residential (i.e., blue). I would like to go back to before it decided to make everything "red". But I didn't know it would go "wrong" until after it was too late. The "take a specified number of steps backwards" button didn't work for me. Also, the name "Marker" is confusing to me?
I felt as if there were too many public buildings in my generation and too little residency buildings every time and didn't understand how to adjust that. I was also a bit confused on what the marker buttons did, other than save a certain point in your generation and generate something new from there on out. So if possible, maybe go a bit more in depth of the function and use of these two buttons.

B.1.3 Task 2 – Game texture generation

Was it logical that toggling the boundary padding would result in a seamless texture? If not, why?

Responses
Yes
Yes – the information area explained it well
Not really. It did not stand out to me. My focus is mostly on the functions on the bottom right
yes
It was clear, but only because the task was "create a seamless texture" and that was the only additional control on the screen, so it was the only thing to try
Yes, it was logical

Yes it was to me.
I didn't notice that my texture wasn't correct, so maybe it wasn't. Considering UX-design, I think it would be better to show both "icons", and highlight one of them to indicate that one is selected.
Yes, it was logical. I would've noticed it earlier if it was visible on my screen.
I watched the video and that made it clear
Yes, this makes sense to me.

How intuitive did you find the visual aid (image) for toggling boundary padding?

Option	Responses	Percentage
1 (Very contrived)	0	0%
2	2	15.4%
3	4	30.8%
4	5	38.5%
5 (Perfectly intuitive)	2	15.4%

Would you like to elaborate your given score? E.g., how you'd like to see it improved.

Responses
The image is clear enough, though more text may be helpful to explain that toggling the boundary button will create a seamless texture.
Maybe a small "toggle" nearby to indicate. I do see how to toggle in info, but not everyone is curious enough to read through fully. As for the actual toggling image, it is sufficient.
The image makes sense only with an explanation
I would remove the grey pattern outside the boundary when it is switched off
I would find a checkbox with the text "Seamless Wrapping" next to it far more intuitive.
It might be more obvious if the repetition of the texture is more visible which could be done by adding another row of pixels around the border of the image. That might make the image too big, however, especially because it functions as a button.
The boundary toggle does not clearly show some sort of continuing pattern. If it did, I think it would be more clear that, that is what it does.
because both images had a clearly visible border and didn't seem super continuous, I didn't really realize what the icon meant. I would suggest also adding a textual hint (e.g., "Should it be tileable?", followed by the icons. (or maybe I'm still not understanding it correctly?). You could also show images that more directly show the uses of each type of texture (e.g., a wall image or a single flower IDK)
The location of it on the screen makes sense, and the image that is used to visualize the padding looks natural.

The image itself is a bit abstract to me, once I knew what it was, it was fine though

B.1.4 Sandbox

Would you like to perform the third task?

Option	Responses	Percentage
Yes	12	92.3%
No	1	7.7%

Why did you create the shared images? What was the incentive to create these?

Responses
I created the castle & floor plan images due to the fact I felt as if they could be useful within creating designs for my own creative work in Minecraft. Also, the flowers textures seemed really interesting with the different designs and shapes of the plants that were generated, so I liked experimenting there.
The Maze to further understand the seamless mode of texture creation. Which is rather clever. The Skyline to see how the generator works with a more uniform generation.
I was curious how much my pc could handle. It seems that it is quite hard for my PC when running above output image size of 100 × 100
I tried to get at least one good picture out of it? I found it very hard to make anything useful
Testing performance of the algorithm (circles at pattern size 5 does a great deal of effort but ends up not generating much interesting stuff, as it just pastes circles around –> might be interesting to optimize such cases), curious how it would deal with something like a circle or a font, attempting to let it actually generate a circle at pattern sizes < 5 (did not really work out), attempting to generate a really large room by going back and forth
My main incentive was finding out what the algorithm is capable of. Image 1 and 3 were mainly just for fun, with image 3 I was trying to generate a texture that could actually be used as a map for a small game.
3-1 & 3-2 I find the difference between a pattern created with small images vs one with a single larger image interesting. In the small pattern the image stays together (little cats look like cats) but the larger images distort and fall apart. 3-3-& 3-4 I find the potential for generating irregular tiles interesting.
Mostly just exploring the tool, and seeing what I would get out with different input images. I really liked how a simple circle gave me interesting patterns that seem like a good base for some natural structure (like marble or wood, see random_pattern.jpg).

I wanted to experiment with padding on other textures like the "cat" texture. It was also interesting to see the "font" option; I wanted to see how the font generated and if the generation made sense. The generated shapes were interesting to see, I wanted to see if it could come up with a rough "city plan".
I just clicked around a bit, this seemed cool
I didn't really know what I was going for at first, but after a bit of playing around I wanted to create some sort of 'flower horizon' with a bunch of different flowers from all sizes. After playing around with the output image size and the generation, I was satisfied with the result that has been uploaded above.

B.1.5 General Questions – After the user test

After watching the introductory video, was anything unclear, that could not be solved by using the info button?

Responses
Nope
No
It was straight forward enough for myself.
No, everything in the video made complete sense. The approach was quick and to the point, but left no information out. Also, the Info section was helpful and had all needed info.
It was pretty clear
Not really. With both the video and info button almost everything was clear
everything was clear
No, everything was clear.
It was not immediately clear to me in task 1 that after going back (in steps or a save point) you would after generate new pattern content from that point on. (rather than generally the same as before)
I think the info button is a big help and explains everything very well!
Everything was clear to me.
As mentioned in the first step, the placing of markers felt a bit unclear to me. Maybe elaborate more on that. I also didn't really understand why there was already a part of the image generated (a small chunk) before I actually pressed the play button. I'll send you some separate screenshots for clarity.

Do you have any other feedback on the current interaction methods?

Responses
This is a great platform, I would love to see its further implementation into games.
Manual adjustments of the textures

Perhaps explain the tile mode more. Not exactly sure what it adds
I found the interaction very limited. I expected to be able to draw the images myself. That I would have loved, because right now I don't feel I have created anything. I just clicked buttons until I got random images I kind of liked.
I think a slider has potential for the back-and-forth stuff. You can neatly show the checkpoints then as well, maybe even with a preview when you hover over the markers. Obviously there's the issue of not knowing where it ends, but there are probably ways to show/visualize that. I would ditch the step slider and make that a number, in fact, I find myself usually putting this on 20-40, so maybe a nice default can be picked there that "generally works". The animation feature seems pointless, seems easier to just click a couple of times. Could experiment with using the scroll-wheel (back-forth if you hover over the image and scroll), and maybe some form of acceleration (where if you scroll rapidly it will perform more steps per scroll than when scrolling slowly). Could work nicely with some sort of timeline visualization with all the checkpoints on it as well
Don't see a lot of use in the save points. In my recent experience you quickly realize your pattern is going in a direction you don't like and use the steps back. Using save point did not add anything for me.
I would like to have more intuitive ways to go back and forward in the simulation (and maybe create branches/timelines from there that can be recalled).
You can't really guide the creation process it seems
I think there was a chunk missing in the top left corner of your program. I will also send a screenshot separately for clarity.

B.2 Second User Test

B.2.1 General Questions – Prior to the user test

What is your relation to content generation?

Option	Responses	Percentage
Programmer	4	36.4%
Artist	6	54.5%
Designer	1	9.1%
Other...	0	0%

How many years of experience do you have in this/these field(s)?

Option	Responses	Percentage
<1	0	0%
1-2	2	18.2%
2-4	4	36.4%
4-6	3	27.3%
6-10	2	18.2%
10-15	0	0%
15+	0	0%

How have you been contacted to participate in this project?

Option	Responses	Percentage
Employee Game Development Company	5	45.5%
Online platforms such as Discord	3	27.3%
Friends or Family	3	27.3%

Which platform will you be using?

Option	Responses	Percentage
Windows	6	85.7%
MacOS	1	14.3%

B.2.2 Task 1 – Direct steering of the output

How intuitive did you find the painting editor?

Option	Responses	Percentage
1 (Very contrived)	0	0%
2	1	9.1%
3	4	36.4%
4	4	36.4%
5 (Perfectly intuitive)	2	18.2%

Would you like to elaborate your given score? E.g., how you'd like to see it improved.

Responses

One of my issues with it is that I cannot easily delete specific things. I know the check pointing system is supposed to help with that, but the way a user (like me) edits things is usually not linear in time, so using checkpoints often also deletes things you wanted to keep, and if you ever forget to set one you're screwed. I would like to be able to just delete a building that I don't like somewhere, or adjust its size somehow. The other problem I have with this particular example is that I feel like WFC adds very little. Having an editor where you could put down some houses/shops-/building tiles by dragging rectangles is probably more intuitive as it doesn't have the distracting user interaction of having to finish a particular zone (e.g., red) with a black tile to indicate the border. My final problem might also be related to this particular example, but I feel like I cannot trust the algorithm to finish the city in an orderly way, so I just do most of it by painting it myself.
When I drag my computer mouse it doesn't draw a continuous line.
I wasn't expecting the patterns to extend out upon placing another pixel at first, but I quickly understood that it was in order to stick to the original image's patterns.
I feel it's the best way to steer the generation in a certain way. The screen shake when not being able to paint on a tile is a little annoying when you manually paint a lot. I was also not able to export the final image if I'm only using the painting editor.
Very straight forward, just select the colour you want to paint, and place some splotches and let the generator do its work
It might be more intuitive if a preview was shown of the tiles that would be added while hovering the mouse over the field, before clicking.
Sometimes it generates a chunk too big to be controlled, but it is pleasant anyway, it's like having our hand taken and just partially choosing where to go. It would be nice for the drawing part to maintain the click.
It requires a lot of clicks, something like having two colours for each mouse button would be nice I think
an eraser function would be nice!
I wasn't aware that it only worked on untouched areas; I assumed that I was able to replace what was already generated with a new colour.Also, when a lot of parts were already painted, there were still a few transparent pixels I wasn't able to colour. I assume this has to do with the auto generation, but it was a trial and error to find out which pixels were still available for me to paint.

B.2.3 Task 2 – Tweaking pattern occurrence

Is it immediately apparent what effect these values next to the patterns have on the output? (Try setting very large values) If not, why?

Responses
Yes.
It's immediately apparent

Yes – I could see the more I increased the chance of having grass (green), the more my image had grass, and the less corner pieces I had, the more straight lines there were in my image.
It was actually not apparent until I read this question. :)
Sometimes the value is more subtle, but with very high values it is visible. I've set some values to 0, but they still appeared in the image (all the grey building blocks were 0, but they were still in the image)
Yes
Yes, the frequency in which a pattern appears.
Yes, but it seems that some patterns rely on others. So I set one at 200 and one at 0.1, but I saw a ratio of like 1/10 and not 1/2000, but I believe it is working perfectly!
Yes, they work as described in the video.
not really, big values seem to be breaking the app
I guessed what effect it would have, I just had to put it to test because I wasn't immediately sure if I was right. Turns out I was right.

Was it easy to understand how to tweak a value?

Option	Responses	Percentage
Yes	11	100%
No	0	0%

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, . . .)

Responses
By putting the occurrence of doors to 0, I managed to generate buildings without doors for the most part, but there still was one building with a door. . . I noticed that there are tiles that are optional (e.g., doors, windows, other obstacles), and there are tiles that are really mandatory for the overall structure (the walls, plains). In my opinion it should be possible to put the probability of such "optional" tiles to 0, instead of just approximately zero. Maybe there is a way to detect when tiles are not "core" to the tileset but rather decoration? Or even better, leave that to the user and just detect which tiles can substitute each other in all cases, and then allow all of these but one to be set to fully 0 (could be nice for the UI as well, to group tiles that way).
It would be nice to type in a number instead of holding down an Increase button.
I have no issues/improvements for the pattern occurrence.
It is basically just trial and error, just tweak the settings until you get what you want
How to tweak a value is easy to understand but less easy to do. It would be easier if the value could be typed instead of using the arrow buttons.
If we could have a preview of which patterns induct other patterns

Being able to set values by typing. Also, the generation got stuck at a certain point and just kept trying to resolve itself. In this instance I'd given the road tiles a very low probability of 0.1. I'd advise creating some sort of maximum attempt of retries and then just letting the generation fail or retry completely a new with a new seed.
At some point the app wasn't able to generate an image and got stuck, and at some point it crashed. No exactly sure what caused it though
I quickly understood that fast-clicking did bigger jumps, and slow-clicking did smaller jumps, even decimals. However, I think it could be useful if I could just type in a number instead of having to click the arrow. It can be clunky to get to a specific number.

B.2.4 Sandbox

Would you like to perform the third task?

Option	Responses	Percentage
Yes	5	45.5%
No	6	54.5%

Why did you create the shared images? What was the incentive to create these?

Responses
Maze – I wanted to experiment with the Seamless Output in order to see how that could affect the layout of an image, and it worked particularly well for the maze in order to make it navigable. Town – With my role as a Designer, planning out towns is often quite necessary, so I wanted to experiment with how the tool compares to my manual planning out. The shapes seemed very boxy, however often towns are like this, so it was interesting to see how similar the two were.
I found the circle nice with its shadow effect, So I wanted to try to make an infinite shape.
Mostly just clicking for fun
I was interested in playing around with the paint tool and

Was it logical that Input Wrapping, Seamless Output & Pattern Size are located under the "Advanced features", if not, why?

Responses
Yes – they don't seem like necessary tools for simpler designs, so it makes sense that they were under "Advances features".

Seamless output (to me) seems like a very common feature that is not necessarily advanced.
Yes it is good.
I would'n consider them to need an additional button to unlock. They seem rather self-explanatory
I wouldn't mind if they would just be available without having to toggle the option. They're easy to ignore on the screen if you don't need them, but it's useful to have them there when you need it, without having to toggle the option.

How intuitive did you find the placing & loading of markers, and the timeline associated?

Option	Responses	Percentage
1 (Very contrived)	0	0%
2	0	0%
3	0	0%
4	3	60%
5 (Perfectly intuitive)	2	40%

Would you like to elaborate your given score? E.g., how you'd like to see it improved. And any issues you encountered?

Responses
At first, I didn't understand the usage of markers, however they are simple to place and use, and the timeline was great for watching the pattern form slowly in order to see whereabouts it decided to expand, and also for manually painting where I wanted to edit.
I managed to get the algorithm stuck a couple of times. In the smart mode sandbox, Textures category with the Wall input image and pattern size set to 3, it got stuck a little over halfway. In the Worlds Side-View with the Flower's input image, the algorithm got often stuck and even generated an index out of range exception once.
The only problem I got was I generated another image and couldn't go back to my previous marker.
Placing and loading markers is easy
I assumed I was able to go back/forward a few steps by sliding the teardrop shaped icon under the timeline, but I am not able to do so. Instead, I have to click the "previous" and "next" buttons.

B.2.5 General Questions – After the user test

After watching the introductory video, was anything unclear, that could not be solved by using the info button?

Not asked during the second user test.

Do you have any other feedback on the current interaction methods?

Responses
UI Tip: make the weight editor use pre-defined weights (like a 5-point scale), because most of the time small differences in the numbers do not matter for the result. This will look nicer, and the user's thought process will be more like "I need a lot of this, and not so much of that" instead of the user setting numbers to arbitrary values
Perhaps an option to be able to upload your own patterns which you can use to generate images.
I like how flexible this program is becoming!
There is no Undo method when you accidentally click wrong, or use wrong tile in paint mode, would be nice to have. Nonetheless, very nice and powerful tool!
Maybe having a display of all the tile in the painting.
Being able to paint helps a lot, it would be nice if you could press and hold the brush like in paint for example

B.3 Third User Test

B.3.1 General Questions – Prior to the user test

What is your relation to content generation?

Option	Responses	Percentage
Programmer	11	64.7%
Artist	4	23.5%
Designer	2	11.8%
Other…	0	0%

How many years of experience do you have in this/these field(s)?

Option	Responses	Percentage
<1	2	11.8%
1-2	2	11.8%
2-4	3	17.6%
4-6	5	29.4%
6-10	4	23.5%
10-15	1	5.9%
15+	0	0%

How have you been contacted to participate in this project?

Option	Responses	Percentage
Employee Game Development Company	3	17.6%
Online platforms such as Discord	10	58.8%
Friends or Family	3	17.6%

Which platform will you be using?

Option	Responses	Percentage
Windows	14	82.4%
MacOS	3	17.6%

B.3.2 Task 1 – Tweaking static pattern occurrence

Task Description: On the bottom left, enable "Advanced Pattern Toggling". Then, through the use of the arrows presented with each pattern, increase the frequency value of the green grass tile slightly (e.g., 5), and generate a complete output image. Then, significantly increase the value of the grass tile (e.g., 50), and again regenerate the output.

How intuitive did you find the tweaking of the values?

Option	Responses	Percentage
1 (Very contrived)	0	0%
2	2	11.8%
3	4	23.5%
4	7	41.2%
5 (Perfectly intuitive)	4	23.5%

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, . . .)

Responses
Really nice updates. Functions are straight forward. I could find the function to clear the image to start over with different settings.
I really liked it, one thing I find that could be improved it simply the wording of the hover tip on the "Increase Jump amount" button. I think "Increase Increment" would probably be better.
I really appreciated being able to define the amount of each tile that I want. This allows me to dictate the outcome in a far more precise way.
overall very good, really good improvement, only sugguestion i would have is maybe making the red/green colour change on the "toggle if this tile may rotate" button more visable, as its hard to tell the difference with the colour so desaturated. other than that very intuitive.
You click the right/left arrow to increase/decrease the value you can increase-/degrease the actual number with. I was confused the first time because the number wasnt changing, only when pressed the upper arrow the number of the green square changed. This system works, but it takes two steps where you expect one step
I don't think individual increments are interesting at all for the occurrences, maybe it's better to do it with toggle symbols where you just pick one preset (e.g.–, -, +, ++, +++, which could be set at 0.1, 0.5, 10, 100, 250), or a slider. That way you capture the idea of "I want a lot of this, and a little bit less of that" much better. You can then use the crossing out as with the other mode for excluding tiles, which is more intuitive.
Reverting values back to 1 is also annoying now, because you have to decrement the value first with whatever increment/decrement you had set, and then reset that, and put it back to 1. It's a lot of clicking for nothing.
It was not clear from simply looking at the arrows what they would do but after clicking once, it was.
Pretty simple clicking the buttons to increase/decrease. Holding them to increase by large amounts was also pretty easy to realize.
There are four identical arrows in different orientations to adjust properties of the value. Use plus and minus for the numeric changes.
The labels "increase value"/"decrease value" does not communicate what it actually does until after you run the generation. Without the instructions I wouldn't have known beforehand that "value" has something to do with the weighting-/dominance/area of this type of tile.
Well, I liked your tool a great deal, it helped a lot in my graphic design needs though (even if you did not mean for it to be used that way, haha) and I find the UI really great. This is something that could even be converted into a million dollar idea.
I found ease in manipulating the values and noting great change in the frequency of the tiles, especially when increasing the value significantly. An improvement may be the ability to add in the value manually, to save time, instead of scrolling to your desired value.
Using up/down and left/right doesn't seem intuitive at all, and at first I thought the value between the arrows was the final, not the step value

I think it would be nice to be able to type a value instead of just clicking "+" for a pretty long time
At first I thought that the left/right jump amount would directly change the number, instead of just changing the order of magnitude of the change when pressing the up/down button

Was the appearance of the output image easily relatable to your changes of the frequency value? Why (not)?

Responses
Yes
Yes, when i made changes in the frequency you can see difference.
Yes definitely.
Yes, it was clear the green tile was way more frequent once I bumped up the value.
Yes, the increased tiles and movement of them were relative to the resulting image perfectly.
yes very much, you could see your changes clearly
Yes, the higher the number of green, the more green was in the image
Yes, it made sense.
Yes, because there was more green on the image at a value of 50 compared to 5.
The rendered image vanishes when the window loses focus, which is confusing when flipping between these instructions and the application. Also, the 'run' icon doesn't restart the process after a prior run, instead I used the revert to save point to go back. The progress bar tear-drop looks like it could be grabbed and scrubbed backwards but cannot. The value changes appeared to have no effect unless the process was restarted.
It was. But with the caveat that it was only clear to me because of the instructions in this form what changing of this value does.
Yes, it was.
Yes, though having a 'weight' column or something to a similar effect would have made it even clearer
Yes, I saw more grass when I selected more grass
The appearance was very relatable, once I realized how to actually set the value

B.3.3 Task 2 – Tweaking dynamic pattern occurrence

Task Description: To the top-right of this pattern section, press the button to reset the patterns to default. Now click on the "1" to the bottom left of the green grass tile. This new window helps you set these frequency values by painting on the output space: yellow means high values, blue means low values. Try to draw a mapping that has areas of very high values, and areas where the value is (almost)

0. Once you have drawn the mapping of the green grass tile, return to the previous screen and re-generate the output to use this mapping.

How intuitive did you find the ability to create a mapping for pattern occurrence values?

Option	Responses	Percentage
1 (Very contrived)	1	5.9%
2	0	0%
3	5	29.4%
4	6	35.3%
5 (Perfectly intuitive)	5	29.4%

Would you like to elaborate your given score? E.g., how you'd like to see it improved.

Responses
Again, i created a image before and couldn't reset it. Besides that, really nice feature
I really liked this as well, I would probably add a smaller brush size + maybe make the brush size adjuster a slider rather than a dropdown menu.
It's a very simple system to work with, I immediately got the hang of it.
This is an extremely helpful way to define how heavy or light the occurrence is. Utilizing color is a naturally intuitive approach, and I appreciated the ability to overlap and blend different levels of occurrence in one area.
very easy to use, great layout,. only worry would be finding how to enable the mapping process as clicking on the number isnt that obvious. maybe having a button on top near the reset tile values button would be easier and more accessible
I could not find this option, so I could not complete the task
There was one issue, but it's probably just a bug: the weight painting only works once you've set the occurrence of a tile via the increments beforehand; if you start painting occurrences right away, it won't actually do anything.
In addition, it's a bit tricky to specify weights for tiles that have the same function (e.g. a wall) but have variations (e.g. wall over water, wall over land, wall over road). Ideally you'd just want to say that you do not want a wall somewhere, but with this you have to specify that same intent 3 times, one for each of the variants.
Also, it seems that having more and more painted weights causes more generation failures.
One issue I encountered was that I couldn't press CTRL+Z to undo my last brush stroke. Otherwise, it was pretty intuitive.
The 'go back to main screen' looks like "undo/discard changes". The naming of some buttons in the interface does not match the references in these instructions, or is ambiguous (press the button in the top right... there are two buttons!)
I found no fault. The occurrence of tiles based on colour value made much sense to me.

Overall quite usable
The mapping interface itself was quite intuitive, but if it wasn't written down, I wouldn't have expected that the "1" is clickable.
The mapping disables the occurrence which means you have no further control, for example I tried to create a green field in the middle but couldn't get rid of the structures. Also the drawing controls were a bit clunky, have a slider for thickness and have options for aliasing/gradients (a bit like weight painting in 3d modellers)
There's a significant different between zero and one, and the painting interface doesn't make it easy to quickly toggle between them.
I'd also like a way to easily increase the weighting of all of the tiles (i.e. emulate having fractional weights by making everything else proportionally more probably)

B.3.4 Task 3: Tweaking pattern symmetry

Task Description: As you might have noticed, there is a new button to the right of the tiles, a green button with arrows. As well as some buttons to the right of these coloured buttons. Try clicking on one or more of these buttons and regenerate the entire output to see what they do.

How intuitive did you find the ability to toggle the use of tile symmetries in the output?

Option	Responses	Percentage
1 (Very contrived)	0	0%
2	2	11.8%
3	3	17.6%
4	6	35.5%
5 (Perfectly intuitive)	6	35.5%

How intuitive did you find the ability to set a tile's orientation, when symmetries are turned off?

Option	Responses	Percentage
1 (Very contrived)	0	0%
2	3	17.6%
3	4	23.5%
4	5	29.4%
5 (Perfectly intuitive)	5	29.4%

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, ...)

Responses
I am not sure what i did with the buttons. It looks cool, but i think i needed more guidence in what that function does to give it a good rating.
I didn't really understand what it was doing, the buttons were a bit strange and seemed to contradict themselves "make the tile invariant to rotation" "rotate the tile when locked", and it all seemed rather hard to understand to me. Plus i encountered an infinite loop
It takes a very short amount of experimenting to understand what the options do, it's pretty intuitive.
Being able to dictate the orientation is another level to the precision abilities with this program. Superb!
all fairly simple to use, love the addition
It does what is says and it is visible in the image
From the button symbols I would not be able to tell what they do. Maybe it's better to have a full rotation symbol (arrow that goes almost 360 deg) with a lock symbol on it, and then have the other button overlayed ontop of the tile, so it's clear that it rotates the tile.
In addition, this feature does not work for tiles that have only 2 orientations, e.g. roads. I wanted to try to make only vertical bridges, but it did not allow me.
Other than that, it's nice to see the effect on the output, I used it for road turns, and indeed you get a pretty interesting pattern once you lock tiles.
The text when you hover over the buttons could tell whether the option is toggled on or is currently off.
The buttons for toggles appear greyed out to some degree even when they are active.
The buttons to set rotation and symmetries do not communicate clearly what state rotations and symmetry settings are currently in. I do not know what green or pink means. Nor would person with colourblindness.
I found a fault with the rotation button; When rotating certain tiles and then generating an image, the image gets to the end of its step , but won't complete. The final result has blank spaces with no tiles present, as shown in my exported image. Other than that, all good.
I didn't really realize I could set the tile's orientation. I also found it hard to see the effect, as some tiles existed already in different rotations.
The toggles are easy to find, however I had a lot of trouble generating solutions when using them. Also you can't choose a subset of the rotations which might have issues (for example you want a u turn)
It was a bit confusing that you couldn't toggle it off without regenerating the whole scene, because it felt like the weights didn't care what was already generated, but the tile flipping does care about the past for some reason.

B.3.5 Task 4: Pattern Exclusion

Task Description: Upon toggling the Advanced Pattern Settings in the bottom left of the application, you are now able to click on patterns that you want to exclude from the output. Please disable ALL patterns with one or less "light" sand pixel, and generate a complete output.

How intuitive did you find the ability to exclude patterns from the output?

Option	Responses	Percentage
1 (Very contrived)	0	0%
2	0	0%
3	0	0%
4	5	29.4%
5 (Perfectly intuitive)	12	70.6%

Did you encounter infinite generation or inability to generate output after excluding one or more patterns? If you did, did you understand why this happened?

Responses
No i did not encounter this. Although i would understand, thanks to the video, why this would happen.
I did not.
No, I didn't encounter any infinite generations.
No, but if it had I do understand it would have been due to too many exclusions. This was perfectly explained in the Explanation Video!
didnt encounter
Did not happen
Honestly not, it is very unclear why such a thing may happen or which exact tile is causing it, other than knowing that you have disabled too many tiles. Maybe this is more intuitive for tilesets that have more familiar coarse-grained structures.
I did not experience infinite generation or the inability to generate output in this case, even after many tries. I did in one of the previous tasks, when locking the rotation of a tile. It was clear why it happened.
Nope i didnt encounter this i think
N/A
It did not happen when I remove patterns with 1 or 0 bright pixels.
Yes, I encountered such situations. I think it is because of incompatibility of the tiles, which are rare and are fixed by a regeneration of image.
No, I'm unsure of what you mean. I encountered no issues.
I didn't.
The selection is quite intuitive. I really like it. It was hard to predict what it would do to the final image, but I guess that's kind of trial & error

I had to disable a lot to trigger it for this one, my theorie why sand isn't prone to failure is that there are a lot of patterns that can go together because there are only 2 color values and the pattern size is quite small
No, I didn't end up excluding vital patterns, I guess. The weight painting was causing more failures than the smart tile exclusion.

B.3.6 Sandbox (optional)

Task Description: This last task allows you to explore all functionalities of the tool in a sandbox. For this reason, this task can take as much or little time as you desire.

Would you like to perform the third task?

Option	Responses	Percentage
Yes	6	35.5%
No	11	64.7%

Please head to either of the tabs, and play around with the miWFC application

Why did you create the shared images? What was the incentive to create these?

Responses
I wanted to test the program to its highest abilities. I played around with various outcomes, both with highly saturated outcomes and then with high levels of occurrence, and back to lowest. These were some of the outcomes, with one being an infinite outcome.
Simply exploring the (new) features.
Knots and similar WFC examples always generate output that looks very pleasing to my eyes.
Well, I used this tool for generating game maps electronically readable, efficiently. Thank you for this tool. I will soon donate to you for this tool if you are open for it.
Seeing if I could make something interesting, I guess.
Also, I somehow managed to make the whole thing freeze when I removed the central flower tile.

B.3.7 General Questions – After the user test

After watching the introductory video, was anything unclear, that could not be solved by using the info button?

Not asked during the third user test.

Do you have any other feedback on the current interaction methods?

Responses
Nice updates. Looking forward to the next one!
I see lots of improvements in this version and I enjoy working with the program.
Seeing the improvements from the last test phase implemented so well into this new version is wonderful. I am very impressed with the level of detail to customization options that have been added. Very great job, I look forward to the next testing!
Nice tool, has much potential. I am curios to see what can be created with less pixelated art
I think the weight painting is very cool, but I do also feel that the editor is starting to suffer from having too many windows/views, maybe there's a way to consolidate everything into a single intuitive view.
Pretty cool software!
I would really like Painting Mode to be improved. As it stands right now it seems I can only draw one pixel at the time. It would be useful if I could draw a bigger area using my mouse. Also sometimes it does not respond and the application does not communicate why.
No
I really like the new extension. I think they help a lot with generating interesting tile patterns. I did find a bug: when I was "reversing" the generation (i.e. with the « button), with a pretty big step size, the whole application stopped responding to any input after I got to the start. I think I reversed to before the generation started or something.
I'd like to be able to toggle a pattern off by selecting it in the generated output. I couldn't figure out how to regenerate an image with fewer than two clicks. Sometimes I felt like I wanted to see a bunch of different results but I had to reset it and then hit play to see anything new.

B.4 Fourth User Test

B.4.1 General Questions – Prior to the user test

What is your relation to content generation?

Option	Responses	Percentage
Programmer	7	63.6%
Artist	1	9.1%
Designer	3	27.3%
Other…	0	0%

How many years of experience do you have in this/these field(s)?

Option	Responses	Percentage
<1	1	9.1%
1-2	1	9.1%
2-4	3	27.3%
4-6	2	18.2%
6-10	3	27.3%
10-15	1	9.1%
15+	0	0%

How have you been contacted to participate in this project?

Option	Responses	Percentage
Employee Game Development Company	1	9.1%
Online platforms such as Discord	6	54.5%
Friends or Family	4	36.4%

Which platform will you be using?

Option	Responses	Percentage
Windows	10	90.9%
MacOS	1	9.1%

B.4.2 Task 1 – Templating

How intuitive did you find the creation and placement of templates?

Option	Responses	Percentage
1	0	0%
2	1	9.1%
3	2	18.2%
4	7	63.6%
5	1	9.1%

Would you like to elaborate on your given score? (e.g., any issues you encountered, how you'd like to see it improved, . . .)

Responses
The shaking on invalid placement is slightly too intense, it would be nice if the dialog that pops up could become a notification instead. In addition, it's not always immediately clear why some placement is invalid. Other than that, pretty straightforward.
Having to select individual tiles was fairly tedious, re-using brush functionality or having drags create boxes would have allowed for quicker selection. The process described seemed more arduous then it should be, and multiple times I accidentally deleted the template or cleared the selection rather then the image. Using the 'generate instantly' on the 'steps to take' scale should, in my opinion, be triggered by the play button since it is otherwise very tedious to clear the entire output.
The idea is really nice. But like you said in the video, the output was not was i intended. The feature is cool.
Creating templates felt a bit hard to discover, using templates was relatively straightforward.
I wanted to manually create the "perfect castle", copy-paste it with templates and autogenerate the terrain around them. First manually placing the tiles, I found it difficult to tell which tiles (in the dropdown) are actually possible to be placed at the current cursor position. With a large tileset, I feel it would make more sense to be able to "select" an empty tile from the painting grid and then select one of the possible options from the Available Tiles list on the right - instead of having to remember which ones I can select from the dropdown. Creating and placing templates was not a problem. Though, having to click each pixel is a bit annoying for creating larger templates, it would be nice to be able to click-and-drag to select tiles or to use the brush tool. However, a much bigger annoyance was some features being disabled with a partially generated image - I placed my templates where I wanted them, but then it was not possible to change the tile weights (in advanced pattern settings). So if I didn't like the distribution of pixels generated around my templates, I would have to reset the whole image to change the weights and manually place the templates again... Running with "Instantly generate output" is also not possible. It also seems quite limiting that the weights cannot be set higher than 250 - is a maximum value necessary for the program to work? If not, then there should be no limit, especially in a creative/design program, where an artist will inevitably want to use the tool in a way you could not possibly have foreseen.

Making the template is easy, even though it is a bit hidden in the paint menu. In general the more you can show the user in once screen the better. Also templating is easy, placing them is okay, though the dropdown could be replaced with some sort of inspector to also order templates in folders etc. Templates are difficult to remove you'd expect right mouse button to erase them if you'd make a mistake. Also rotating should also be bound to a hotkey. The screen shake when placing a tile is quite clear but having a pop up every time you make a mistake isn't that nice, it'd be better if there was a small notification or something that doesn't completely break the flow
id like to see an "undo" option when selecting a template region, as its very easy to make a mistake selecting the pixels (maybe there already is one, unsure)
When creating templates, I personally had trouble seeing which pixels I was selecting to create them. High contrasting colours in the pixel selection would make this significantly easier I think.
quick generation, clear tutorial videos, intuitive UI; too many separated islands in the generated map,

Do you find this option a useful addition?

Responses
Yes, with this addition you have more control of the output
Definitely, but I did not use it because I had already generated some tiles around the area I wanted to template. I did not realize at the time that I could have actually made use of it, if I deleted all of the generated tiles with the brush tool, or set a marker to revert to (maybe have an option that is enabled by default to set one automatically when the painting editor is closed?).
Not really, it's more work to deselect tiles since the desired template is usually way smaller than the full image.
Not particularly, I would find the aforementioned tools more useful.
yes a lot, very good addition
Yes, and I appreciate the detail put into options for customization
This could be useful, yes.
YES
yes

B.4.3 Task 2 – Custom Input Images

How intuitive did you find the ability to create and use your own input images?

Option	Responses	Percentage
1	1	9.1%
2	0	0%
3	1	9.1%
4	6	54.5%
5	3	27.3%

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, . . .)

Responses
The pattern size and input wrapping options are actually essential information, it's weird that they are hidden under the advanced options toggle. Instead, you should be prompted for this when you load a custom image, it should somehow be saved with the image.
It is intuitive. But it was hard to import an image. Keep getting the wrong dimensions, although i adepted my dimensions to the export result.
The program would sometimes break on resetting the output (size >100) - with the loading animation staying on the screen and the output saying "no result could be generated". Only a restart seems to fix this. Importing a generated 128x128 image took multiple minutes and the brush tool did not work with it.
Adding files in folders isn't very intuitive. Especially if the user then has to manually refresh the dropdown list by exiting and then again entering custom mode
overall very easy to use, simple and effective, only problem i encountered was when i tried generating an image, it always seemed to be 1 set colour, unless i tweaked the settings (removed some tiles through advanced pattern settings or messed around with imput wrapping or pattern size) then it changede to giving me just striped images, and then at last it managed to give me different images each time. unsure if this is something im doing wrong or a bug in the code, but i thought i should mention it either way
It's quite logical once you know what to do.
The elements provided proved to be very simple to use.

What did you learn from creating and using your own input images, was it hard to create input images that caused the output to appear as you intended?

Responses
There are quite a lot of intents that can be put into input images that WFC will fail to capture, things such as symmetry, specific shapes that need to be preserved, etc. However, I noticed that WFC is very good at preserving notions of "inside" and "outside", which is quite interesting and useful

It became clear that more complex shapes that do not generate neat 3x3/2x2 'tiles' will have unclear effects, and that colours may be used ambiguously.

No, was not hard. Output was not as i intended.

I had a pretty good idea what the results would be, and the output matched my expectations.

Using the painting and template tools to fill in regions worked reasonably for forming the output, although painting one tile at a time is a bit slow and selecting parts of a generated image for templates was still a bit annoying, because selecting a single tile at a time is still faster than reverting to markers for large outputs. (Depending on the color, it's sometimes also not clear which tiles are actually selected for the template - showing this by only changing lightness does not seem like the best option)
The Extracted Tiles view doesn't seem very useful without being tied to the input image (e.g. showing from where on the input image the tile was taken from or just being able to enable/disable tiles by clicking on the full image).

It was very hard, WFC with a kernel size of 3 is quite limited in its ability to understand and learn patterns

nope, very easy to use, really intuitive and u can see it was clearly thought out, besides the one error discribed above. well done

This was surprisingly easy and accurate, the shadows were even correctly places most of the time. I'm only missing the red stripes.

Unless I knew completely what each enabled tile did, it obviously involved a lot of experimentation to see how each tile is connected with each other.

The tools seems to be very intuitive and powerfull. The aspect ratio between patterns and background are sound. The generated images do get some prior knowledges from the original one.

B.4.4 Task 3 – Post Processing

Please elaborate which colour of items represent what type of item or object, and why you decided to add it, including its frequency, location, and possible linked items.

Responses

Green ones are guards, aqua ones are objectives. The guards have to protect objectives, though I made the objective dependent on the guard through linking. The objective spawns close to a guard, but not too close (3-6). None of them are allowed to spawn in the big room.

I didn't notice I could choose colour, and chose random parameters since I didn't see how I could make them meaningful in the abstract image.

I left the frequency as default to see what would happen. Location and items were completely random to play around with it.

I tried to add a bridge first, but there wasn't really a way to set one up with the item post processing (could probably do it with adding the patterns, but that'd be a different problem).

Brown - chest - 6x on blue-gray tiles * Linked: Yellow - key - distance 1...20
Red: spawn. Green: ammo. Blue: health
the yellow item represented gold, and i decided to add it to spawn 1 on the whole image on any black square (a house), not linked to anything. the red items represented fire, and i added 25 of them and allowed them to spawn on any square of the image (black , green or yellow, also not linked to any item
I simply randomized my items to see how the algorithm would react
Healing Items are marked as green with a relatively high frequency in the world to help in a theoretical game setting. Boss Encounter happens in a generally secluded area by itself to create a challenge for a player. Hidden Items reward players for thinking quite literally outside the box. Normal Ammo linked with Rocket Ammo, same reasoning with Healing Items but with the extra step of giving the player an extra challenge to locate the linked item for a maximum advantage.
Item 1 (in green): location of current player/user, 1 item randomly generated on the path Item 2 (in blue): obstacles placed to hinder the player, 10-20 obstacle generated randomly on the path
It's two different types of flowers, I wanted some in a more central area and others more to the side. The frequencies are low so they match with the other flowers.

How intuitive did you find the ability to add a new layer of custom items or objects into the world?

Option	Responses	Percentage
1	0	0%
2	2	18.2%
3	2	18.2%
4	5	45.5%
5	2	18.2%

Would you like to elaborate on your given score? (e.g., how you'd like to see it improved, any issues you encountered, ...)

Responses
There seems to be no way to "retry" just a single item type or instance nor to manually position some of them.
The 3 exported items were confusing for a second. But not a problem
While it seems to work well enough, it is fairly unclear how useful this feature might be with its limited controlability and the abundant repetition within spaces. Being able to select specific tiles for items to be spawned on might make more sense.

It was unclear how the linked items worked. Also there is quite little control over where items are placed. For example I didn't want ammo and health to spawn near the player but I couldn't really control both. I'd also liked it if I could place ammo in certain locations e.g. corners but that also is not really controlled that well, or I'd have to paint them. Which beats the purpose of having it done procedurally.

It's just hard to fully capture user intent through this. I wanted the guards to spawn in the main/big room, but the dependent items in the smaller rooms. In additions, I would've liked having multiple dependent objects per main object. That way you can have guards that protect multiple objectives (maybe it would even be interesting to make this linking very generic/bidirectional, so you can vary the number of both dependent things as you like, e.g. 2 guards where the objectives have to spawn close to either of them or all of them).
Also, sometimes the WFC tileset simply doesn't offer enough information for you to make a meaningful choice besides "pls don't spawn on top of a wall", basically you just choose what is regarded as floor, which could also be an automatic process. Even with a tileset where WFC does generate points of interest in terms of tiles, this very much defeats the purpose of doing the post-processing, because all you then do is just mark tiles already generated by WFC for containing specific items. It could be more interesting if you could somehow specify neighbourhoods instead of selecting a single tile (e.g., this thing should always spawn in corners).

very easy to use, simple to set anything you want. i love the way the controls are set out, how you can easily reroll the positions of everything, and edit the items after already creating them. imo perfectly done, cant think of anything to change

It took a bit of puzzling to figure it out entirely, but it makes sense at the end.

Honestly, the system does what it intends to do perfectly. Doesn't hurt to have more options and settings in the future though :'D

B.4.5 Sandbox – Final Task

Why did you create the shared images? What was the incentive to create these?

Responses
I wanted to try a make like an interior castle setting and was pretty impressed with just how it was able to generate different things based on my really basic input.
It seems that with miWFC, you can create art pieces and mimic certain styles of art beside world generation.
I'm gruesome
I am an endless piece of string, by myself. I can handle stress and tension, but under pressure I will mention, that I have many, many uses, can tie up papers, shoes and pots, and furthermore what I produce is, many kinds of handy knots. So when you see me, know one thing: I am an endless piece of string! (I just made that up because I liked the circle input becoming a string in output.)
I was wondering what would happen if i upped one or two of the same items. In this case water and how de land and castle items would act. Endresult was interesting!

Because I liked how it looked.
Minimalism
i created the image because i think it reflects me and my personality well. im a rather detailed person, some would say perfectionist, and i like to focus on the details of art, as i find that is always my strong point. i made this image using the font category, and collaged the multiple different images in a photo editing app, rotating some of them and making them fit together.
Once I get correct dimensions I want to come back and generate a map for a city connected with rivers, but no roads.
I am very into music and play an instrument myself. The green represent the plants on my piano, and yellow just matched it well. The red background with yellow decorations refer to Turkish carpets, they're pretty and part of my culture. The colour combination of yellow and red also suits me.

B.4.6 General Questions – After the user test

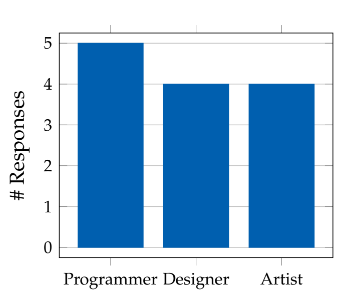
After watching the introductory video, was anything unclear, that could not be solved by using the info button?

Not asked during the fourth user test.

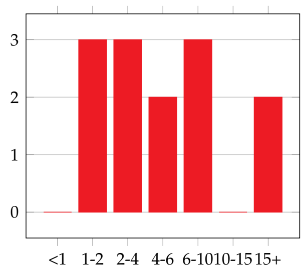
Do you have any other feedback on the current interaction methods?

Responses
Not that I can think of no.
Nice job on the probability tweaking in the simple mode, it's a lot better now!
the response to hitting play and getting a stern error message and screenshake is a bit jarring. Generating a bunch of different things to compare takes extra steps. It'd be nice to have a way to view multiple outputs from the same tile set and settings.

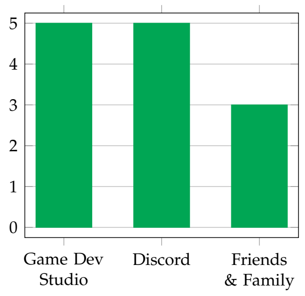
C Graphs



(a) Relation to content generation



(b) Experience in years



(c) How were you contacted?

FIGURE C.1: First User Test Demographic Results

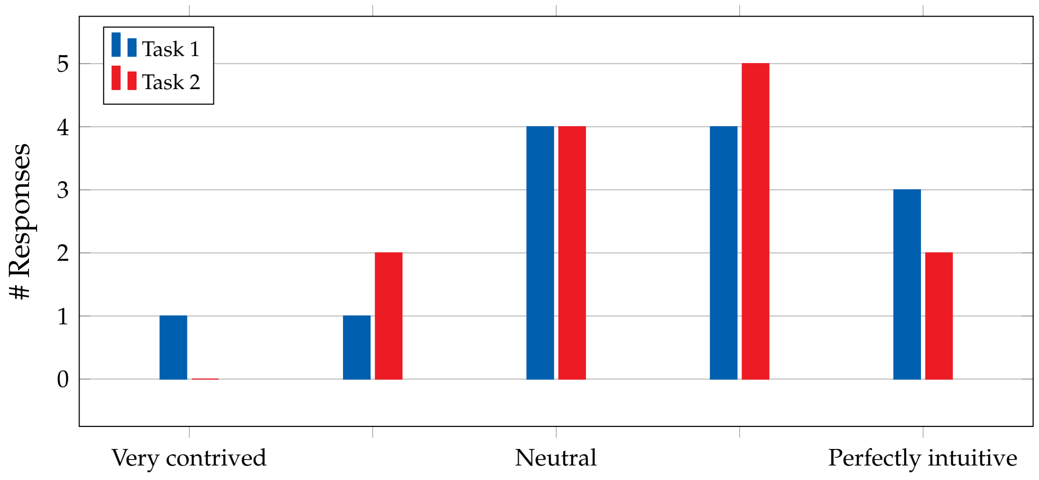
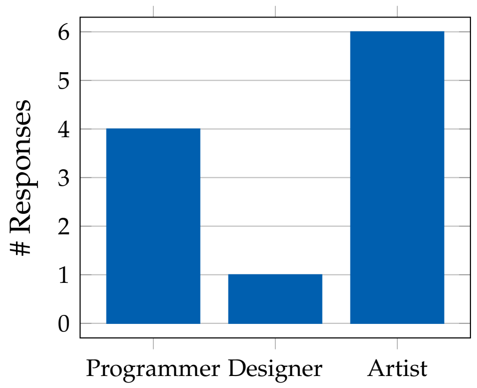
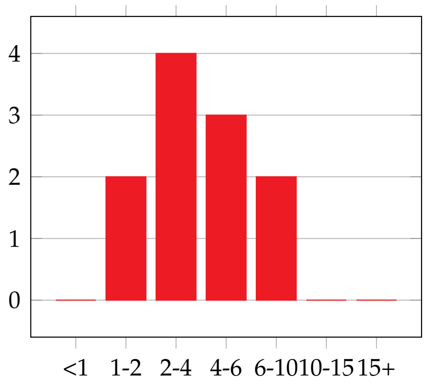


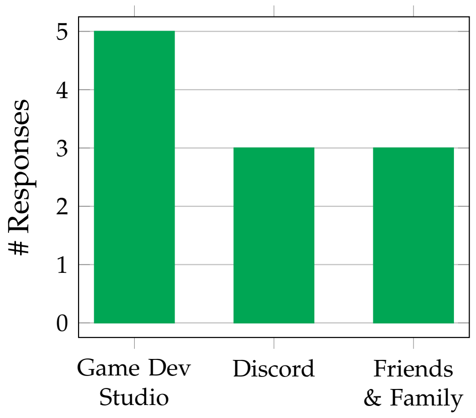
FIGURE C.2: First User Test Task Results



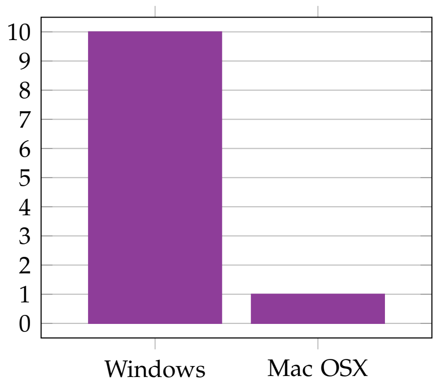
(a) Relation to content generation



(b) Experience in years

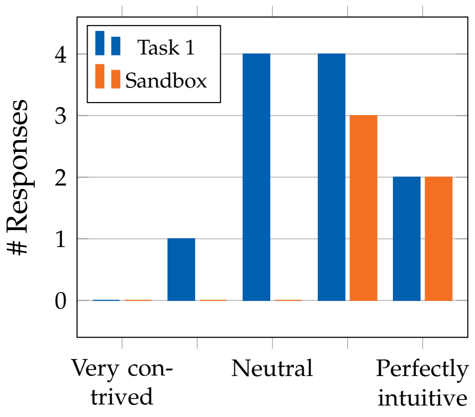


(c) How were you contacted?



