Punch Out Model Synthesis: A Stochastic Algorithm for Constraint Based Tiling Generation

Zzyv Zzyzek [1]

Abstract

As an artistic aid in tiled level design, Constraint Based Tiling Generation (CBTG) algorithms can help to automatically create level realizations from a set of tiles and placement constraints. Merrell's *Model Synthesis* (MMS) and Gumin's *Wave Function Collapse* (WFC) have been proposed as Constraint Based Tiling Generation (CBTG) algorithms that work well for many scenarios but have limitations in problem size, problem setup and solution biasing. We present Punch Out Model Synthesis (POMS), a Constraint Based Tiling Generation algorithm, that can handle large problem sizes, requires minimal assumptions for setup and can help mitigate solution biasing. POMS attempts to resolve indeterminate grid regions by trying to progressively realize sub-blocks, performing a stochastic boundary erosion on previously resolved regions should sub-block resolution fail. We highlight the results of running a reference implementation on different tile sets and discuss a tile correlation length, implied by the tile constraints, and its role in choosing an appropriate block size to aid POMS in successfully finding grid realizations.

1. Introduction

1.1. Overview

We present Punch Out Model Synthesis (POMS), an algorithm that works on a regular 2D or 3D grid to find a tile placement realization subject to pairwise tile constraints in each grid direction ($\pm X, \pm Y, \pm Z$).

POMS is a grid level stochastic Constraint Based Tiling Generation (CBTG) algorithm whose primary benefits are:

- Requires minimal assumptions on initial setup state
- Has resources that scale primarily with block size and not grid size
- Can reliably find realizations on arbitrarily sized grids with tile constraints that have finite correlation length

and primary drawbacks that:

- Has limited success for tile constraints that have long range correlation length

Further, we:

- Provide the results of running a reference implementation on a variety of tile sets (Section 4)
- Explore the Tile Arc Consistent Correlation Length (TACCL), a heuristic for tile correlation length, that is used to inform the block size choice and solution strategies for a variety of tile sets (Section 4)

Here, a grid realization is a single tile assignment per grid cell that respects the tile constraints.

POMS works by initially setting the grid in an indeterminate state and progressively realizing sub blocks of the grid. Block boundary edges that fall within the larger grid body are *pinned* so that, should a block realization succeed, the block can be re-integrated back into the larger grid. Should block level realization fail, depending on the type of block realization failure, either the failed block region is set to an indeterminate state or the block region is reverted to its previous state and all realized region boundaries within the grid are considered for *erosion* by probabilistically setting them to an indeterminate state.

POMS is a stochastic algorithm because of the reversion step. Undoing previous cell realizations, via block region reversion or erosion, are done in the hopes of removing a localized contradiction. Any expectation of progress for POMS primarily comes from choosing an appropriate block size. Conceptually, correlation length is the influence that a cell tile choice has over other cell tile options at distant locations. In this paper, we attempt to quantify an aspect of correlation length and use it to inform the block size choice.

If there is a finite length of correlation that one cell's tile choice has with another, any contradiction that might appear during the course of resolution are localized to a region. Reverting the region around the contradiction allows for another attempt at finding a realization without destroying the bulk of pre-existing realizations elsewhere in the grid. Under some conditions of configuration randomness, and for many tile constraints, resolved cells located far enough away from each other have little or no influence over one another. The correlation distance informs the block size choice as block sizes chosen large enough allow for the tile values in the middle cells of a block to be chosen independent of any tiles fixed on the boundary.

Expecting center tile choices to be independent of boundary tile values is, in general, unreasonable as the problem is known to be NP-Complete. Even under random configuration assumptions, the issue can be further complicated if the correlation length is not finite, or finite but large. Even though the general problem is likely intractable, or the finite correlation length assumption is violated, many tile constraints are under constrained and the ability of POMS to overcome local constraints provides enough capability to find realizations.

Unfortunately, for many tile sets, simple global constraints, tile distribution or sparse initial configuration restrictions are enough to confound POMS into failing to find a realization. Different choices for block scheduling policies, different block resolution algorithms and other parameter choices can help mitigate these shortcomings and will be briefly addressed (Section 4).

1.2. Definitions

We discuss some preliminary ideas before describing details of the Punch Out Model Synthesis (POMS) algorithm.

EXAG 2024: AIIDE Workshop on Experimental AI in Games, November 18–22, 2024, Lexington, KY, USA

zzyzek@thumpernet.com (. Z. Z.)
https://zzyzek.github.io/ (. Z. Z.)
0009-0005-2175-1166 (. Z. Z.)

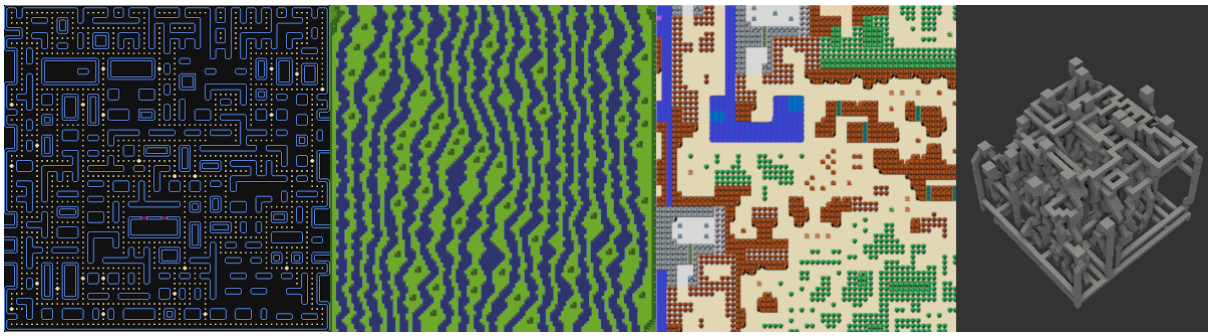

Figure 1: Examples outputs of Punch Out Model Synthesis (POMS) run on different tile sets. From left to right, the *Pill Mortal* tile set, the *Forest Micro* tile set, the *Overhead Action RPG Overworld* tile set and the *Brutal Plum* tile set

A *grid* is a collection of *cells* in the shape of a rectangular cuboid, of size $N_x \cdot N_y \cdot N_z$, $(N_x, N_y, N_z \in \mathbb{N})$.

Each cell in the grid is a variable whose domain is D *tiles* $(D \in \mathbb{N})$. Values in neighboring cells are subject to a set of provided *tile constraints*. Here, the set of tile constraints is pairwise, and only nearest neighbor in each of the major grid dimensions $(\pm X, \pm Y, \pm Z)$.

A tile at a grid cell location is said to have *support from a direction* if there is at least one tile in the neighboring grid cell direction that respects the tile constraints. A tile, at a grid cell location, *has support* or *is supported* if a tile has support in each direction.

A cell is said to be *resolved* if there is only one tile present and the tile has support from its neighbors. Should every cell in the grid be resolved, the grid is said to be resolved. Should a cell hold no tiles, the grid is then said to be in a *contradictory* state.

The set of D tiles is called the *tile domain*. The set of available tile options at a cell is called the *cell's tile domain* and represents the available tiles that can be placed at a given cell location.

The problem of Constraint Based Tiling Generation (CBTG) is to find a single tile assignment to each grid cell location subject to the tile constraints. That is, resolve every cell in the grid.

It is sometimes desirable to pick out a sub region from a grid for special consideration. Here we identify a *block* as a sub region of cells from the grid. Our concern is with blocks that are rectangular cuboid in shape, of size $M_x, M_y, M_z \in \mathbb{N}$, and that can be smaller than the grid size $(1 \leq M_x \leq N_x, 1 \leq M_y \leq N_y, 1 \leq M_z \leq N_z)$.

If every tile in every cell in a block is supported, the block of cells is said to be in an *Arc Consistent* (AC) state. That is, subject to the tile constraints, if every tile in every cell in a block region has at least one valid neighboring tile, the block region is said to be in an arc consistent state.

One method of attempting to put a block region into an arc consistent state is to remove tiles that have no support from the list of permissible tiles at a cell location. Each tile removed can have a cascading effect by potentially causing a tile in a neighboring cell to be unsupported. The repeated process of removing unsupported tiles throughout a block region until a contradiction is encountered or the block region is in an arc consistent state is called *constraint propagation*. Constraint propagation can be used as the basis for a Constraint Based Tiling Generation (CBTG) solver.

We differentiate a *block level* solver and a *grid level* solver. We define a *block level* solver as an algorithm that keeps full state of the block it is trying to solve by propagating constraints and maintaining arc consistency throughout its run. A *grid level* solver need not keep full state of the grid and often will only keep minimal information about whether a grid cell is resolved or is in an indeterminate state. A block level solver typically needs more resources as, for example, it might maintain a memory intensive data structure associated with maintaining arc consistency.

POMS is a grid level solver with one of its input parameters designated to specify which underlying block level solver to use. In this paper we use *Breakout Model Synthesis* (BMS), a stochastic block level solver introduced in Hoetzlein's `just_math` project [1]. For completeness, pseudocode for *Breakout Model Synthesis* is given in Appendix A.

Note that since POMS is a grid level solver, the grid can be in an arc *inconsistent* state during the course of the algorithm. This poses no problem in and of itself as the block level solver will attempt to put the block in an arc consistent state while trying to make progress towards a fully resolved grid.

2. Related Work

To our knowledge, Merrell was the first to introduce the modern formulation of Constraint Based Tiling Generation (CBTG) [1] [3, 4]. Merrell introduced a Modify in blocks Model Synthesis (MMS) algorithm that starts with a fully resolved grid and applies a one shot block level constraint solver on sub-blocks within the grid.

Merrell noticed that the block level algorithm undergoes a phase transition of solvability [2] , with a decreasing probability for successful block resolution as block size increases. Instead of attempting to resolve a large grid in one try, the MMS algorithm progressively resolves sub-blocks and reintegrates them back into the grid. If a block level resolution fails, the block is discarded without altering the grid and another block is chosen.

For many problems, MMS is ideal as it always keeps a full resolution of the grid throughout its run. Merrell introduced a sequential overlapping schedule for block choice and, for some suitable assumptions on block size and underlying

[1] The term *Constraint Based Tile Generators* was coined by Adam Newgas [2] and has been adopted in this paper, with slightly different wording, as the name for the specialization of the more general Constraint Satisfaction Problem.

[2] Note that the phase transition is only for *fallible* models. *Infallible* models will be discussed briefly later in this section

distribution, the mixing time can be quick, requiring only a few passes through the grid.

Unfortunately, MMS has two major drawbacks, the second of which Merrell noticed and discussed in his thesis:

- The requirement of an initially resolved state to start MMS might be difficult to achieve, either in a practical or theoretical sense.
- Features bigger than the chosen block size will be missed by MMS as there is no way to realize larger features through single block level alterations.

Many of the tile sets and tile constraints that Merrell provides in [3, 4] have an "empty" tile that has itself as an admissible neighbor, creating a situation where the initial fully resolved grid can be easily created by populating each cell with an "empty" tile. All 2D tile sets and tile constraints presented in this paper do not have a valid resolution with a single replicated tile and require some amount of knowledge about the tile constraints to create a fully resolved configuration.

In general, for some tile constraints, finding a class of fully resolved initial configurations might be done through engineering effort. For example, it might be possible to find a small tileable block that is then able to be replicated through to the whole grid [3].

For other tile constraints, finding any fully resolved grid from an initially indeterminate state might be the goal. In such cases, running MMS would defeat the purpose, as MMS can't start without a solution but if a solution is known, MMS wouldn't be needed. As an example, a tile set can be constructed where a grid realization connects two endpoints via a path without explicitly describing where the path meanders through the grid. MMS would require a full realization of the path before it could start.

POMS sidesteps the need for a fully realized initial configuration by allowing the grid to be in an indeterminate state.

The inability to handle certain types of unbounded constraints, such as the implicit top and bottom equal river count constraint that appear in Woźniak's *Forest Micro* tile set (section 4), is the deeper issue with MMS. Choosing a small block size for MMS could miss novel features whereas too large of a block size either decreases the likelihood of block resolution or turns MMS into a block level solver.

Gumin introduced the Wave Function Collapse (WFC) project which improved the block level solver used in MMS and added facilities for automatic tile constraint deduction from exemplar scenes [5]. WFC uses a maximum entropy heuristic to choose which cells to resolve. Though extensions are possible, WFC as presented by Gumin is a one-shot block level solver, giving up should a contradiction be encountered.

Since MMS is a grid level solver, other block level solvers can be used, such as WFC, to resolve underlying blocks, with modifications added to allow for constraints, boundary conditions and other relevant features. Merrell provides a comparison between MMS and WFC in [6].

Breakout Model Synthesis (BMS) was introduced in Hoetzlein's `just_math` project [1]. Hoetzlein's `just_math` project also introduced the Tile Arc Consistent Correlation Length (TACCL) after noticing that knowledge of the tile correlation length could be used to create algorithms that took advantage of it. POMS takes the ideas of tile correlation and the TACCL to inform its backtracking strategies when applied to the larger grid without needing the resource requirements that BMS, as a block level solver, would require.

Algorithm	*WFC*	*BMS*	*MMS*	*POMS*
Solver Type	Block	Block	Grid	**Grid**
Contradiction Resilience	No	Yes	Yes	**Yes**
Block Step Consistent	n/a	n/a	Yes	***No***
Indeterminate Initial State	Yes	Yes	No	**Yes**
Ergodic	Yes	Yes	No	**Yes**

Table 1
WFC: Gumin's Wave Function Collapse
BMS: Breakout Model Synthesis
MMS: Merrell's Modify in Blocks Model Synthesis
POMS: Punch Out Model Synthesis (our algorithm)

Table 2 provides a summary of the differences between WFC, BMS, MMS and POMS. Here, *contradiction resilience* means that the algorithm can recover should a contradiction be encountered, *block step consistent* means the algorithm is in an arc consistent state after every block resolution, *indeterminate initial state* means the algorithm doesn't require a fully resolved initial configuration and *ergodic* means that, in general, all solution states are possibly to reach. Note that an assertion of being *ergodic* in this context only means solutions are possible and does not mean *unbiased* as, depending on the tile constraints or configuration, solution biasing may occur. Punch Out Model Synthesis (POMS), the algorithm presented in this paper, is highlighted for ease of comparison.

In terms of algorithms to ensure arc consistency, MMS and WFC have used AC3 and AC4. AC3 is easy to understand and can be performant if tile count is low but quickly suffers as tile count increases [7]. AC4 is optimal, in general, but requires large amounts of auxiliary space [8]. POMS uses AC4 exclusively as tile count is often large (1,000 or more).

CBTGs are a specialization of a more general Constraint Satisfaction Problem (CSP). Karth and Smith offer some history of CSPs, some common concepts and algorithms in [9, 10]. Of note is Karth and Smith's observation that shallow backtracking does little to help resolve conflicts.

Two areas of research activity for CBTGs are attempts to make infinite CBTG algorithms and attempts at giving more control over created output. Kleinberg provides an algorithm for infinite WFC by chaining blocks together [11]. While this can produce large scenes, the tile constraints are conditioned so that failure probability is low and Kleinberg admits that block resolution can fail without any recourse on how to continue.

Of note is Merrell's discussion of *infallible* models [4], where the tile constraints are conditioned so as to never be able to encounter a contradiction. For infallible models, infinite block chaining is always achievable as there is no possibility of a contradiction occurring. Though infallible models represent a case that is always solvable, it's unclear how possible or how difficult it is convert tile constraints from exemplar scenes to infallible models. In particular, all tile sets considered in this paper are fallible models.

Newgas provides `Tessera`, a software project that implements WFC along with options for a variety of constraints [12]. Nie et al. provide an extension to WFC to infinite

[3] This method was suggested to the authors through personal communication with P. Merrell

grids but require the tile set to be *complete* or *sub-complete* which may be difficult for tile sets in the wild [13]. Cooper introduces `Sturgeon` that incorporates a mid level API to specify more explicit and longer range constraints as an addition to WFC [14]. Though Cooper's `Sturgeon` program and ideas look promising, the sizes involved are relatively small (40x40 and below) and it's unclear how well the constraints would work on various tile sets, initial conditions or how well the method would scale for larger level sizes.

Alaka and Bidarra attempt to offer more control over output level design by considering a user interface to group tiles and weight individual tile probabilities into user specified regions [15]. Langendam and Bidarra attempt to offer more control over output levels through a mixed initiative graphic tool that offers the ability to interact with the underlying WFC solver in various ways [16].

Of note is Lucas and Volz's straight forward application of counting tile frequency and measuring the Kullback-Leibler divergence to attempt to get a better understanding of how biased the resulting generated map is [17]. Karth and Smith use a vector-quantized variational auto encoder (VQ-VAE) to create reduced tile domain maps which can then be input into WFC to produce novel results [9].

Automatic tile constraint creation from exemplar scenes was provided in Gumin's *Wave Function Collapse* (WFC) project, [5]. Sherratt provides a good introduction to the pre processing step for automatic tile constraint creation in [18]. Newgas offers a nice introduction to WFC and automatic tile constraint creation in [19].

3. Algorithm

Punch Out Model Synthesis (POMS) works to progressively resolve chosen blocks within a larger grid. Blocks that fully resolve are then saved back into the larger grid. Should blocks not be able to fully resolve, portions of the grid are reverted back to an indeterminate state, depending on the mode of failure.

Algorithm 1 Punch Out Model Synthesis

Input,Output: Grid G,
Input: Block Choice Scheduler, BCS,
Input: Erosion Choice Scheduler, ECS,
Input: Block Solver Algorithm, A
Set G to a fully indeterminate state
while G not fully resolved **do**
  Choose block B in G by querying BCS
  Initialize B to indeterminate state
  From G, set and pin B edges not on G boundary
  Apply initial setup restrictions to B
  Run $A(B)$ ▷ Attempt to resolve B with A
  if initial arc consistency is impossible for B **then**
    Set B region to indeterminate state in G
  else if $A(B)$ fails to find a full resolution in B **then**
    Erode boundaries in G using ECS
  else ▷ success
    Copy fully resolved tiles from B into G
  end if
end while

POMS retains a copy of the larger grid, keeping either the fully resolved tile per cell or an indicator that the cell is in an indeterminate state. Each round consists of choosing a block from some block choice scheduler and doing full block level resolution from some underlying block level algorithm. In this paper, we only consider Breakout Model Synthesis (BMS) [1] for the underlying block level resolution algorithm.

Each block chosen has its boundary pinned to the values as they appear in the grid, except if the block boundary falls on the grid boundary. Here, *pinning* means setting a cell's tile domain to a certain set of values and not allowing any constraint propagation to be performed on the cell. The pinned cell's tile domain can affect a neighboring cell tile but is otherwise not considered for constraint propagation. For the block boundary pinning, the block boundary cell's tile domain is restricted to a single tile value, taken from the tile at the grid cell. Pinning ensures that the block can integrate into the larger grid by guaranteeing the boundary conditions of the block match the related regions in the grid.

If the block and grid share a boundary, the tiles at the cell boundaries are not pinned but are subject to boundary conditions. Here, we only consider boundary conditions where a fixed tile is used when neighboring constraints would fall out of grid bounds.

Block regions are chosen by a *Block Choice Scheduler* (BCS) as referenced in Algorithm 1. From our experimentation, the BCS is problem specific and will be discussed later (Section 4).

Once a block is chosen, the block is initialized by adding the full tile domain of tile values to every block cell, applying setup restrictions and then running an initial constraint propagation that attempts to put the block in an arc consistent state. The setup restrictions only allow for tiles to be added, removed or pinned on an individual cell. Once setup restrictions are imposed, an attempt is made to run the constraint propagation and put the block in an arc consistent state.

If the block is successfully put in an arc consistent state, the block level resolution algorithm proceeds and attempts to find a fully realized block. If a fully realized block is found, it's put back into the grid and the algorithm continues on by choosing another block until the grid is fully realized.

Should the initial attempt to put the block choice into an arc consistent state fail, the block region is reverted to an indeterminate state in the grid. The motivation for reverting the whole block region, as opposed to just relying on erosion, is that if a block cannot be put into an arc consistent state, subject to its pinned boundary values, there might be a strong contradiction that requires an aggressive back-off strategy.

If the initial block level arc consistency attempt succeeded but the block level resolution algorithm failed to find a fully realized block, boundary tiles of fully resolved regions are probabilistically reverted to an indeterminate state. The process of probabilistically reverting boundary cell locations to indeterminate states is called *erosion*, with the *probability of erosion* being a tunable parameter that sets the probability that a cell located on a resolved boundary is reverted.

Algorithm 1 gives pseudo code for the Punch Out Model Synthesis (POMS) algorithm. Figure 2 gives an overview of the POMS algorithm, highlighting a single step. Figure 3 shows a slideshow of POMS being run on the *Pill Mortal* tile set for a 64x64 grid size with 32x32 block size.

The probability of erosion is managed by the *Erosion Choice Scheduler* (ECS) as referenced in Algorithm 1. We have found that a good choice for ECS is to increase the erosion probability by the number of failed attempts, allowing for a more aggressive erosion strategy should block reso-

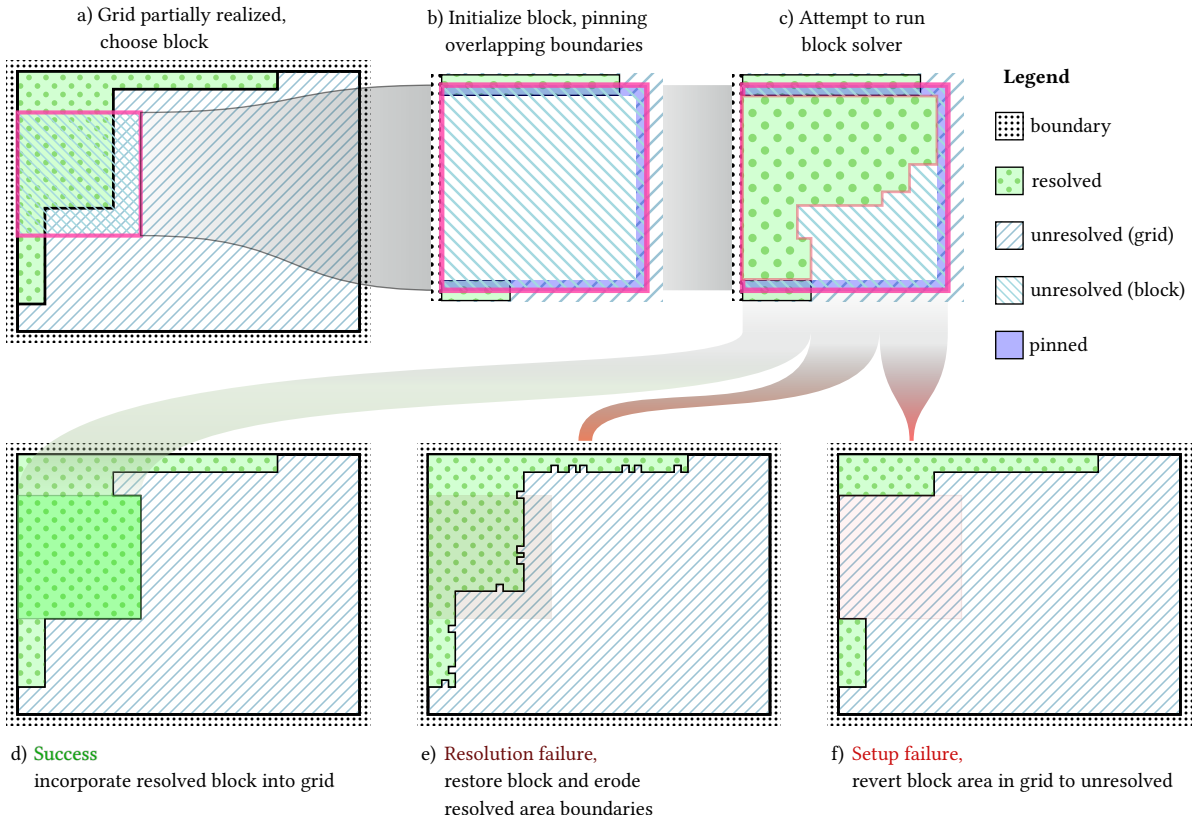

Figure 2: a) A block is chosen in the partially resolved grid, based on a block choice scheduler b) Once the block is chosen, the boundary is pinned if not on the a grid boundary and the center put into an indeterminate state. c) The block level solver attempts to find a solution for the block, with any pinned boundary restrictions d) If successful, the block is incorporated back into the grid. e) If the block solver algorithm failed to resolve, after some maximum iteration count, say, then the grid is reverted to its previous state and resolved boundaries are eroded based on an erosion choice scheduler. f) If the block solver algorithm failed to start because the block could not be put into an arc consistent state given the tiles pinned on the boundary, the block area in the grid is reverted to an indeterminate state.

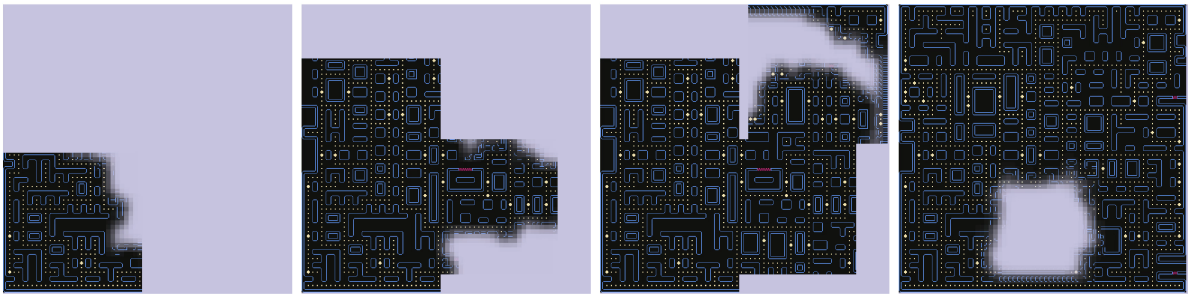

Figure 3: A slideshow of POMS run on the *Pill Mortal* tile set. The block size is 32x32 and the grid size is 64x64 with a block choice policy that chooses block centers uniformly at random from the available unresolved cell locations in the grid.

lution become increasingly difficult. Without the erosion, block level solvers could be doomed to perpetually attempt resolution on blocks with identical initial state. The erosion provides a backtracking mechanism, allowing for a change in initial state of chosen blocks and potentially removing contradictions on resolved regions in the grid.

Maximum iteration counts can be added so that POMS or the underlying block level resolution algorithm (A) don't loop forever. The iteration counts and other checks in Algorithm 1 have been omitted for brevity.

Since the grid only keeps one value per cell and full constraint propagation is done on a block level, only a block level's worth of data need be kept in active memory. Full constraint propagation is done on an individual block level but the block size can be chosen to be much smaller than the grid, allowing block sizes to be tuned as resources allow.

The indeterminate state of the larger grid also imposes minimal assumptions for setup, as there is no need for a fully realized initial state. Further, the lax assumptions on initial state allow for more freedom in choosing initial restrictions.

4. Results

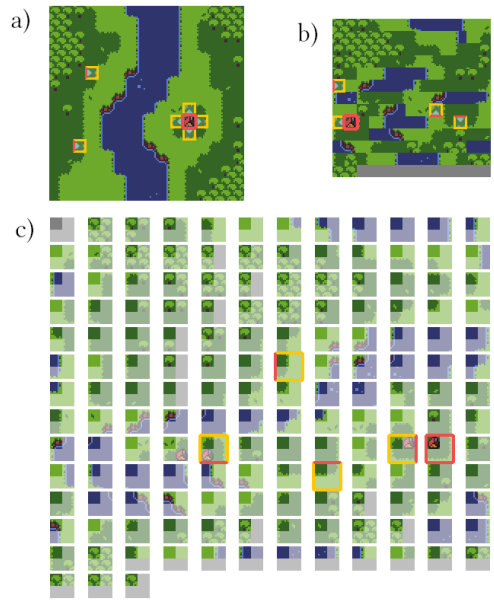

Figure 4: a) The exemplar image, take from Woźniak's Forest Micro tile set with a single tile highlighted in red with it's neighbors, as they appear in the exemplar image highlighted in orange. b) The inferred tile set, packed into an image with the highlighted tile in red and its neighbor tiles highlighted in orange. Note that even though tiles in the tile set might look the same, these represent different tiles as they have different constraints for which neighbors are permissible. c) A catalog of super tiles, with the highlighted tile in red and its neighbors highlighted in orange. Each super tile consists of a 2x2 window taken from the exemplar image (a). Neighbors are derived from the corresponding overlap of the 2x1 tile window for left and right neighbors and a 1x2 tile window for the up and down neighbors. Grey regions represent tiles that fall outside of the exemplar image.

In this section, we highlight five tile sets that represent different aspects of the benefits and pitfalls of the Punch Out Model Synthesis (POMS) algorithm.

The four 2D tile sets had their tile constraints inferred from an exemplar image using an automated tile constraint generation method. The exemplar image is split into tiles and a sliding super tile window, of a larger size than the tile itself, is moved over the exemplar image. The super tiles are deduplicated and checked for overlapping bands to other super tiles. For every matching, overlapping band, a constraint is added to the tile constraints, allowing an admissible pairing for the tiles in the appropriate dimension direction.

Figure 4 shows the exemplar image (*a*), the inferred tile set (*b*) and the complete list of super tiles for Woźniak's *Forest Micro* tile set. A single tile is highlighted in red and its neighbors are highlighted in yellow with red edges corresponding to their neighboring direction.

Figure 4 uses a tile size of 8x8 pixels, with a 2x2 super tile size. The upper left hand tile is used as the displayed representative tile, which can be seen by comparing the non dimmed tiles in the super tile list from Figure 4 (*c*) to the packed tile set image (*b*). A 2x1 overlapping band is in each direction, suitably rotated, to find which super tiles neighbor other super tiles. An interested reader can confirm that there is a 2x1 overlap to the highlighted red super tile to each of its highlighted yellow neighbors in Figure 4.

Boundary constraints need special consideration. One option is to only allow a special "zero" tile at grid boundaries, ensuring "zero" tile constraints are added for tile pairs that fall across the edge boundaries of the exemplar image. This is the option chosen for Woźniak's *Forest Micro* tile set and can be seen in the "zero" (gray) tiles present for the super tiles listed in Figure 4 (*c*). Another option is to have wrap around boundary conditions, with a sliding window that wraps around from right to left and from down to up. This is what is done for the LUNARSIGNALS' *Overhead Action RPG Overworld* tile set.

For the automatic tile constraint creation from exemplar images, some artistic input is needed in choosing tile size, window size and boundary conditions. The tile sizes, window sizes and boundary conditions for the 2D tile sets used in this paper were chosen through inspection and experimentation. An in depth explanation of automatic tile constraint creation is beyond the scope of this paper and references will be provided later (Section 4). The five tile sets we use here are summarized in Table 2.

As a simple, measurable quantity to help understand the correlation length implied by the tile constraints, we define a Tile Arc Consistent Correlation Length (TACCL), as first mentioned in Hoetzlein's `just_math` project [1]. Starting from an indeterminate grid, the center cell is resolved to a tile and constraint propagation is performed to put the grid back into an arc consistent state. The maximum distance the cells in the grid are altered is the Tile Arc Consistent Correlation Length (TACCL). Algorithm 2 provides pseudo code to determine the Tile Arc Consistent Correlation length.

Algorithm 2 Tile Arc Consistent Correlation Length

Output: Tile Arc Consistent Correlation Length L
Input: Block Size N
Create a block B of dimension N
$L = 1$
for tile every tile t **do**
  Put block B in a fully indeterminate state
  Apply initial setup restrictions to B
  Resolve center cell in B to t, make B arc consistent
  Save maximum length, d, of altered cells
  if $d > L$ **then**
    $L = d$
  end if
end for
Return L

The TACCL is meant to estimate the correlation length of an underlying tile constraint set but can be misleading as some tile sets will have a finite TACCL even though correlation lengths can be unbounded. An unbounded TACCL implies an unbounded correlation length but the converse is not true and a finite TACCL does not necessarily imply a finite correlation length. The *Brutal Plum* tile set, that has an unbounded correlation length but finite TACCL, displays this phenomenon and will be discussed later in this section.

Figure 5 and Figure 6 show the TACCL, reference images and example runs for the five tile sets listed in Table 2.

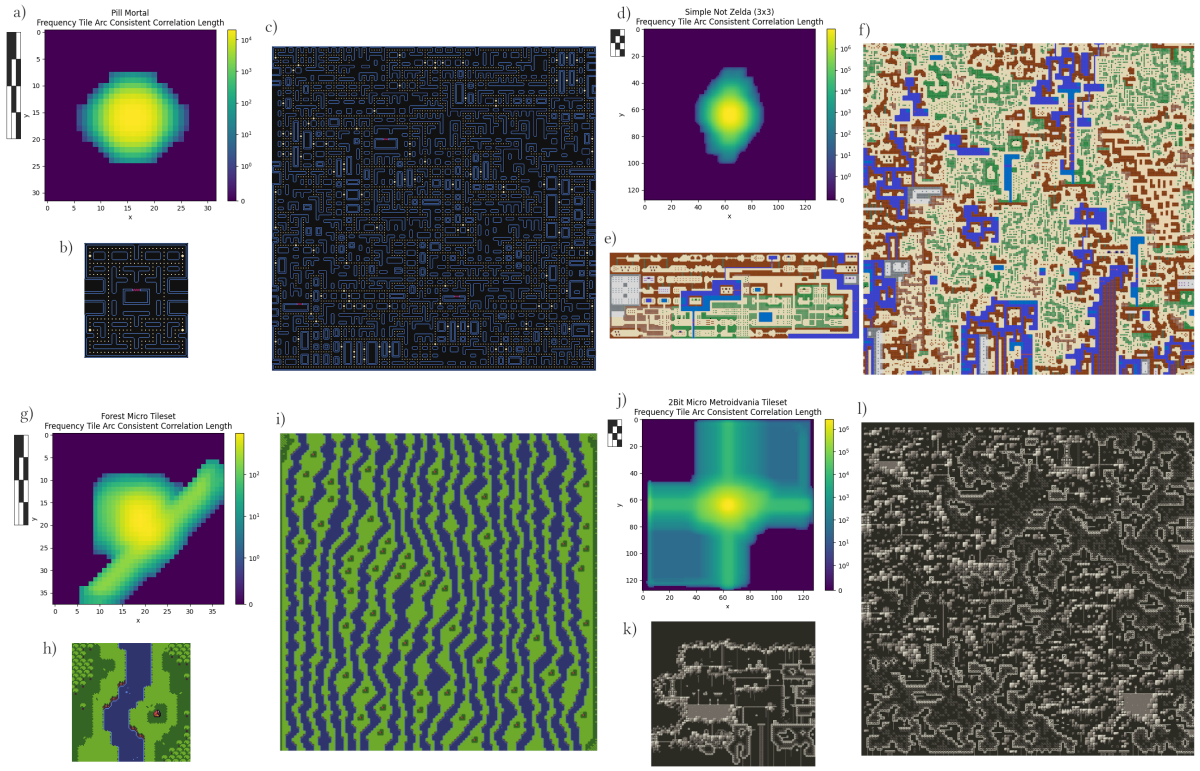

Figure 5: Tile Arc Consistent Correlation Length (TACCL) plots, source exemplar image and example output for four 2D tile sets as listed in Table 2. The TACCL, exemplar image and example 64x64 output using a block size of 8x8 for the *Pill Mortal* tile set are shown in a), b) and c) respectively. The TACCL, exemplar image and an example 256x256 output using a block size of 50x70 for LUNARSIGNALS' *Overhead Action RPG Overworld* are shown in d), e) and f) respectively. The TACCL, exemplar image and an example 128x128 output using a block size of 48x48 for Woźniak's *Forest Micro* tile set are shown in g), h) and i) respectively. The TACCL, exemplar image and an example 128x128 output using a block size of 48x48 for 0x72's *Two Bit Micro Metroidvania* tile set are shown in j), k), l) respectively.

Table 2

Tile Set	Tile Count	Dim.	Tile Arc Consistent Correlation Length (x/y)
Pill Mortal	190	2D	24
Overhead Action RPG Overworld [20]	3031	2D	50/70
Forest Micro [21]	159	2D	Unbounded
Two Bit Micro Metroidvania [22]	1807	2D	Unbounded
Brutal Plum	90	3D	16

4.1. Bounded Tile Arc Consistent Correlation Length (TACCL)

The first two tile sets that appear in Figure 5, *Pill Mortal* and LUNARSIGNALS' *Overhead Action RPG Overworld* (OARPGO) tile set, both have bounded Tile Arc Consistent Correlation lengths (TACCL).

The *Pill Mortal* tile set was generated from the exemplar image shown (Figure 5, label *a*), with a tile window of 2x2 and each tile consisting of 8x8 pixels. A 2x1 overlapping tile region was used to create the tile constraint set with boundary conditions implied by the exemplar image. When running the Punch Out Model Synthesis (POMS) algorithm, a special "zero" tile, with placement constraints allowing it to neighbor boundary tiles, is virtually placed outside of any admissible region and removed from the interior admissible region.

The generated realization in Figure 5 (label *c*) was created from a POMS run with block size 32x32 on a 128x128 grid. A uniform weighting for all 190 tiles was used. The block choice scheduler considered all unresolved tiles in the grid uniformly as candidates for the center of the block.

LUNARSIGNALS' OARPGO tile set was generated from the exemplar image show using a 3x3 tile window, with each tile consisting of 16x16 pixels. A 3x2 overlapping tile region was used to create the tile constraint set with wrap around boundary conditions. The tile set consists of 3031 tiles and a uniform tile weighting was chosen. In this case, a 3x3 window was chosen as a smaller 2x2 window was observed to not give good aesthetic results.

When generating outputs using the OARPGO tile set, the outer frame of the grid is pinned to an unresolved state. This is necessary as the tile constraint set was created with wrap around conditions and has no valid constraints that match the "zero" tile to the rest of the tile domain.

The example 256x256 output (Figure 5, label *f*) was created with a block size of 50x70. A block choice strategy was used that preferentially chose block centers from unresolved grid cells nearer to the center. This has the effect of resolving regions from the "center out", never creating isolated regions that need to be joined and effectively ensuring a

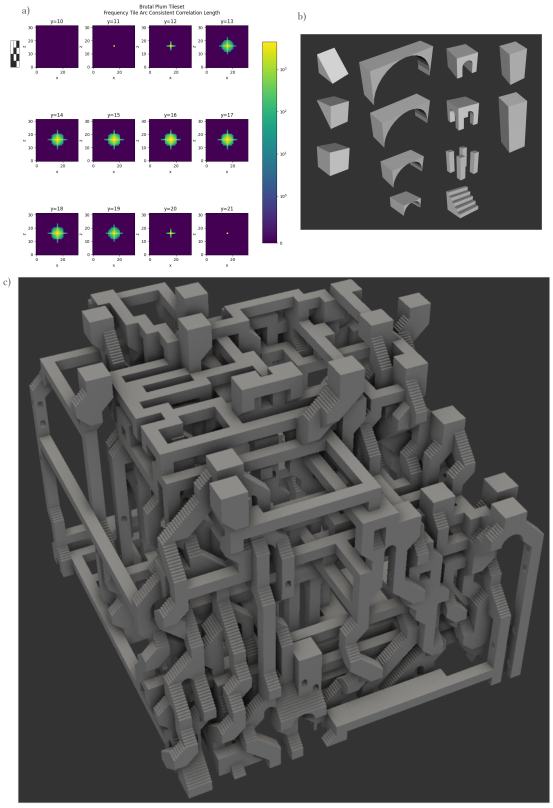

Figure 6: The Tile Arc Consistent Correlation Length (TACCL) plot, source tile set and an example 32x32x32 output for the 3d *Brutal Plum* tile set listed in a), b) and c) respectively. A block size of 22x22x22 was used.

large contiguous region during the course of the algorithm. From observation, other block choice strategies were not as successful as choosing block centers with a grid center bias.

It should be noted that without wrap around boundary conditions, the OARPGO tile set would have unbounded TACCL. Further, without wrap around boundary conditions POMS has trouble finding solutions as the grid boundary region is non trivial for this tile set. The success of the grid center biased block choice strategy, the failure of other block choice strategies and the unbounded TACCL of non wrap around boundary conditions, could suggest that the bounded TACCL does not capture some longer range or unbounded constraints embedded within the wrap around OARPGO tile set constraints.

Of note is the "stuttering" effect that happens with many features being repeated in a linear direction. For example, there are long regions of vertical rocks surrounded by water tiles that continue on downward until the pinned boundary is hit. This effect becomes even more pronounced when other tile weightings are used. We suspect this is because of a certain pattern preference that then gets re-enforced by a surrounding structure. We make note of this effect but otherwise don't investigate further in this paper.

4.2. Unbounded Tile Arc Consistent Correlation Length (TACCL)

The last two tile sets in Figure 5, Woźniak's *Forest Micro* tile set and 0x72's *Two Bit Micro Metroidvania* tile set, both have unbounded Tile Arc Consistent Correlation Lengths (TACCL).

The *Forest Micro* tile set was generated from the example image shown with a 16x16 pixel tile size and a 2x2 tile window. A 2x1 overlapping tile region was used to create the tile constraint set with boundary conditions implied by the exemplar image. A "zero" tile was added to neighbor boundary tiles and the "zero" tile was removed from the admissible region of the grid and placed virtually outside of the grid boundaries.

The implied constraints derived from the *Forest Micro* tile set exemplar image (Figure 5, label *h*) consist of 159 tiles. The *Forest Micro* tile set has an implied global constraint that the river count from the top row of the realized grid must match the river count on the bottom row. This global constraint is not explicitly present or specified but is a by-product of the local constraints.

For large grid sizes, POMS fails to find realizations for the *Forest Micro* tile set when a block choice policy is chosen that weights unresolved grid positions equally. Under these conditions, POMS effectively makes a random choice for the number of rivers on the top and bottom row. One can expect, with enough random sampling, the river count to be identical, but the problem becomes exponentially less likely as grid size grows.

To guide POMS in finding solutions for the *Forest Micro* tile set, a block choice policy was chosen that preferentially weights cell positions starting from the upper left and decreases as cells are considered going down and to the right. The diagonal weighting has the effect of keeping a growing contiguous region that locally keeps the top and bottom river counts the same. Contradictions that do occur tend to be localized and their resolution keeps the river counts the same

Though the *Forest Micro* tile set has an unbounded correlation length, the global constraint is weak enough to be overcome by a simple "weighted diagonal" heuristic. Figure 5 label *i* shows an example run for a 128x128 grid with a block size of 48x48, using the "weighted diagonal" block choice scheduling strategy heuristic.

The tile constraint set for 0x72's *Two Bit Micro Metroidvania* (2BMMV) tile set was generated from the exemplar image shown (Figure 5, label *k*) with a 24x24 pixel tile size and a 2x2 window tile size. A 2x1 overlapping region to determine neighboring constraints was used resulting in a tile set size of 1807. A "zero" tile was added as neighbors to the boundary tiles, removed from the admissible region of the grid and placed virtually outside of the grid boundaries.

As can be seen by Figure 5 (label *j*), the 2BMMV tile set has unbounded correlation length, but the constraint is weak enough to be overcome by running POMS with a block size of 48x48 and a block choice scheduler that chooses block centers from unresolved grid cell positions uniformly. The unbounded correlation length comes from the boundary restrictions which might disappear if care were taken to create wrap around tile constraints.

The 3D *Brutal Plum* tile set was programmatically generated from a set of 20 unique tiles that, after rotations and deduplication, grows to 90 tiles. Many of the tiles are aesthetically identical but are logically different to embed desired features, such as requiring certain logical tiles to be above or below other tiles. In particular, non empty tiles must have a non empty path to the ground base plane, giving an implied global constraint. The global constraint is not captured by the Tile Arc Consistent Correlation Length (TACCL) and

highlights the failure of the TACCL heuristic to properly capture an unbounded correlation length embedded in the tile set.

Though the correlation length for the *Brutal Plum* tile constraints is unbounded, POMS still is able to reliably find realizations with block size 22x22x22. For aesthetic reasons, a tile weighting that increases the likelihood of the empty tile, the arch tiles and stair tiles was chosen. The reader is referred to the reference implementation [4] for further details.

5. Conclusion

Punch Out Model Synthesis (POMS) provides an algorithm for the Constraint Based Tiling Generation problem. We have shown that POMS can discover realizations from tile constraints that have finite correlation length. We have also shown that POMS is able to find realizations for some problems that have weak implied global constraints. If the tile set and configuration are not conditioned well, POMS may fail to find a solution or provide biased results.

Tile constraints that have unbounded or long range correlations are more difficult and sometimes intractable. Constraint Based Tiling Generation problems problems are, in general, NP-Complete, so there is likely no comprehensive strategy that leads to efficient methods of solution but the worst case complexity results sometimes obscure when problems are readily solvable. For many tile constraints that are NP-Complete, the general complexity result might not apply to some generic configuration, allowing some ensembles of tile constraint problems to be easily solvable.

For many real world Constraint Based Tiling Generation problems, we still lack understanding of the interplay between how difficult it is to find realizations given tile constraints and initial configuration. The Tile Arc Consistent Correlation Length (TACCL) is one heuristic measure that attempts to quantify how difficult tile constraints are to resolve. The TACCL has the benefit of being easy to calculate but its interpretation is easily confounded, so should be considered a coarse measure with limited applicability.

A libre/free reference implementation for Punch Out Model Synthesis (POMS) has been developed and can be downloaded from its repository [4].

References

[1] R. Hoetzlein, just_math, 2023. URL: https://github.com/ramakarl/just_math/tree/3ff9d10c3d170855cef1db970c0278b15e91fa13/math_belief_prop.

[2] A. Newgas, Constraint-based tile generators, 2021. URL: https://www.boristhebrave.com/2021/10/31/constraint-based-tile-generators/.

[3] P. Merrell, Example-based model synthesis, in: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, I3D '07, Association for Computing Machinery, New York, NY, USA, 2007, p. 105–112. URL: https://doi.org/10.1145/1230100.1230119. doi:10.1145/1230100.1230119.

[4] P. Merrell, Model Synthesis, Ph.D. thesis, Chapel Hill, NC, USA, 2009.

[5] M. Gumin, Wave function collapse algorithm, 2016. URL: https://github.com/mxgmn/WaveFunctionCollapse.

[6] P. Merrell, Comparing model synthesis and wave function collapse, 2021. URL: https://paulmerrell.org/wp-content/uploads/2021/07/comparison.pdf.

[7] R. J. Wallace, Why ac-3 is almost always better than ac4 for establishing arc consistency in csps, in: International Joint Conference on Artificial Intelligence, 1993. URL: https://api.semanticscholar.org/CorpusID:17765393.

[8] R. Mohr, T. C. Henderson, Arc and path consistency revisited, Artificial Intelligence 28 (1986) 225–233. URL: https://www.sciencedirect.com/science/article/pii/0004370286900834. doi:https://doi.org/10.1016/0004-3702(86)90083-4.

[9] I. Karth, A. M. Smith, Wavefunctioncollapse is constraint solving in the wild, in: Proceedings of the 12th International Conference on the Foundations of Digital Games, FDG '17, Association for Computing Machinery, New York, NY, USA, 2017. URL: https://doi.org/10.1145/3102071.3110566. doi:10.1145/3102071.3110566.

[10] I. Karth, A. M. Smith, Wavefunctioncollapse: Content generation via constraint solving and machine learning, IEEE Transactions on Games 14 (2022) 364–376. doi:10.1109/TG.2021.3076368.

[11] M. Kleineberg, Infinite procedurally generated city with the wave function collapse algorithm, 2019. URL: https://marian42.de/article/wfc/.

[12] A. Newgas, Tessera: A practical system for extended wavefunctioncollapse, in: Proceedings of the 16th International Conference on the Foundations of Digital Games, FDG '21, Association for Computing Machinery, New York, NY, USA, 2021. URL: https://doi.org/10.1145/3472538.3472605. doi:10.1145/3472538.3472605.

[13] Y. Nie, S. Zheng, Z. Zhuang, X. Song, Extend wave function collapse algorithm to large-scale content generation, in: 2023 IEEE Conference on Games (CoG), 2023, pp. 1–8. doi:10.1109/CoG57401.2023.10333214.

[14] S. Cooper, Sturgeon: Tile-based procedural level generation via learned and designed constraints, Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 18 (2022) 26–36. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/21944. doi:10.1609/aiide.v18i1.21944.

[15] S. Alaka, R. Bidarra, Hierarchical semantic wave function collapse, in: Proceedings of the 18th International Conference on the Foundations of Digital Games, FDG '23, Association for Computing Machinery, New York, NY, USA, 2023. URL: https://doi.org/10.1145/3582437.3587209. doi:10.1145/3582437.3587209.

[16] T. S. L. Langendam, R. Bidarra, miwfc - designer empowerment through mixed-initiative wave function collapse, in: Proceedings of the 17th International Conference on the Foundations of Digital Games, FDG '22, Association for Computing Machinery, New York, NY, USA, 2022. URL: https://doi.org/10.1145/3555858.3563266. doi:10.1145/3555858.3563266.

[17] S. M. Lucas, V. Volz, Tile pattern kl-divergence for analysing and evolving game levels, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, Association for Computing

[4] https://github.com/zzyzek/PunchOutModelSynthesis

Machinery, New York, NY, USA, 2019, p. 170–178. URL: https://doi.org/10.1145/3321707.3321781. doi:10.1145/3321707.3321781.

[18] S. Sherratt, Procedural generation with wave function collapse, 2019. URL: https://www.gridbugs.org/wave-function-collapse/.

[19] A. Newgas, Wave function collapse explained, 2021. URL: https://www.boristhebrave.com/2020/04/13/wave-function-collapse-explained/.

[20] LUNARSIGNALS, Overhead action rpg overworld, 2016. URL: https://opengameart.org/content/overhead-action-rpg-overworld.

[21] K. Woźniak, Micro tileset - overworld and dungeon, 2015. URL: https://opengameart.org/content/micro-tileset-overworld-and-dungeon.

[22] R. Norenberg, 2bit micro metroidvania tileset, 2024. URL: https://0x72.itch.io/2bitmicrometroidvaniatileset.

A. Appendix: Breakout Model Synthesis (BMS)

Algorithm 3 Breakout Model Synthesis

Input,Output: Block B,
Input: Setup restrictions S,
Input: Tile constraints C,
Input: Tile Choice Scheduler TCS
Input: Soften Choice Scheduler SCS
Put B into a fully indeterminate state
Apply setup restrictions S to B
Apply tile constraints C until B is arc consistent
if unable to create initial arc consistent state **then**
    return failure
end if
Save copy of B to P
while B not fully resolved **do**
    $B' = B$
    Choose tile and cell to resolve in B from TCS
    Propagate resolution until B is arc consistent
    if contradiction encountered **then**
        Revert B back to B'
        Query SCS for a sub-region, R, to soften
        Revert region R in B back to P
        Constraint propagate until B is arc consistent
    end if
    if iteration count has been exceeded **then**
        return failure
    end if
end while
return success