Tessera: A Practical System for Extended WaveFunctionCollapse

Adam Newgas
adam2020@newgas.net

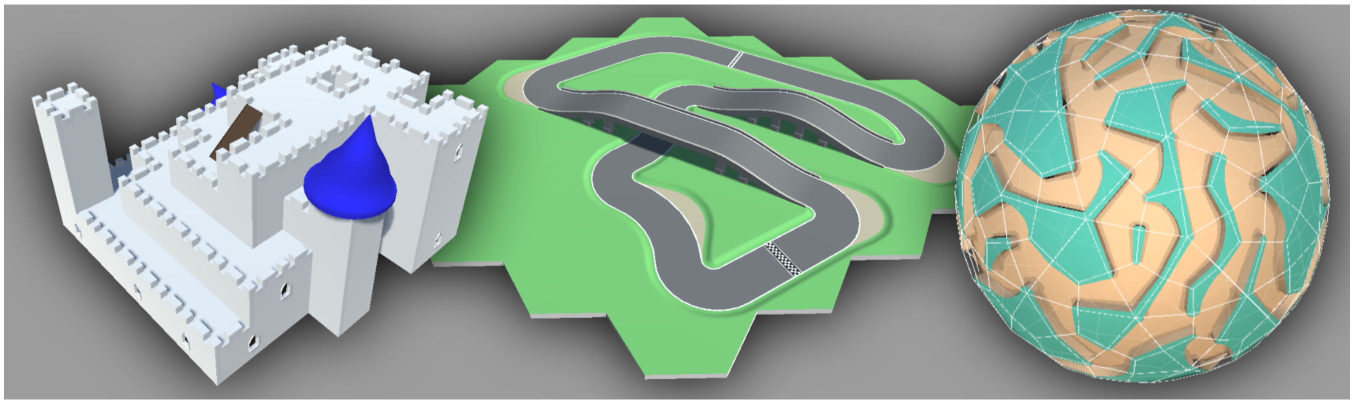


Figure 1: Various example outputs of Tessera

ABSTRACT

Constraint-based procedural generation has recently had a lot of interest following the publication of the WaveFunctionCollapse (WFC) algorithm, but usability issues have restricted the number of games and projects that have resulted. We present *Tessera*, a library and tool for WFC specifically designed to address the practical issues that arise from constraint based generation. We discuss the user interface of the tool and useful extensions to the base algorithm of WFC.

CCS CONCEPTS

• **Theory of computation** → **Constraint and logic programming**; • **Mathematics of computing** → *Solvers*; • **Applied computing** → **Computer games**; • **Computing methodologies** → Spatial and physical reasoning; • **Human-centered computing** → Graphical user interfaces.

KEYWORDS

procedural content generation, wavefunctioncollapse, constraint solving

ACM Reference Format:
Adam Newgas. 2021. *Tessera*: A Practical System for Extended WaveFunctionCollapse. In *The 16th International Conference on the Foundations of Digital Games (FDG) 2021 (FDG'21), August 3–6, 2021, Montreal, QC, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472538.3472605

1 INTRODUCTION

Procedural Content Generation (PCG) is the study of generative methods producing assets for games and other media. One approach is search-based PCG[33], where a large space of possible outputs is filtered by criteria specifying what is desirable. Criteria are commonly specified as a series of hard constraints, which can be done declaratively with Constraint Programming languages. Algorithms for efficiently searching the space given these constraints are called Constraint Solvers.

WaveFunctionCollapse (WFC)[11] is a recent constraint-based algorithm for procedural generation. The key idea is an extension of standard constraint solvers with a "minimal entropy heuristic" that randomly directs the solver's search in a way that follows a user-specified frequency distribution without compromising the efficiency of the search procedure.

WFC was originally conceived as a texture synthesis algorithm: the search space is a rectangular grid of pixels, and the constraints and frequencies are derived from an input texture, called a sample. There are two modes: simple, that learns constraints regarding pairs of adjacent pixels; and overlapping, which learns constraints about small patches of pixels.

When first published, WFC used the Arc Consistency 3 algorithm[20] as the solver, and had no support for backtracking. These choices position WFC as a deliberate simplification of Constraint Programming and Constraint Solving.

Since publication, the range of WFC usage has broadened. For game level generation purposes, WFC is often used to generate a grid of tiles instead a grid of pixels, in 2d or in 3d. Constraints on nearby tiles can be learnt from a sample level or specified by other means. Adding backtracking or using a different constraint solver are also common modifications.

WFC quickly met with a lot of interest[13], due to being simple to implement and understand while producing high quality results without a complex set of configuration. Another plus is that

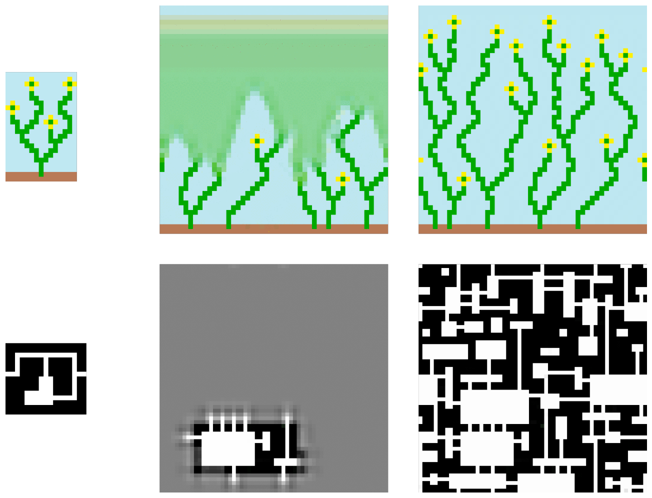


Figure 2: Two examples of running WFC as texture synthesis in overlapped mode. The left column is the original sample, the middle column is midway through processing (showing unresolved tiles with their average color), and the right column is the finished output. ©Maxim Gumin. Used with permission

constraint-based generators give qualitatively different output to classic constructive approaches. WFC's popularity eventually lead to several commercial games developed using it for tile-based level generation[6, 9, 31].

However, none of these games use WFC unmodified. That is because the algorithm has several shortcomings that make it hard to take beyond its roots in texture synthesis, including:

- Local similarity - whether in simple or overlapped mode, WFC constraints only involve nearby tiles. This tends to give the output a homogenous look that is suitable for textures, but looks bland and uninteresting when applied to level design.
- Inconvenience of sample-based input - specifying a sample as the only major input is extremely convenient at getting a prototype working immediately, but it scales poorly to more complex examples. As more tiles are needed, or multiple rotations, the amount of sample material required to cover combinations grows exponentially.
- Difficulty of control - for games, it's typically necessary to control procedural content to fit the requirements of the design. That means having ways of parameterizing the generation, or integrating it with content authored in a different way.
- Weakness of Arc Consistency 3 Algorithm - this algorithm is simple to implement, but is slower and less reliable than more modern constraint solvers.

1.1 Our contributions

In this paper, we present *Tessera*, a tool for integrating WFC into Unity3D games that addresses these issues. Tessera supports a wide variety of controls and options that make it suitable for many sorts of level generation. A few example are shown in Figure 1.

We propose a new paint-based approach to declare tile adjacencies that offers advantages over sample-based input and other available tools. It is easy to use, and integrates naturally with other features of Tessera.

Tessera also includes several extensions to WFC intended to increase the range of possibilities and the ability to control WFC to achieve desired results. In particular, we introduce a class of global constraints to supplement the local only behaviour of WFC. Constraints of this kind have not been applied to WFC before, and our approach offers performance advantages over similar work in Answer Set Programming (described below).

2 RELATED WORK

2.1 Research

There have been numerous investigations into using constraint solving techniques for PCG. It is appropriate for puzzle, narrative and level design by appropriate choice of constraints. We will focus on related level generation work.

Much of the level-based research is focused on a form of constraint problems called Answer Set Programming (ASP), using a declarative syntax such as AnsProlog. *ASP for PCG*[23] gives a general introduction to this approach and demonstrates generating a "chromatic maze" which follows certain generation rules while always having a path from start to end of a given length. *ASP with applications to mazes and levels*[21] describes how path constraints similar to Teserra's can be implemented as ASP constraints. There are numerous other explorations of ASP level generation[1, 5, 24, 25, 27].

Tanagra[26] is a mixed initiative tool that uses the Choco[12] constraint solver to modify user authored levels so that they are always completable, by setting various linear constraints. Whitehead[36] discusses using the Z3[7] constraint solver to solve placing square rooms satisfying adjacency and overlapping constraints, also by setting linear constraints. Choco and Z3 are SMT solvers - a different class from ASP based solvers[36].

WFC itself is described extensively in *WaveFunctionCollapse is Constraint Solving in the Wild*[13]. Then *Addressing the Fundamental Tension of PCGML with Discriminative Learning*[14] discusses some of the possibilities and difficulties of controlling the output of WFC in mixed-initiative design. *Automatic Generation of Game Content using a Graph-based Wave Function Collapse Algorithm*[15] extends WFC to operate on graphs.

2.2 WFC in Games

WFC is too recent to have seen large commercial uptake, but it has been used in a few games. They use various techniques and extensions to make WFC practical for real world design.

Caves of Qud[9] is a 2d roguelike game that uses WFC in overlapping mode to generate parts of levels. It runs WFC separately in different sub areas to avoid the homogeneity problem mentioned above[4]. It avoids dealing with rotation by using a simplified tileset, which is post-processed for the final map.

Bad North[6] is a Viking based strategy game that uses WFC to generate small 3d islands for the player to defend. There are over 400 tiles, some spanning over multiple cells, so the generation pre-selects a subset of them for each island to give some visual consistency. A heuristic is applied to ensure the entire level is navigable[29].

Townscaper[31] is a mixed initiative a town planning toy by the same author. Users click on the cells of a 3d grid to add / remove from the town and a WFC variant sets appropriate tiles nearby to make this so. It features an irregular, infinite grid of degree 4 instead of WFC's usual square grid.

2.3 Similar commercial tools

There are numerous library implementations of WFC, but only a few of them have focused on exploring how to easily configure or control the process. They are mostly ports of the original implementation[11] and behave similarly. That implementation allows configuration either by giving a sample input, or via an XML configuration file. The XML configuration contains flat adjacency lists, and specifies the symmetry of each tile, which is used to generate additional adjacencies by rotation of the supplied set. This configuration format has not proven popular, which illustrates the difficulty of improving on the sample-based approach.

Generate Worlds[8] is a tool for assembling voxel based tiles into procedurally generated levels. It does not use WFC, but does use a similar constraint-based algorithm that propagates information a fixed distance (rather than recursively, as is done in Arc Consistency 3). Tile adjacency information is inferred from the voxels on the boundary of each tile, meaning new tiles can be added without worrying about how they combine with existing ones.

Tile Composer[28] is a commercial produce for running WFC in Unity. Configuration of adjacencies is done via "connectors". A set is associated with each face of each tile, and adjacencies are determined by matching items from opposing sets. It also supports using the Z3[7] constraint solver in place of WFC, in which case it supports a few additional constraints such as fixing the count of the number of tiles.

3 TESSERA OVERVIEW

Tessera is divided into two components. *DeBroglie*[18] is an open-source C# library handling the computation of WFC, and the *Tessera Pro*[19] package that handles integration into the Unity game engine, UI, and layering in additional features. DeBroglie and Tessera Pro were developed simultaneously to work together, so we will not always distinguish which component is responsible for each feature discussed below[1].

DeBroglie is a modern WaveFunctionCollapse generator under a MIT license. It mostly operates in simple mode, although overlapped mode is supported via a transformation called "dual graph constraint binarization"[2]. DeBroglie supports uses a custom implementation of the Arc Consistency 4 Algorithm[17] with support for backtracking and user defined constraints[2]. Arc Consistency 4 usually has superior performance to 3, but isn't fundamentally more powerful as it lacks innovations of state of the art constraint solvers[3].

DeBroglie relies on data-driven configuration - the tiles are just opaque values, basic constraints are specified as lists of tuples and the grid is defined in terms of a graph structure with adjacencies and edge labels. More advanced features are supported via user-specified callbacks that are invoked during generation. The library is therefore very flexible, but can be hard to configure.

Tessera Pro is a layer on top of DeBroglie that makes the system friendly to end users. It is opinionated - it specifically works inside of the Unity game engine and editor, supports the simple mode of WFC only, and has a specific data model that it is designed to work with. Tessera's UI is integrated into Unity's Editor, and it includes a run-time component when building a game.

4 KEY FEATURES

4.1 Paint-based tile annotation

Tessera does not use a sample-based input, and instead features a "painting" system for specifying tile adjacencies.

When editing a cube shaped tile, a bounding cube is displayed, with each face is subdivided into a 3×3 grid in which each sub-face is can be independently painted by choosing colors from palette of colors (Figure 3). Each color acts as a separate label, and can have the display color and tool-tip customized. 2d tiles are also supported, just using edges instead of faces and subdividing into three instead of 9. Triangular and Hexagon Prism tiles are also supported, using a different set of faces and slightly modified subdivision scheme. We'll refer only to the 3d cube case for consistency, and note that all Tessera features extend to the other cases.

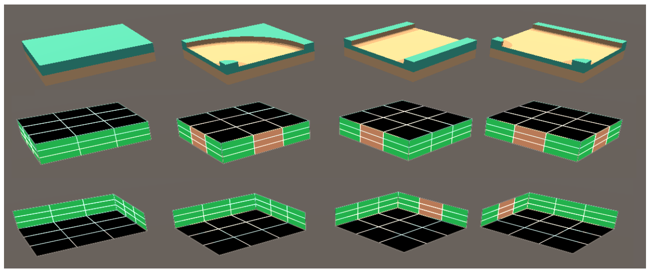


Figure 3: Top row, several example tiles of a brown path over green grass. Middle row, the sub-faces of those tiles, painted green/brown to indicate connections and symmetry. Bottom row, the same painted tiles, showing only backfaces. As these tiles will only be used in a 2d grid, there's no need to paint the top/bottom in this case.

Tessera uses the painted colors to construct adjacency constraints. To determine if two tiles can be placed adjacently, they are considered side by side in the orientation in question, and the sub-faces that face each other are paired up. By default, two tiles can be placed adjacent if and only if all nine sub-face pairs have matching colors (or are unpainted). In other words, two faces can be placed together if their color patterns are mirror images. This provides an intuitive

[1]Both components have documentation online which gives further information.

[2]A configuration option enables Arc Consistency 3 and a solver similar to Generate Worlds', but they are rarely useful.

[3]Specifically, more modern solvers are often based on DPLL(T)[10] or feature Conflict Driven Clause Learning[22]

way of describing adjacencies similar to Wang tiles[35], a system of adjacency constraints which specifies a single color per face.

Per-face label schemes are superior to sample-based inputs or adjacency lists as they are much easier to extend to larger tile sets without needing a combinatorial explosion of inputs. Users can evaluate at-a-glance if two tiles can connect, and design each tile in isolation.

Using colors per sub-face instead of a single color per face has several advantages:

- The system is a superset of the features needed for many popular labelling schemes, allowing a wide variety of tilesets to be straightforwardly annotated. Wang tiles, marching cubes, the blob pattern[34] are all supported.
- Tessera tiles may be configured to be usable in multiple rotations and reflections. The sub-faces are also rotated / reflected before the matching process, so adjacencies can be correctly inferred between tiles of different rotations. This system easily capture relationships such as "tile A, rotated by 90 degrees on the x-axis, can connect along the x-axis to tile B" which other systems struggle to describe[4].
- Drawing small patterns on each face can serve as a better mnemonic than assigning a unique color to every possibility.
- Additional information can be packed into the same format. For example, path constraints (described below) read the color of the central sub-face to infer path information. The mirror constraint infers tile symmetries based on the symmetry of the painted pattern.

If further customization is needed, the users can manually specify which colors may placed adjacent to each other, instead of requiring an exact match (Figure 5). This feature gives enough expressiveness to match using adjacency lists. The main limitation of the painting system is that it does not extend to running WFC in overlapped mode.

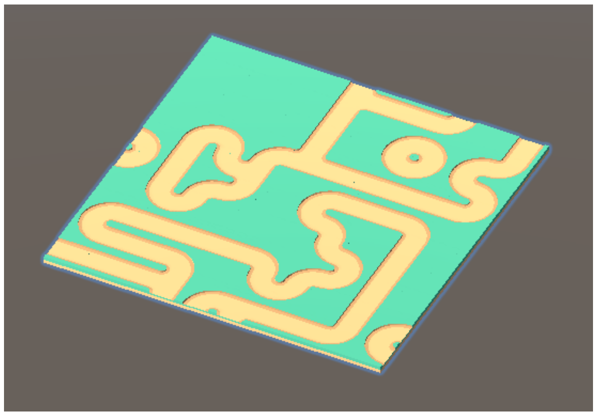
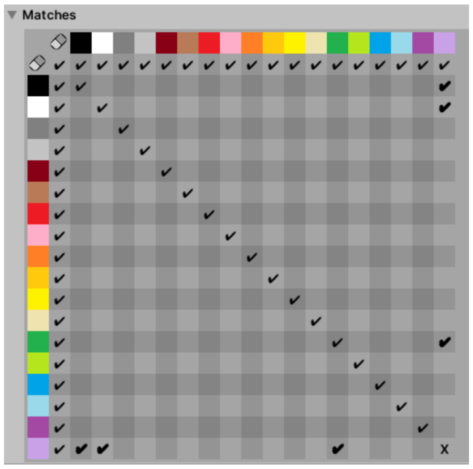


Figure 4: A generated 10×10 grid using the tiles painted in Figure 3, with rotations.

[4]Note this is mostly important in 3d, where if you have two cube tiles with free rotation and reflection, there are 8 ways a given pair of faces can align. In 2d, there's only 2 ways for edges to do so. That said, painting sub-faces is not sufficiently flexible to capture all possible symmetry subgroups of a square face. But it does do all the common ones.


Matches

	◇	⬛	⬜	▩	▨	🟥	🟫	🟥	🟪	🟧	🟧	🟨	🟨	🟩	🟩	🟦	🟦	🟪	🟪
◇	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
⬛	✓	✓																	✔
⬜	✓		✓																✔
▩	✓			✓															
▨	✓				✓														
🟥	✓					✓													
🟫	✓						✓												
🟥	✓							✓											
🟪	✓								✓										
🟧	✓									✓									
🟧	✓										✓								
🟨	✓											✓							
🟨	✓												✓						
🟩	✓													✓					✔
🟩	✓														✓				
🟦	✓															✓			
🟦	✓																✓		
🟪	✓																	✓	
🟪	✓	✔	✔											✔					X

Figure 5: UI configuring possible color matches, with some deviations from the default for the last color, lilac.

While this painting system shares some similarities with Generate Worlds[8] and Tiled's Mixed-set Terrains[16], Tessera's system is easier to use and offers a great deal more flexibility.

4.2 Scene integration and multi-pass support

Before running, Tessera scans the Unity scene for any relevant objects, such as pre-placed tiles or collider volumes with a Tessera specific annotation. These objects are translated into additional constraints before WFC is run. This feature allows users to author part of a scene manually, or with an unrelated procedural generator, and then use WFC to fill in the remaining details (see Figure 6).

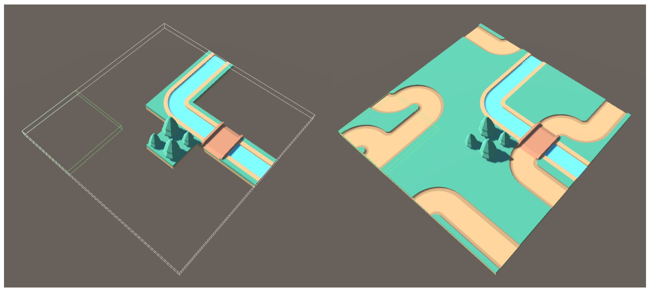


Figure 6: Left, a scene with a some user specified objects: some pre-placed custom tiles and a volume that restricts the selection of tiles inside to exclude grass tiles. Right, the same scene after running Tessera, which has respected the user's constraints.

A consequence of this feature is that Tessera naturally supports multi-pass generation. Tile generated by one generator will be recognized by another, so that one run of WFC inserts a set of tiles into the scene, and then a second run interprets those tiles as constraints for a different generator[5]. For example a user could

[5]Tiles can be configured to become another Unity object when generated, so subsequent passes don't have to re-use the same object configuration.

generate a landscape with several biomes, then in another pass generate vegetation that is consistent with the landscape. It can also be used to run different generators in different zones of the same level, similar to the *Caves of Qud* behaviour described in Related Work (Section 2).

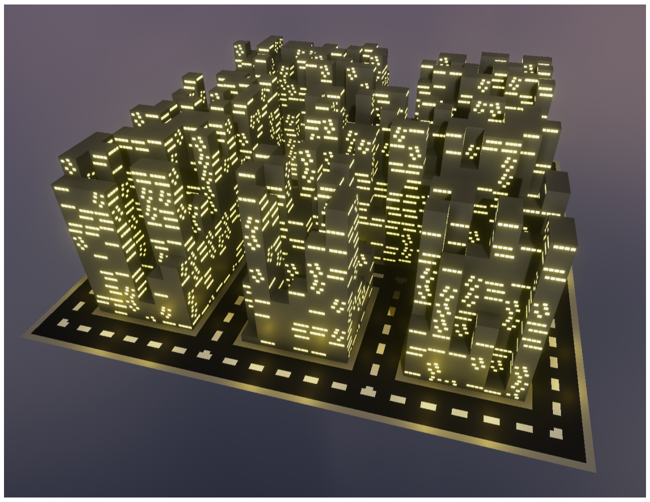


Figure 7: A multi-pass example. The street plan was generated with one set of tiles, then another pass fills the blank areas with skyscrapers, using a 3d grid with a smaller tile size.

4.3 Multi tile modules

Tessera supports using Unity objects that are large enough to cover multiple cells in the grid. These are called "big tiles", or modules. They are supported by internally subdividing the object into multiple tiles, then creating additional adjacency rules that the ensure the smaller tiles must connect to each other to form the full module. Then constraints are applied to the grid boundary to ensure that these tiles always appear together as a complete set.

The tile painting system naturally extends to these modules by simply giving the module a larger set of faces, and associating each face with one of the subdivided tiles (Figure 8).

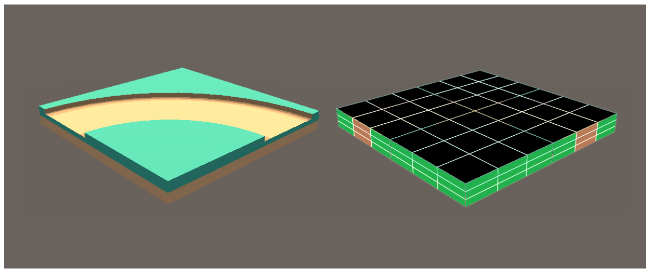


Figure 8: A module covering the space of $2 \times 1 \times 2$ cells, and its corresponding painting. It needs 16 faces (144 sub-faces) to cover the external surface area. Note how the paint on this is designed to align with the tiles from Figure 3.

This feature mirrors similar behaviour seen in Bad North, which uses modules to create smoother transitions and larger set pieces that aren't restricted to the size of a single cell[30].

4.4 Grid support

The constraints of simple model WFC are only from one adjacent cell to another. Thus, the algorithm can be described as running on a graph of nodes with one edge per adjacency[15]. Tessera uses this to support multiple different "grid types". Tessera supports regular square, triangular and hexagonal grids, and also irregular 3d grids based on the surfaces of quad or triangle meshes. Currently the system is limited to grids with a fixed number of neighbours per cell, so cannot do arbitrary graphs.

For irregular grids, a trilinear transformation is applied to each tile to conform it to the boundary of the cell and another transform aligns the normals between tiles (Figure 9). This allows users to design cube tiles and have them work on all sorts of grid shapes.

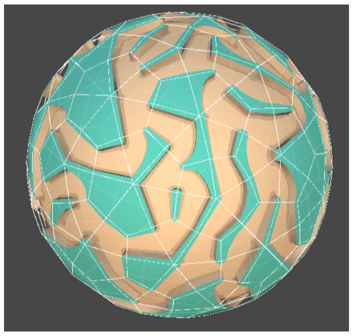


Figure 9: A generated level using a quad mesh of a sphere to define the grid. The mesh is highlighted in white.

Features like scene integration work based off world space positioning, so work the same for any shape or degree of grid.

4.5 Constraints

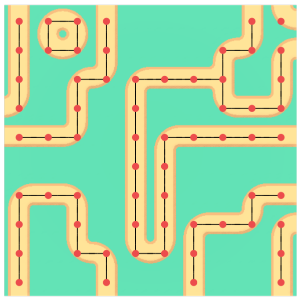
While Tessera is not a fully general constraint solver, the implementation does permit arbitrary constraints to be configured. These constraints are consulted during the search process, and can narrow the domains of variables (i.e. the set of possible tiles for a given cell). Tessera comes with a suite of constraints, focused on providing *global* conditions to supplement the local constraints WFC is equipped with. This includes enforcing symmetry, or particular tile counts.

Particularly useful are path constraints. Path constraints work by defining a subset of the tiles, called path tiles. Define a subgraph, called the path graph, by considering only the cells in the grid with path tiles in them (Figure 10a). Each constraint asserts some basic graph theoretic property of the path graph. There are currently the following constraints (Figure 11):

- Connected - the graph must be fully connected i.e. given two path tiles, there is a series of adjacent cells between them all containing path tiles.
- Acyclic - the graph must acyclic i.e. there is at most one possible path between any two path tiles.

- Cyclic - every graph node must be inside at least one loop.
- Parity - early failure constraint for specific tilesets[6]

The path constraints also support *directed* variants where the path tiles are not just organized into a set, but also the path can only connect through the tile on specified faces. In Tessera this is specified by a set of colors, which are scanned for on the tile paint the users put on each tile. The directed variants use the same implementations but use a larger path graph that has multiple nodes per cell - one for the cell itself, and an additional node per face. (Figure 10b).


(a)

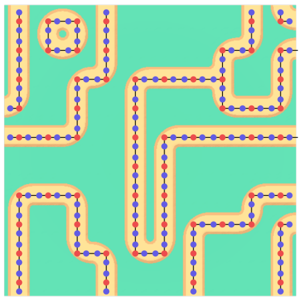

(b)

Figure 10: A grid of cells with fully specified tiles, overlaid with the subgraph of nodes used by an undirected and directed constraint. Red nodes correspond to cells themselves, and blue nodes correspond to directed adjacencies from one cell to another.

Graph-based connectivity constraints have been used for procedural content generation before[21]. However, previous work uses Answer Set Programming, which requires connectivity to be described as large collection of simple logical constraints. This can be slow to evaluate. Tessera's global constraints work like any other SMT solver[7], they have a specific *theory* of graphs and can take advantage of graph specific properties to achieve much better performance.

Our connectivity constraint runs a modified version of Tarjan's algorithm[32] on the partially generated level. This finds articulation points[8] and unreachable nodes, then directs the solver's search appropriately. A similar approach is described in [3]. The other path constraints are also implemented with Tarjan's, or other depth-first search techniques.

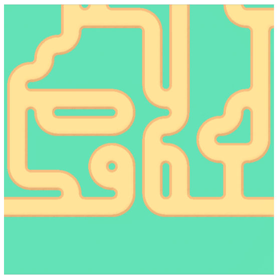
4.6 Debugging utilities

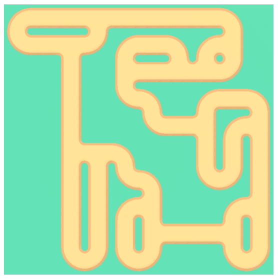
Tessera comes with several features to assist users with designing their tiles and constraints. A common problem with constraint-based generation is when the generation fails, there is no clear explanation for what caused the issue. If a contradiction occurs

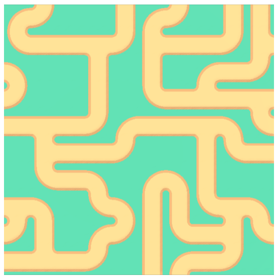
[6]Tilesets that have directed paths, and all the tiles have an even number of exits. https://twitter.com/boris_brave/status/1350467618298331137

[7]Satisfiability modulo theories, i.e. a solver that has extensions beyond the basic boolean logic of a SAT solver[36]

[8]Also known as "cut vertices", articulation points are nodes that, if removed, would separate the graph into multiple connected components. To deal with unresolved cells during generation, a slightly modified definition is used instead.


(a) Connected


(b) Cyclic


(c) Acyclic

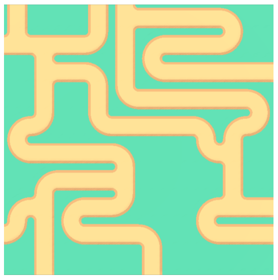

(d) Acyclic and connected

Figure 11: Example outputs using various path constraints

Tessera will log what location the contradiction is at. It can be configured to output a minimized set of tiles that cause the issue to occur, which can indicate to the user if a new tile needs to be added to the tile set.

When backtracking is enabled, instead of reaching a contradictory state the solver tends to get stuck repeatedly exploring the same area over and over. In this case, users can enable an animation of the generation process, stepping through each guess/backtrack one at a time. This usually highlights the problem area that the solver tries to generate over and over.

5 CONCLUSION

In this paper, we presented *Tessera*, a Unity tool for configuring and running constraint-based procedural generation. We've covered how it makes configuration easy for users, and how it supplies extensions to WFC to give it more control and power. Both of these show some improvements from earlier work.

Tessera has been a popular tool for WFC since release, being ranked #29 in "Tools/Level Design" in the Unity asset store. Meanwhile the library component, DeBroglie, has received less attention despite having similar features, being free and released earlier. We attribute this is mostly due to the user friendliness of the design.

We believe that constraint based procedural generation is a rich area for design, due to its flexibility, and ease of integration with other techniques and with mixed initiative requests from users. WFC occupies a "sweet spot" in that it is powerful enough to produce a wide range of effects, but simple enough that it is comprehensible to users, and doesn't require using a constraint programming language to control. Controllability remains an issue, even with Tessera's modifications, and we hope future research will streamline the experience for users even further.

REFERENCES

[1] A. M. Abuzuraiq, A. Ferguson, and P. Pasquier. 2019. Taksim: A Constrained Graph Partitioning Framework for Procedural Content Generation. In *2019 IEEE Conference on Games (CoG)*. 1–8. https://doi.org/10.1109/CIG.2019.8848065

[2] Fahiem Bacchus and Peter van Beek. 1998. On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. (09 1998).

[3] Christian Bessiere, Emmanuel Hebrard, George Katsirelos, and Toby Walsh. 2015. Reasoning about Connectivity Constraints. In *Proceedings of the 24th International Conference on Artificial Intelligence* (Buenos Aires, Argentina) *(IJCAI'15)*. AAAI Press, 2568–2574.

[4] Brian Bucklew. 2019. *Dungeon Generation via Wave Function Collapse*. YouTube. https://www.youtube.com/watch?v=fnFj3dOKcIQ

[5] Kate Compton, Adam Smith, and Michael Mateas. 2012. Anza Island: Novel Gameplay Using ASP. In *Proceedings of the The Third Workshop on Procedural Content Generation in Games* (Raleigh, NC, USA) *(PCG'12)*. Association for Computing Machinery, New York, NY, USA, 1–4. https://doi.org/10.1145/2538528.2538539

[6] Plausible Concept. 2018. Bad North. https://www.badnorth.com

[7] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[8] Isaac Dykeman. 2019. Generate Worlds. http://generateworlds.com

[9] Freehold Games. 2017. Caves of Qud. https://twitter.com/cavesofqud

[10] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli, Rajeev Alur, and Doron Peled. 2004. DPLL(T): Fast Decision Procedures. *Computer aided verification : 16th International Conference, CAV 2004, Springer, 175-188 (2004)*. https://doi.org/10.1007/978-3-540-27813-9_14

[11] Maxim Gumin. 2016. WaveFunctionCollapse. GitHub repository. https://github.com/mxgmn/WaveFunctionCollapse

[12] Narendra Jussien, Guillaume Rochart, and Xavier Lorca. 2008. Choco: an Open Source Java Constraint Programming Library. *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)* (01 2008).

[13] Isaac Karth and Adam M. Smith. 2017. WaveFunctionCollapse is Constraint Solving in the Wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games* (Hyannis, Massachusetts) *(FDG '17)*. Association for Computing Machinery, New York, NY, USA, Article 68, 10 pages. https://doi.org/10.1145/3102071.3110566

[14] 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* (San Luis Obispo, California, USA) *(FDG '19)*. Association for Computing Machinery, New York, NY, USA, Article 89, 9 pages. https://doi.org/10.1145/3337722.3341845

[15] Hwanhee Kim, Seongtaek Lee, Hyundong Lee, Teasung Hahn, and Shinjin Kang. 2019. Automatic Generation of Game Content using a Graph-based Wave Function Collapse Algorithm. 1–4. https://doi.org/10.1109/CIG.2019.8848019

[16] Thorbjørn Lindeijer. 2021. Using Terrains. https://doc.mapeditor.org/en/stable/manual/terrain/

[17] R. Mohr and T. Henderson. 1986. Arc and Path Consistency Revisited. *Artif. Intell.* 28 (1986), 225–233.

[18] Adam Newgas. 2018. DeBroglie. https://github.com/BorisTheBrave/DeBroglie

[19] Adam Newgas. 2019. Tessera Pro. http://u3d.as/1Jqi

[20] Stuart Russel and Peter Norvig. 2003. *Artificial Intelligence : A Modern Approach*. 202–233.

[21] Noor Shaker, Julian Togelius, and Mark J. Nelson. 2016. *ASP with applications to mazes and levels. Procedural Content Generation in Games* (1st ed.). Springer Publishing Company, Incorporated.

[22] João P. Marques Silva and Karem A. Sakallah. 1997. GRASP—a New Search Algorithm for Satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design* (San Jose, California, USA) *(ICCAD '96)*. IEEE Computer Society, USA, 220–227.

[23] Adam Smith and Michael Mateas. 2011. Answer Set Programming for Procedural Content Generation: A Design Space Approach. *Computational Intelligence and AI in Games, IEEE Transactions on* 3 (10 2011), 187 – 200. https://doi.org/10.1109/TCIAIG.2011.2158545

[24] A. J. Smith and J. Bryson. 2014. A Logical Approach to Building Dungeons: Answer Set Programming for Hierarchical Procedural Content Generation in Roguelike Games Legible Machine Learning for Body Language Based Game-play Emerging Dimension Weights in a Conceptual Spaces Model of Concept Combination From Observer-Relativity to Assig.

[25] A. M. Smith, Erik Andersen, M. Mateas, and Z. Popovic. 2012. A case study of expressively constrainable level design automation tools for a puzzle game. In *FDG*.

[26] Gillian Smith, Jim Whitehead, and Michael Mateas. 2011. Tanagra: Reactive Planning and Constraint Solving for Mixed-Initiative Level Design. *Computational Intelligence and AI in Games, IEEE Transactions on* 3 (10 2011), 201 – 215. https://doi.org/10.1109/TCIAIG.2011.2159716

[27] Thomas Smith, Julian Padget, and Andrew Vidler. 2018. Graph-Based Generation of Action-Adventure Dungeon Levels Using Answer Set Programming. In *Proceedings of the 13th International Conference on the Foundations of Digital Games* (Malmö, Sweden) *(FDG '18)*. Association for Computing Machinery, New York, NY, USA, Article 52, 10 pages. https://doi.org/10.1145/3235765.3235817

[28] Simon Stix. 2020. Tile Composer. https://stixgames.com

[29] Oskar Stålberg. 2017. https://twitter.com/OskSta/status/917406082384957440

[30] Oskar Stålberg. 2019. https://twitter.com/OskSta/status/1131228226901172226

[31] Oskar Stålberg. 2020. Townscaper. https://store.steampowered.com/app/1291340/Townscaper/

[32] R. Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1 (1972), 146–160.

[33] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. 2011. Search-Based Procedural Content Generation: A Taxonomy and Survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186. https://doi.org/10.1109/TCIAIG.2011.2148116

[34] Guy Walker. 2014. Wang Tiles: Blob Tileset. http://www.cr31.co.uk/stagecast/wang/blob.html

[35] H. Wang. 1961. Proving theorems by pattern recognition — II. *The Bell System Technical Journal* 40, 1 (1961), 1–41. https://doi.org/10.1002/j.1538-7305.1961.tb03975.x

[36] Jim Whitehead. 2020. Spatial Layout of Procedural Dungeons Using Linear Constraints and SMT Solvers. In *International Conference on the Foundations of Digital Games* (Bugibba, Malta) *(FDG '20)*. Association for Computing Machinery, New York, NY, USA, Article 101, 9 pages. https://doi.org/10.1145/3402942.3409603