Universiteit Leiden

Modern Game AI Algorithms 2023, Assignment 1

Procedural Content Generation

Tom Stein

February-March 2023

Leiden University
Faculty of Science
Leiden Institute of Advanced Computer Science (LIACS)
https://liacs.leidenuniv.nl/

Abstract

1 Introduction

The world in modern games are becoming very large, requiring lots of manual work by designers to build and place buildings and other elements of the world. Recently, the use of procedural content generation (PCG) in this area stated to increase because it lowers the required amount of work by automatically generating believable content. There are many problems where PCG can be applied, but this report explores the generation of a believable house in Minecraft, including the placement, consistent structure and randomness. For the generation of the house itself, the wave function collapse algorithm will be used.

The wave function collapse (WFC) algorithm [Gum], which was introduced in 2016 by Maxim Gumin, is a new algorithm for PCG inspired by quantum mechanic principles. It uses the concept of superposition and state collapse. A particle can be in a superposition, i.e. in many states at once, until it collapses to a single state. The same concept is used in the WFC algorithm, which first initializes all "particles" in the state space to a superposition and then collapses one of them, propagating the induced changes to neighboring particles that are defined by a set of rules. