Better Resemblance without Bigger Patterns: Making Context-sensitive Decisions in WFC

Bahar Bateni
bbateni@ucsc.edu
University of California Santa Cruz
Santa Cruz, USA

Isaac Karth
ikarth@ucsc.edu
University of California Santa Cruz
Santa Cruz, USA

Adam M. Smith
amsmith@ucsc.edu
University of California Santa Cruz
Santa Cruz, USA

ABSTRACT

Gumin's WaveFunctionCollapse (WFC) algorithm attempts to generate output designs that resemble provided input designs. While the algorithm's constraint-solving core is able to ensure that no local patterns are adjacent in the outputs that were not adjacent in the input, it does not accurately reproduce statistical properties of the input designs. Examining the algorithm's behavior at the level of pattern adjacencies, we show that there are large gaps between the statistics of the input and output designs, even when applying Gumin's search heuristic intended to influence output statistics. By offering a very small revision to this search heuristic, we show that the resemblance of outputs to inputs can be dramatically improved. Another way of improving resemblance is to increase the size of local patterns considered by WFC, but this can easily lead to a kind of overfitting that results in the outputs plagiarizing large portions of the input design. By contrast, our alternate revision increases resemblance without increasing pattern size. The simplicity of our method, requiring a very localized change to existing WFC implementations, allows it to be immediately applied to a wide range of applications.

CCS CONCEPTS

• **Computing methodologies** → **Heuristic function construction**; *Statistical relational learning*; • **Applied computing** → Computer games.

KEYWORDS

procedural content generation, constraint solving, machine learning

ACM Reference Format:
Bahar Bateni, Isaac Karth, and Adam M. Smith. 2023. Better Resemblance without Bigger Patterns: Making Context-sensitive Decisions in WFC. In *Foundations of Digital Games 2023 (FDG 2023), April 12–14, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3582437.3582441

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
FDG 2023, April 12–14, 2023, Lisbon, Portugal
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9855-8/23/04…$15.00
https://doi.org/10.1145/3582437.3582441

1 INTRODUCTION

In the literature of procedural content generation via machine learning (PCGML) [17], generators often try to generate novel output designs that resemble examples provided by human artists. WaveFunctionCollapse (WFC) is a generative algorithm that was introduced by Maxim Gumin in late 2016, immediately capturing the attention of many technical artists and academic researchers [8]. The goal of the WFC algorithm is to generate grid-structured designs (such as images, tile-based game maps, poems, etc.) while enforcing a sense of resemblance defined at the level of local pattern adjacencies. In Gumin's project documentation [6] the notion of resemblance is captured by these two conditions:

C1. The output should contain only those NxN patterns of pixels that are present in the input.
C2. Distribution of NxN patterns in the input should be similar to the distribution of NxN patterns over a sufficiently large number of outputs. In other words, probability to meet a particular pattern in the output should be close to the density of such patterns in the input.

As WFC applies constraint solving methods, the C1 condition is guaranteed to hold on any of the algorithm's outputs. However, the C2 condition was left as a weak constraint with a heuristic solution. In this paper, we show that a small change to this heuristic can dramatically improve the qualitative resemblance of outputs to inputs without increasing the size of patterns considered by the algorithm. Further, we show that the quantitative sense of resemblance suggested by Gumin's definition of C2 is not able to distinguish improvements in resemblance quality. However, our refined metrics that examine distributions at the level of adjacencies (the same granularity used for constraint solving) is able to reveal the problem and verify the effectiveness of our improvement.

Here is the central question investigated by our research: **Are there simple and easily replicable modifications to the WFC algorithm that would allow it to produce outputs that better resemble the inputs?** In this paper, we contribute:

- Resemblance metrics for understanding the relationship between source input and generated outputs (formalizing the intent of Gumin's C2 condition)
- Quantitative characterization of the effect of Gumin's original heuristic on resemblance, revealing a large gap
- A new tile decision heuristic that is simple to add to existing WFC implementations
- Empirical demonstration that our method increases resemblance without the drawbacks of using bigger patterns (loss of diversity, or increasing contradiction rate, and increasing

execution time), and that these improvements can still be seen in the case of using bigger patterns

2 RELATED WORK

The main implementation of the WFC algorithm, as introduced by Gumin [6], makes use of two heuristics: one *selects* which location in the grid to work on next, and the other *decides* what to place at that grid location. Karth and Smith [8] explored the impact of these heuristics by using alternatives for each. Both of these heuristics were shown to impact the aesthetics of the results. For the tile selection heuristic, they showed that Gumin's entropy heuristic behaved similarly to a simpler lexical heuristic, but the lexical heuristic lead to certain aesthetically undesirable artifacts. As for the tile decision heuristic, they reported that Gumin's tile frequency weighted heuristic *inconsistently* achieved the stated goal of Gumin's C2 condition, but they did not quantitatively measure this outcome.

In the broader literature on sampling from discrete spaces defined by constraints, the topic of drawing samples from a desired distribution is surprisingly deep. If the constraints are approached as hard constraints, the problem of distribution-aware sampling of satisfying assignments has a surprisingly high theoretical complexity. For SAT problems, Chakraborty et al. show that the problem can be approximately solved given a SAT-solver in some cases and a pseudo-Boolean (PB) satisfiability solver otherwise [1]. In their analysis, the difficulty of the problem is tied to the gap between the most and least weighted options. In other words, the problem is easy when most options are intended to have similar likelihoods. Later in this paper, we show that Gumin's heuristic yields increasingly inaccurate results when the scenarios considered involve many low-frequency patterns.

Other technical games researchers have considered statistical modeling as the primary concern for PCGML systems (relaxing the adjacency constraints enforced by WFC). For example, Snodgrass and Ontañon tackle this problem in a PCG tile-based map generation setting [16]. They propose fitting a multi-dimensional Markov chain model to the training data and then sampling from it with a local resampling strategy to resolve constraint violations. Because generators based on constraint solving can be used to directly enforce high level playability properties [15], we are interested in adding statistical sensitivity to an existing constraint solving method rather than attempting the reverse.

A predecessor and direct inspiration to Gumin's WFC is Merrell's Model Synthesis [11] in which novel three-dimensional figures are generated by composing reusable geometric tiles. Merrell's work builds on earlier traditions in computer graphics, such as texture synthesis, where visual resemblance is the key feature of the generator. In the setting of a 3D tiled model, Merrell defines a resemblance distance between a vertex position v and a vertex position v' to be the number of unique tiles in a cubic region around v that doesn't exist in the cubic region around v', not considering their exact position. To generate 3D models that resemble an example model E, he assign weights to the choice of each label for a vertex v based on the number of vertices v' in the example model that have a resemblance distance of less than some threshold to vertex v. This results in a kind of context-sensitive decision making process that is related but distinct from the method contributed by our paper. (In our method, a context either matches or it does not, without the need to specify how one context could be a better match than another.)

Because the tile weights have such an impact on the appearance of the results, yet another approach is to simply give the user the ability to choose these weights for themselves. Langendam and Bidarra's mixed-initiative WFC tool, called miWFC, provides the user with an interface for interactive direct manipulation of tile weights [10]. They found the base algorithm's weights to create outputs that *mostly* resemble the input, but provided the user with the weight manipulation ability to enable a much larger variety of outputs without the need to author additional examples (cf. Karth's conversational integration model [7]). While this lets the users push designs to get closer to their desired visuals, with the weights being context-independent achieving better resemblance is still difficult. Besides, we would like to get the base algorithm to do a better job before requiring user intervention.

Sturgeon [4] is a very recent WFC variation proposed by Cooper and used for level generation. This implementation involves learning design constraints from examples. These constraints include tile-to-tile adjacency as well as long-range reachability along certain paths. Additionally, frequency constraints are used to account for the number of tiles allowed in a tag or in a specific region of input and output being similar. These constraints are treated as weak constraints in the underlying constraint solver. Their work shows that there are many more characteristics to learn from input designs, but the learned patterns in Sturgeon do not fit the representation used by existing WFC implementations. In this paper, we aim to improve resemblance via interventions strictly within the skeleton of Gumin's approach. This will allow our improvements to be quickly ported to other WFC implements without revising the core algorithm.

3 RESEMBLANCE METRICS

Recall from the introduction of the paper that WFC aims to produce outputs that resemble the inputs. In this section, we define the following metrics to compare the resemblance of two designs. To make these metrics easier to understand, we will describe them in the context of comparing two-dimensional images composed of repeating pixel tiles.

(1) **Tile Frequency.** This metric is based on the C2 condition of Gumin's local similarity definition. Given an image, we can count the number of appearances for each of the tiles in the image and use this frequency to compare the observed frequency in generated images. Arguably, if two images closely resemble each other, we should expect their tile frequencies to be close, but the reverse of this should not necessarily be expected, as shown with an example in Fig. 1
(2) **Edge Frequency.** We define this metric for each pair of tiles as the number of times the two of them appeared as neighbors in a specific orientation (i.e. vertical or horizontal). Note that if the edge frequency of a pair of tiles is zero in the input image, the edge is considered invalid. As a result, the WFC algorithm will prevent the output image from having a non-zero frequency for such pairs. The fact that the

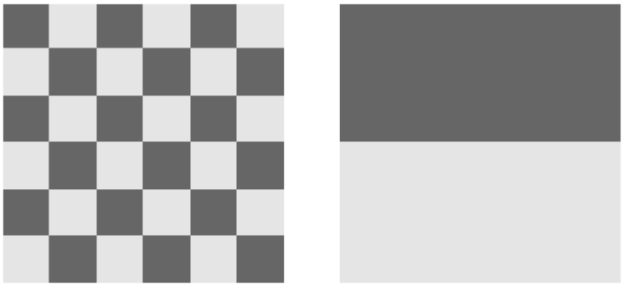

Figure 1: Example of two images with exactly matching *tile* frequencies but low resemblance to each other. In both of the images, exactly half the tiles are light and half are dark.

edge frequencies of output images are always subsets of the edge sets in the source image will be exploited later in our summary metrics.

The metrics defined above can be used to define distributions on the input and output images. To compare the two distributions, other than comparing them by plotting the histograms (examples shown later in this paper), we use the KL-Divergence between the two distributions as a quantitative summary metric.

Commonly used to compare two statistical distributions, the KL-Divergence [9] of the two distributions is defined as follows:

$$\sum_{x \in X} P(x) \log \frac{P(x)}{Q(x)}$$

where Q is the source distribution (associated with the input image in WFC) and P is the target distribution (associated with generated outputs of WFC). Note that the KL-Divergence assumes the two distributions have the same domain. The set of values X is the set of tiles or edges in the generated images. As a result, the value of $P(x)$ is never zero. But it should also never happen that $Q(x)$ is zero when $P(x) > 0$. This assumption is true here for both the tile frequency and the edge frequency. The WFC algorithm guarantees that every tile and edge in the output are valid, meaning that the same tile and edge have appeared in the input at least once.

4 MAKING CONTEXT-SENSITIVE DECISIONS

This section describes our new method of making decisions about what to place at each location in the design grid in a way that is sensitive to the decisions made during earlier iterations of the overall WFC algorithm.

4.1 Definition

The WFC algorithm follows the idea that tiles of the output can be represented with a metaphorical *wave function* that models yet-to-be-design parts of the output as being in a superposition of possible states. At first, all the tile locations of the wave are in an unobserved state. Then it iteratively does the following four steps: it selects an unobserved tile location, it decides what tile to put at that location, it applies this choice (or metaphorical *observation*) to the wave, and finally, it updates the possible values of the other unobserved

```
while not is_complete(wave):
  location = selection_heuristic(wave)
  decision = decision_heuristic(wave, location)
  wave = apply_choice(wave, location, decision)
  wave = propagate_constraints(wave)
```

Figure 2: Pseudocode for the main loop of the WFC algorithm. To implement the heuristic proposed in this paper, the only change would be in the decision heuristic function.

locations in the wave using a standard constraint propagation algorithm [12]. Pseudocode for the main loop of this algorithm can be found in Fig. 2. Consistent with previous interpretations [8], we refer to the two heuristics that control this process as the selection heuristic and the decision heuristic, which select what location to observe and decide what tile to put at that location respectively.

In Gumin's implementation, the decision heuristic is to have the probabilities for the possible tiles at the selected location to be proportional to their frequency in the input. This was defined with the intent to satisfy the C2 constraint mentioned earlier. We can show the distribution this heuristic is sampling by the following equation:

$$\forall x \in \text{PossibleTiles(location)}$$
$$p(x) = \frac{\text{freq}(x)}{\sum_{y \in \text{PossibleTiles}(location)} \text{freq}(y)}$$

Note that the denominator normalizes the frequencies to a probability in range $[0, 1]$ and with a sum of 1.

We propose an alternative decision heuristic that more closely models the input. Additionally, we show that implementing this new heuristic only needs small changes to the code, and it has an insignificant effects on the performance of the WFC algorithm.

Our proposed heuristic, **Context-sensitive Decision Heuristic**, is defined by sampling from a conditional probability as follows:

$$\forall x \in \text{PossibleTiles(location)}, c = \text{ContextAround(location)}$$
$$p(x|c) = \frac{p(x, c)}{p(c)}$$

We define the context around a location to be a tuple consisting of the previously-decided tile values of the neighboring locations (or a special marker for neighboring locations that have yet to be decided). Note that in general this context includes any edges directly connected to a single tile and is not limited to generating 2D images.

Because the value of $p(c)$ is the same for all the possible tiles x, we can ignore calculating it.

We estimate $p(x, c)$ as the frequency of (x, c) in the original image where the values of c come from masking all possible sets of neighbors. That is, even though every tile is seen with a *complete* context in the input image, we synthesize additional observations of each tile in situations where only some of its neighbors have previously been decided. For two-dimensional ($d = 2$) images, we tabulate each source image tile as occurring in $2^{2d} = 16$ different contexts. These frequency tables are turned into sampling distributions via normalization.

Our tile sampling distribution can be expressed by the following:

$$\forall x \in \text{PossibleTiles}(location), c = \text{ContextAround(location)}$$

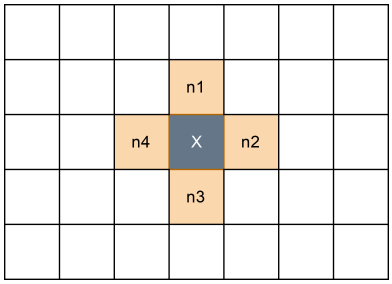

Figure 3: Set of tiles in the context of tile `x`. The context of tile `x` would be its four neighbors, or `(n1, n2, n3, n4)`.

$$p(x|c) = \frac{\text{freq}(x, c)}{\sum_{y \in \text{PossibleTiles(location)}} \text{freq}(y, c)}$$

4.2 Software Implementation

To implement this concept inside WFC, we add a preprocessing step to calculate the joint frequencies of each tile and context, or freq(x, c). This can be calculated by iterating over all the tile locations in the source image. For each location, we create the tuple `(x, n1, n2, n3, n4)` in which `x` is the tile value of that location and `n1` to `n4` are the tile values of its four (two-dimensional) neighbors, as shown in Fig. 3. Then, we keep track of the number of times this key appears in a dictionary. This is similar to keeping track of the frequency of tiles in Gumin's implementation, with the only difference being that the table index now consists of a tuple of 5 tile values instead of a single tile value.

We introduce an `UNK` (unknown marker) tile showing that a value is unknown at the moment (not yet decided). We use this tile in two different cases: first, if the tile is out of boundary (both in the input and output image) and second, when the neighbor tile in the context is unobserved at the moment. Based on this, we should also populate the values of $freq(x, c)$ for when one or more tiles in the context are changed to `UNK`.

To implement this, we need to count 1 toward $freq(x, c')$ for each time we encounter the (x, c) key. If none of the four tiles in c are already `UNK` (because of being out of bounds), this creates $2^4 = 16$ different keys for c'. If one of them is already `UNK`, it creates $2^3 = 8$ c' keys, and so on.

Next, let us consider the changes in the WFC's main loop. The only change involves the part that implements the decision heuristic. Instead of using $freq(x)$, we simply use $freq(x, c)$ in which c is the context key for the neighbors of this location (and `UNK` for the unobserved neighbors). The pseudocode for the new decision heuristic as well as Gumin's version can be seen in Fig. 4.

During execution, the WFC algorithm may encounter many novel contexts (arrangements of previously decided tiles that never occurred in the input image but are nevertheless still valid because they satisfy all immediate adjacency constraints). As a fall-back condition, if at some point the context around the selected location has never appeared in the original image for any of the possible tiles (i.e. $freq(x, c) = 0$ for every x in the possible tiles at some location), we use Gumin's frequency weighted probabilities (i.e. $freq(x)$) for the decision heuristic at that location.

```
def gumin_decision_heuristic(wave, location):
  possibilities = wave[location]
  return sample(possibilities * weights)

def context_decision_heuristic(wave, location):
  possibilities = wave[location]
  context = construct_context(wave, location)
  if context in context_weights:
    return sample(possibilities * context_weights[context])
  else:
    return gumin_decision_heuristic(wave, location)

```

Figure 4: The pseudocode for Gumin's decision heuristic vs. the context-sensitive decision heuristic. While Gumin's tile frequency heuristic samples the frequency of the possible tiles to decide on a tile, our proposed heuristic samples the frequency of tiles in the specific context of the selected location. Note that this is the only difference between the two methods.

4.3 Performance Analysis

In terms of memory consumption, the size of the dictionary used for keeping track of frequencies is at most 16 times the number of tiles in the source image to account for putting `UNK` for each of the 4 neighbors. In practice, it is hardly even close to this number because most of the context keys are not unique, especially when part of them is replaced with `UNK`. Furthermore, the size of this dictionary is negligible compared to the memory used for keeping track of all the possibilities at each location (the wave), and is not sensitive to the size or number of outputs to be generated.

As for the execution time, the preprocessing time which can be 16 times more than Gumin's method is negligible compared to the execution time of the constraint solving phase of WFC. In the main loop of WFC, the only difference is during the decision heuristic. The key is created from 5 values instead of 1, but because accessing a key in a dictionary (hash table) is $O(1)$, the number of keys in the dictionary does not affect the execution time of accessing a key.

5 COMPARING DECISION HEURISTICS

This section describes evaluation of the impacts of our new decision heuristic on input–output resemblance.

5.1 Experimental Setup

To compare the different decision heuristics, we implemented the WFC algorithm in Python including three different options for the decision heuristics and three for the selection heuristics.[1]

Among other configurable options, our implementation allows us to customize both the *selection* and *decision* heuristics. The implemented selection heuristics are the location with the lowest number of options, the location with the lowest Shannon entropy for the tile probabilities at that location, and finally selecting tiles in order from top left to bottom right (lexical). We find these heuristics to not have an interesting impact on the resemblance metrics defined earlier. Because of this and also because the main focus of this paper

[1]Our python implementation can be found at https://github.com/iambb5445/context-sensitive-WFC

is changing the decision heuristics, we only include the results for the lexical heuristic. This experimental design choice has the effect that all variation among the multiple outputs of the generator is attributable to stochasticity of the *decision* heuristic.

As for the decision heuristics, the three options we implemented are Gumin's Tile Frequency heuristic, our context-sensitive heuristic, and a uniform sampling heuristic as a baseline. For each of the examples in this section, we run all three of the decision heuristics to show how our proposed heuristic affects the resemblance metrics in comparison with the other decision heuristics.

Finally, we implemented two ways of handling contradictions that arise during search. If there is a situation in which a position does not have any valid options to choose, we either let that position to be blank, or we try back-tracking our decisions to find another solution. In Gumin's original implementation, the algorithm would repeatedly restart upon encountering a contradiction until a contradiction-free result was found. Given that our quantitative results would already be averaged over many samples drawn with different random seed values, we eliminated this restart behavior in our experiments.

For a qualitative analysis of resemblance (allowing the reader to form their own judgements by direct inspection), we report one uncurated example output for each of our scenarios with the three decision heuristics mentioned above and in two different output sizes (20 by 20 and 100 by 100 tiles). We found the large scale outputs to show the overall style better, while the smaller outputs show more details of the quality of the generated images.

Our quantitative analysis reports results averaged over 100 runs of the respective algorithms with a 20 by 20 target output size to show how using different decision heuristics affects them.

In the following subsections, we analyze the quality of the results and the resemblance metrics for each scenario. A summary of the resemblance metrics can be found in Table 1.

5.2 Black and White Stick

One of our goals in our experiments is to show the effectiveness of our context-sensitive decision heuristic in a minimal, easy to understand example. Motivated by this goal, we defined an example image with only two different pixels: black and white (rendered as light and dark for contrast with the paper's background color). The input image consists of only 7x7 pixels, and the only white pixels are located at the fourth column, all the rows except first and last, as shown in Fig. 5. We consider each tile to be a single pixel of the image. We found this example easy to understand and to help us discuss the behavior of different heuristics.

In this toy scenario, all the neighboring combinations of tiles are valid in all orientations except for two white tiles horizontally being neighbors. As a result, when the tiles to the right or left are not white, the uniform heuristic chooses black and white tiles with a 50-50 chance. Otherwise, it chooses a black tile. So we expect the output distribution to be almost 50-50, but more toward choosing black tiles.

The Tile Frequency heuristic at each position chooses black and white tiles with a 44 to 5 chance, because out of the 49 tiles in the input image, only 4 were white tiles. As a result, it is rarely bound by the white-white invalid horizontal neighboring and can match

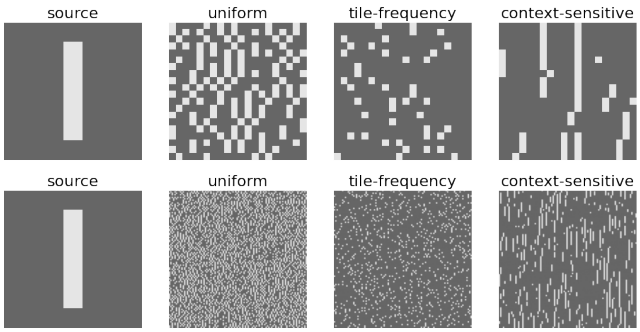

Figure 5: The output of running WFC on the stick example. The source image consists of 7 by 7 tiles, each either black or white (rendered as light and dark for contrast with the paper's background color). The outputs are generated by using uniform distribution heuristic, tile frequency heuristic, and our new context-sensitive heuristic. The first row shows outputs with size 20 by 20 while the second row shows outputs with size 100 by 100, showing the overall cohesiveness of the output for each decision heuristic.

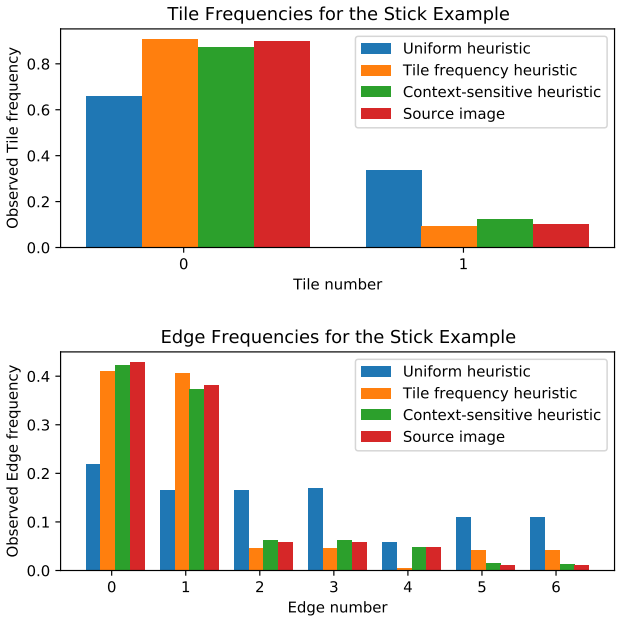

Figure 6: The empirical distribution of tiles and edges for each of the three different heuristics for the stick example. The right-most bar of each cluster represents the the ideal outcome. Distributions associated with each heuristic are the result of averaging over 100 outputs, 20 by 20 tiles each (showing *systematic biases* rather than attempting to visualize variance). The context-sensitive heuristic seems to sacrifice the tile frequency resemblance slightly to create considerably more accurate edge frequency resemblance (matching our own visual interpretation of output images).

Table 1: The resemblance metrics for each scenario over 100 outputs of size 20 by 20. Recall from the metric definitions (Section 3) that lower KL-divergence values represent a better match between outputs and inputs.

Example	Decision Heuristic	Tile Frequency Resemblance	Edge Frequency Resemblance
Stick	Uniform	0.21	0.57
Stick	Tile Frequency	0.00041	0.084
Stick	Context-sensitive	0.0020	0.00048
Zelda Map	Uniform	0.43	1.48
Zelda Map	Tile Frequency	0.14	1.50
Zelda Map	Context-sensitive	0.027	0.047

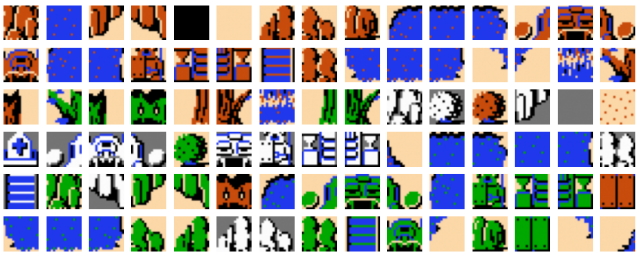

Figure 7: The set of tiles in the Zelda map. There are 90 unique tiles in the zelda map, each of size 16x16 pixels.

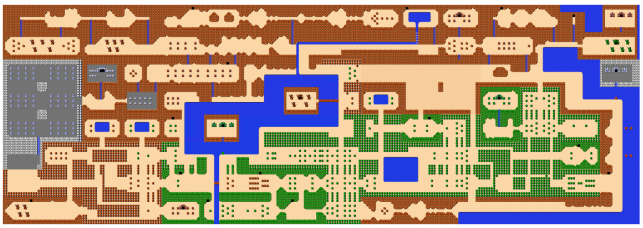

Figure 8: The Zelda map source image. Source: https://nesmaps.com/maps/Zelda/Zelda.html

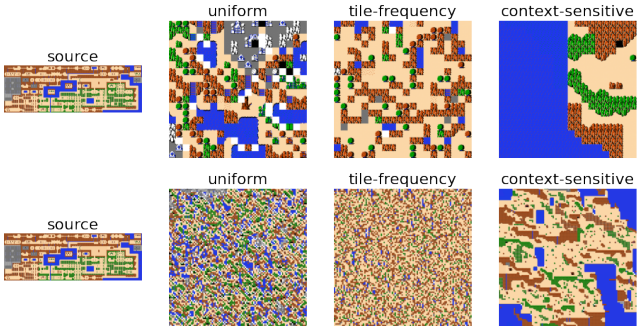

Figure 9: The output of running WFC on the Zelda map example. The first row are outputs with size 20 by 20 while the second row are outputs with size 100 by 100, showing the overall cohesiveness of the output for each decision heuristic. The white tiles show contradictions. Note the presence of large-scale patterns that only emerges in the context-sensitive case.

the input tile distribution almost perfectly, as shown in the tile frequency histogram of Fig. 6.

The context-sensitive heuristic results qualitatively resemble the input more accurately as shown in Fig. 6 histograms and Fig. 5 example results. The reason is that when deciding the tile in a position with a white tile above or below, it prioritizes the white tiles, even though the black tiles are more frequent in the input image. The reason is that in the context of having a white tile above or below, the conditional probability of the white tile is more than black tile (out of the 5 tiles with a white tile above in the input image, only one is a black tile). In other words, it sacrifices matching the tile frequency metric based on the context. For the same reason, it has better results based on all the metrics except the tile frequency metric, in which it has a higher (worse) KL-Divergence to the tile frequency of the input image compared to Gumin's frequency metric, as shown in Table 1.

5.3 Zelda Overworld Map

To show the difference our proposed heuristic makes in an example in a realistic setting, we use the overworld map from the first *Legend of Zelda* game [13]. As shown in Fig. 7, this map consists of 90 different tiles, each of them 16 by 16 pixels.

As seen in Fig. 9, the context-sensitive heuristic has a significant impact on how cohesive the output map is. Note that in all three of the decision heuristics, every two neighboring tiles are valid and have appeared at least once as neighbors in that specific direction in the input. However, the output for the uniform distribution does not resemble a Zelda map. The output for Gumin's tile frequency heuristic, even though better than the uniform distribution, still lacks the overall cohesiveness and does not resemble the input map, which is evident in both the tile frequency metric and the edge frequency metric, as demonstrated in Table 1.

At first glance, it might seem counter-intuitive that Gumin's tile frequency heuristic which focuses on having the same tile frequency in the input and output fails to do so compared to the context-sensitive heuristic. The reason is that tile frequency heuristic chooses the most frequent tiles most of the time, and the edge constraints prevent the WFC algorithm from trying new areas with other, less frequent tiles. This results in the *systematic* overuse of frequent tiles and underuse of rare tiles, as evident in Fig. 10. For larger and more interesting designs where special tiles are used sparingly among a sea of more common tiles, the systematic bias of

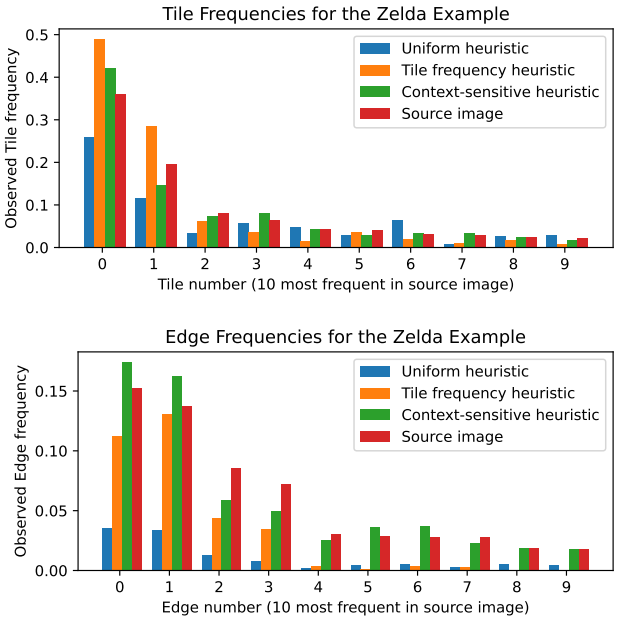

Figure 10: The distribution of tiles and edges for each of the three different heuristics for the Zelda map example. Results are for 100 samples of 20 by 20 tiles each. The context-sensitive heuristic performs better than the other two heuristics in both the tile frequency distribution and edge frequency distribution. According to both frequency distributions, the context-sensitive method shows better resemblance.

the original heuristic becomes worse. The algorithm needs to use lower-frequency transition tiles in order to unlock the ability to place other varieties of high-frequency tiles. Note that in the first example (black and white stick) the simplicity of the input and the fact that most of the edges were valid gave the frequency heuristic the power to match the tile frequency distribution closely. As we see here, this is not the case in many more complex and realistic examples.

5.4 Minecraft

As we mentioned in the Definition section, the definition of context and the context-sensitive decision heuristic is not limited to the 2D setting. We defined the context as the set of tiles immediately adjacent to the target tile. Applying this definition to the 2D environment resulted in a tuple of 4 tiles, but in an N dimensional setting the context would be a tuple of 2N tiles. This results in a tuple of 6 tiles in 3D, for the 6 directions (forward, backward, up, down, left, and right). We can then populate the (tile, context) frequency lookup table and the implementation of context-sensitive heuristic remains unaffected. Whenever we are deciding what tile to choose for a location, we find the context around the target location and look up $freq(x, c)$ in which x is the tile option and c is the context tuple. Finally, we sample the distribution of tiles weighted by the value of $freq(x, c)$ for each valid tile option x.

Figure 11: The training data for the Minecraft example. The training data consists of 80 by 32 by 96 blocks. This image is an in-game screenshot, but the saturation and brightness have been modified for better visibility.

To show that the benefits of using the context-sensitive heuristic are not limited to 2D environments, we run the same experiment in a 3D setting of Minecraft game. Note that this is not to showcase the ability of WFC to generate a Minecraft world, but to show how the benefits of the context-sensitive heuristic are not specific to the dimension of the data.

A Minecraft world consists of a randomly generated grid of blocks. We use a subsection of this world of size 80 by 32 by 96 blocks as the training data, as shown in Fig. 11. We select this section only because it contains a Minecraft village, which has more complex structures than other parts of the world. To completely capture the block properties such as block orientation, we assign each block and its set of properties a unique identifier. This results in 121 possible tiles for each grid location based on the possible options in the training set.

We define our generation task to generate a 16 by 16 by 16 chunk of blocks, in which a 4 by 16 by 16 section is exactly copied from part of the training data. We use the copied part of the data to constraint the generation and to see how well different decision heuristic can continue the chunk while maintaining its cohesiveness. The results are shown in Fig. 12.

A qualitative analysis of the results agrees with the results from 2D environments and shows that the context-sensitive heuristic performs significantly better in generating coherent results. Note that in all of the scenarios the tile size is one block.

6 BIGGER PATTERNS

Recall from the introduction that increasing the pattern size is the typical way to achieve better resemblance in the output.[2] This means that at each position of the resulting image instead of choosing a tile (a 1 by 1 pattern), we can choose an N by N pattern which overlaps with the patterns at its neighboring positions, as shown in Fig. 13. This has the advantage that it will provide more information when choosing patterns that contain those neighboring tiles.

[2]In the literature of constraint-based generators driven by examples, it is expected that small increases in size of the algorithm's analysis window can lead to large increases in the rate of *plagiarism*, requiring special techniques to counteract [14].

Table 2: The resemblance metrics for each scenario over 100 outputs of size 20 by 20, with 3x3 overlapping patterns. Recall from the metric definitions (Section 3) that *lower* KL-divergence values represent a better match between outputs and inputs. The last row is copied from the previous section to make comparing the results easier. Note that even when using larger patterns increases in resemblance are still possible with better heuristics.

Example	Pattern Size	Decision Heuristic	Pattern Frequency Resemblance	Pattern-Edge Frequency Resemblance	Tile Frequency Resemblance	Edge Frequency Resemblance
Zelda Map	3x3	Uniform	0.51	0.78	0.078	0.18
Zelda Map	3x3	Tile Frequency	0.17	0.30	0.029	0.051
Zelda Map	3x3	Context-sensitive	0.11	0.19	0.016	0.030
Zelda Map	1x1	Context-sensitive	-	-	0.027	0.047

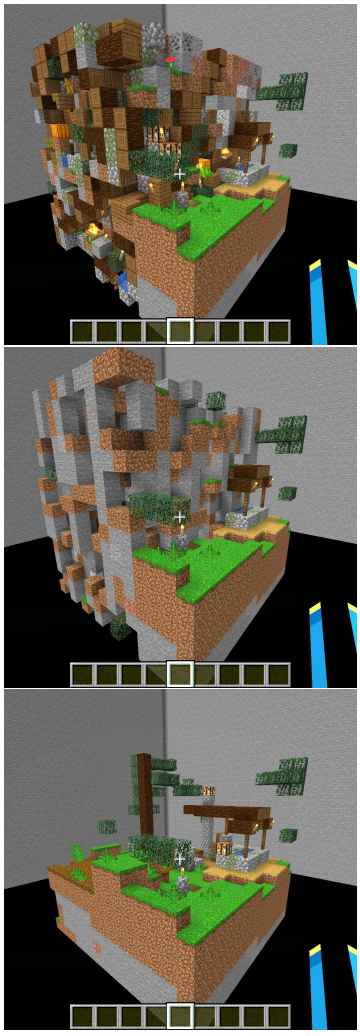

Figure 12: The output of running WFC on the Minecraft example. Top: Uniform heuristic. Middle: Tile frequency heuristic. Bottom: Context-sensitive heuristic. In all cases, the first 4 by 16 by 16 blocks were fixed. These are in-game screenshots, but the saturation and brightness have been modified for better visibility.

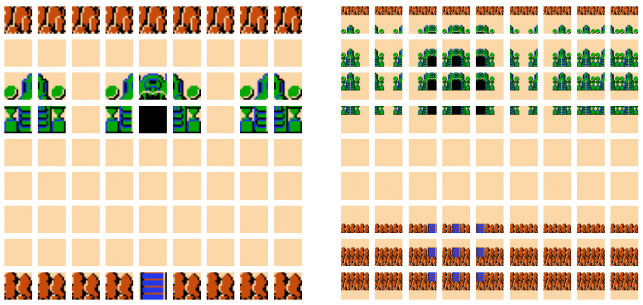

Figure 13: A section of the Zelda map, divided by tiles (left) and overlapping 3x3 patterns (right). Note that here the source image is wrapped around to account for the tiles outside of its boundary.

Additionally, this will affect Gumin's decision heuristic to choose based on the pattern frequency instead of tile frequency. Note that this does not affect the constraint-solving core of the WFC implementation at all. The main loop of WFC is ignorant to what type of information each tile option contains and how it affects the other location and is only aware of the set of valid tiles and neighborings.

In this section, we document the result of increasing the pattern size to 3x3 for each of the scenarios of the Zelda map example considered in the previous section. Because the WFC algorithm and the decision heuristics now consider frequencies at the pattern level, we report the resemblance metrics based on pattern frequency and pattern edge frequency. On the other hand, to compare these results with results from the previous section, we need to calculate the resemblance metrics at tile level as well. Table 2 summarizes these results.

Overall, the results indicate that when using overlapping patterns, the context-sensitive heuristic performs better than the other two with all the metrics. Even without using overlapping patterns, the context-sensitive heuristic performs better than the other two, but only slightly.

Based on the results, it might seem like using overlapping patterns can be a safe replacement for using the context-sensitive heuristic. But this is not true, as using the overlapping patterns introduces many disadvantages compared to using the context-sensitive heuristic.

First, using overlapping patterns changes the number of building blocks in the WFC algorithm drastically. For example, in the Zelda map example there are only 90 unique tiles, but there are 2958 unique 3x3 patterns used in the source image. Not only does this have a significant impact on the execution time of the algorithm, but also it increases the chance of encountering contradictions. If we want to only get valid outputs without blank positions, we have to either restart on contradictions, or use backtracking. The contradictions now can happen enough to make the execution time change exponentially based on the size of the output, even when using backtracking.

Additionally, each pattern can only be matched with a small portion of all the possible patterns. This means that there are fewer possible legal ways of putting the patterns back together. As a result, the outputs will be less diverse, and copying parts of the input happens more frequently. In other words, to get the same diversity in results and lower the number of contradictions, we would need more samples of the input data. Similar to the curse of dimensionality, because of the increase in the number of patterns, we need more samples to cover all the needed information. To have enough information about the possible neighborings for each NxN pattern, instead of E edges our data needs to include up to E^N edges, because each adjacency now includes N pairs of neighboring tiles instead of 1.

To look at this from another angle, our very simple context-sensitive heuristic is able to make the decision process to care about the context tiles without impacting the set of legal tiles. Every valid output from running WFC with the uniform heuristic or Gumin's tile frequency heuristic is still valid when using the context-sensitive heuristic. But using the overlapping patterns makes many of the outputs illegal and changes the number of constraints, resulting in less diversity, more contradictions and increased execution time.

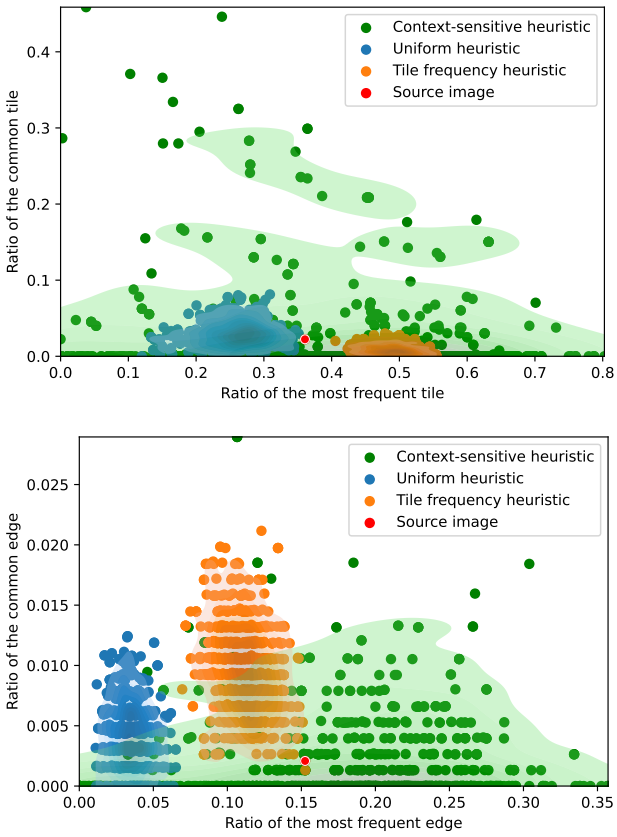

Figure 14: The Expressive Range Analysis of the three heuristics. The top image is the tile-based expressive range while the bottom image shows the edge-based expressive range. In both cases each point represents a generated image of size 20 by 20 out of the 1000 samples generated for each heuristic. The source point represents the ratio of the same tile and edge in the source image. In both the tile-centric analysis and the edge-centric analysis, our method produces distribution of designs that is both more broad and more centered on the training data than the earlier methods.

7 EXPRESSIVE RANGE ANALYSIS

So far, we have shown that the context-sensitive heuristic uses a small modification on the decision heuristic to significantly improve the resemblance. One question that might arise is what effect does this heuristic have on the expressive range of the WFC algorithm. To answer this question, we analyze the expressive range for the Zelda map example from section 5.3.

In order to describe the expressive range of the WFC algorithm for each of the heuristics, we must be able to comapare the generated results. We define the following metrics to do so:

- Ratio of the most frequent tile: The ratio of the most frequent tile to all tiles in a generated image. The most frequent tile is the tile that appeared the most in the source image.
- Ratio of a common tile: The ratio of a common tile to all tiles in a generated image. The common tile is the tile that ranked at the 10th percentile mark in terms of frequency in the source image. In the zelda example, given that we have 90 tiles, this is the tile that ranked 9th in terms of frequency in the source image.
- Ratio of the most frequent edge: The ratio of the most frequent edge to all edges in a generated image. The most frequent edge is the edge that appeared the most in the source image.
- Ratio of a common edge: The ratio of a common edge to all edges in a generated image. The common edge is the edge that ranked at the 10th percentile mark in terms of frequency in the source image.

These metrics have been chosen similar to the resemblance metrics: they compare results based on the frequency of tiles and edges. The reason that we have chosen specifically the most frequent tile and a common tile is that for less frequent tiles, the ratio is almost always very close to zero and has a smaller range for both the source image and the generated ones. The same goes for the edges.

To do the analysis, first we generate 1000 samples of size 20 by 20 for each of our heuristics. We then calculate the tile ratio metrics and edge ratio metrics. Fig.14 shows the metrics in the results as well as the source image.

As the reader can see, in both the tile-based and edge-based analysis the context-sensitive heuristic has a significantly wider range of outputs. More importantly, in both cases the context-sensitive heuristic's expressive range better encompasses the source image data point.

In the case of the uniform heuristic, the heuristic uses a uniform distribution and doesn't prioritize any tile over the other. Because of that, the common tiles and edges have a lower frequency compared to the source image.

As for the tile frequency heuristic, while it can mimic the frequency of the common tile in the source image quite well, it overuses the most frequent tile. This is perhaps due to the fact that the most frequent tile is less constrained and appears in the set of valid tiles more often. Then, because the heuristic does not care about the context, the tile always has a higher probability of getting chosen. More importantly, because the heuristic focuses on the frequency of tiles, it has a very limited tile-based expressive range. This agrees with the qualitative evaluation of the generated results.

The tile frequency heuristic also doesn't perform well in the edge-based expressive range analysis. It almost always overuses the common edge while underusing the most frequent edge. In contrast, the context-sensitive heuristic shows samples of both overusing and underusing those edges, showing a wider range of possibilities and a broader expressive range.

8 DISCUSSION

Recall from the introduction that our research question focused on small and replicable enhancements to WFC that increase resemblance. Historical trends in the computer graphics literature (for texture synthesis) seem to suggest that the problems associated with using larger and larger pattern sizes need to be addressed with richer [19] and richer [3] statistical modeling methods. The massive data requirements for these approaches are at odds with the typical usage scenarios for WFC where the artist provides a single small example and expects the algorithm to work without further intervention.

For example, in the recent MaskGIT paper [2], the authors describe how they train a neural network to select a discrete tile to place at each location in a grid conditional on the state of tiles elsewhere in the grid. A key difference of this work relative to our proposal is that the complex MaskGIT model allows any image tile to influence any other image tile. This is great in terms of increasing resemblance, but it significantly complicates the model's architecture and the amount of data to train the model without problematic overfitting (sometimes referred to as *plagiarism*). The global attention mechanism in recent Transformer models [5, 18] is needed to express these long range dependencies.

In contrast with these trends, in our method **we intentionally choose an attention pattern that is specifically matched to the scale of constraints already considered by WFC** and is trainable with simple counting rather than gradient descent.

9 CONCLUSION

In this work, we introduced resemblance metrics that were able to show limitations of common WFC implementations. In response, we showed how a simple new heuristic could close the newly illustrated quality gap without changing the overall flow of the algorithm. Overall, this is an example of exerting more control over the statistical properties of constraint-based generators. The way in which we learn heuristics to guide constraint-solving search has implications for future generators that want to use very rich statistical models of style while making strong guarantees about features of the designs they produce.

REFERENCES

[1] Supratik Chakraborty, Daniel Fremont, Kuldeep Meel, Sanjit Seshia, and Moshe Vardi. 2014. Distribution-aware sampling and weighted model counting for SAT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

[2] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. 2022. MaskGIT: Masked Generative Image Transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, New Orleans, LA, USA, 11315–11325. https://doi.org/10.1109/CVPR52688.2022.01103

[3] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. 2020. Generative pretraining from pixels. In *International conference on machine learning*. PMLR, 1691–1703.

[4] Seth Cooper. 2022. Sturgeon: Tile-Based Procedural Level Generation via Learned and Designed Constraints. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 18, 1 (Oct. 2022), 26–36. https://doi.org/10.1609/aiide.v18i1.21944

[5] Patrick Esser, Robin Rombach, and Bjorn Ommer. 2021. Taming Transformers for High-Resolution Image Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Nashville, TN, USA, 12873–12883. https://doi.org/10.1109/CVPR46437.2021.01268

[6] Maxim Gumin. 2016. Wave Function Collapse Algorithm. https://github.com/mxgmn/WaveFunctionCollapse

[7] Isaac Karth and Adam M Smith. 2019. Addressing the fundamental tension of PCGML with discriminative learning. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*. 1–9.

[8] Isaac Karth and Adam M. Smith. 2022. WaveFunctionCollapse: Content Generation via Constraint Solving and Machine Learning. *IEEE Transactions on Games* 14, 3 (2022), 364–376. https://doi.org/10.1109/TG.2021.3076368

[9] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* 22, 1 (1951), 79–86.

[10] Thijmen Stefanus Leendert Langendam and Rafael Bidarra. 2022. MiWFC - Designer Empowerment through Mixed-Initiative Wave Function Collapse. In *Proceedings of the 17th International Conference on the Foundations of Digital Games* (Athens, Greece) *(FDG '22)*. Association for Computing Machinery, New York, NY, USA, Article 66, 8 pages. https://doi.org/10.1145/3555858.3563266

[11] Paul Merrell. 2007. Example-Based Model Synthesis. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (Seattle, Washington) *(I3D '07)*. Association for Computing Machinery, New York, NY, USA, 105–112. https://doi.org/10.1145/1230100.1230119

[12] Roger Mohr and Thomas C Henderson. 1986. Arc and path consistency revisited. *Artificial intelligence* 28, 2 (1986), 225–233.

[13] Nintendo. 1986. Legend of Zelda. [Family Computer Disk System].

[14] Alexandre Papadopoulos, Pierre Roy, and François Pachet. 2014. Avoiding plagiarism in Markov sequence generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

[15] Adam M Smith and Michael Mateas. 2011. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 187–200.

[16] Sam Snodgrass and Santiago Ontanón. 2016. Controllable Procedural Content Generation via Constrained Multi-Dimensional Markov Chain Sampling.. In *IJCAI*. 780–786.

[17] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games* 10, 3 (Sep. 2018), 257–270. https://doi.org/10.1109/TG.2018.2846639

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All

you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc., Long Beach, CA, USA. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[19] Li-Yi Wei and Marc Levoy. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. 479–488.