(d) Operating system

FIGURE C.3: Second User Test Demographic Results



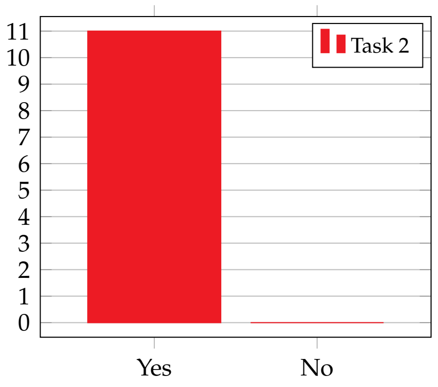
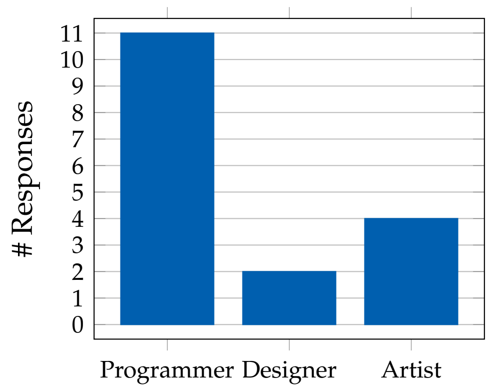
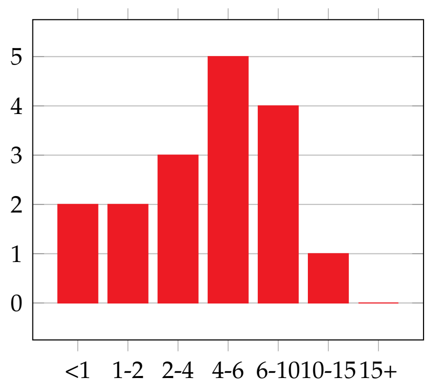


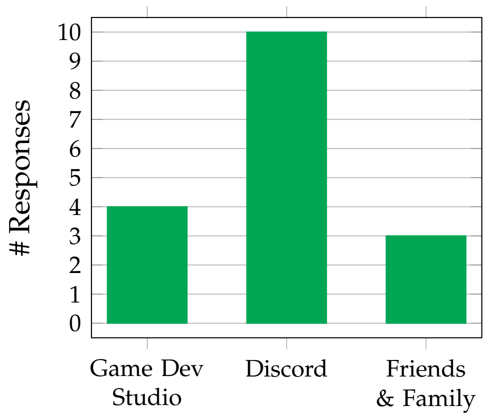
FIGURE C.4: Second User Test Task Results



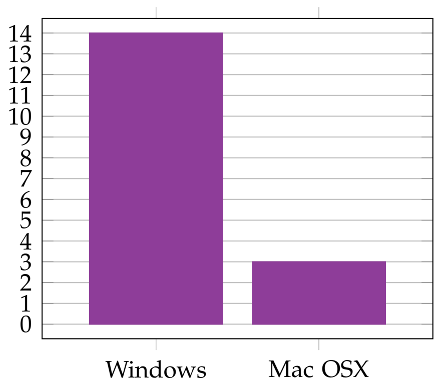
(a) Relation to content generation



(b) Experience in years



(c) How were you contacted?



(d) Operating system

FIGURE C.5: Third User Test Demographic Results

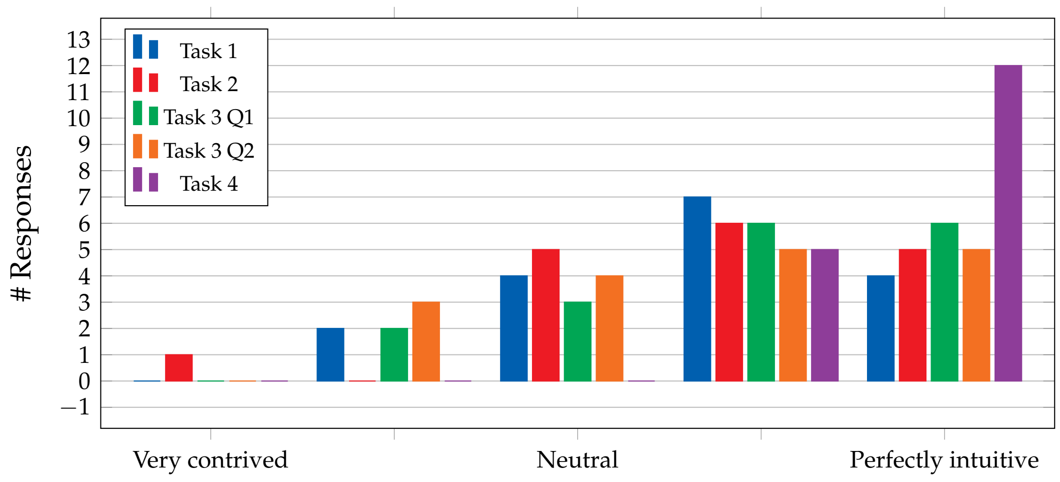
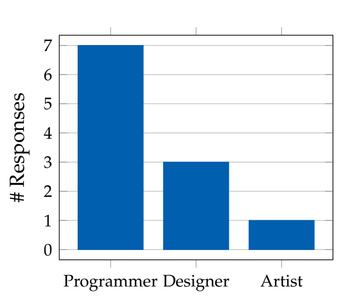
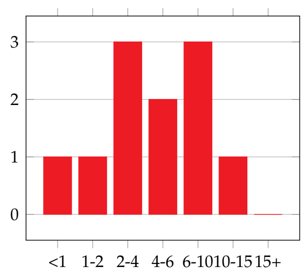


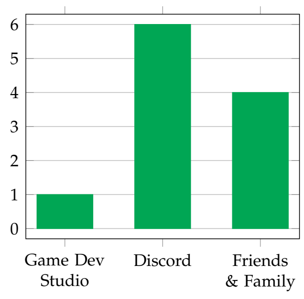
FIGURE C.6: Third User Test Task Results



(a) Relation to content generation



(b) Experience in years



(c) How were you contacted?

FIGURE C.7: Fourth User Test Demographic Results

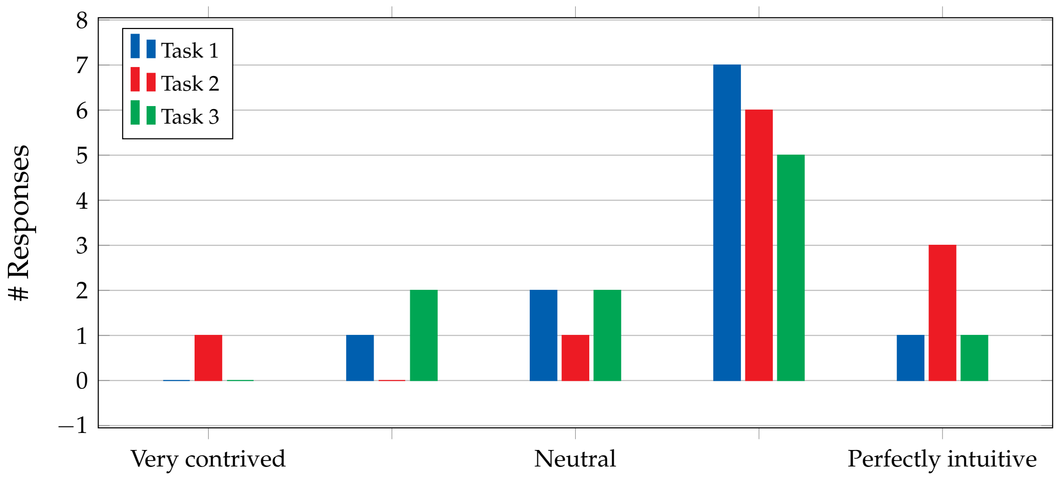


FIGURE C.8: Fourth User Test Task Results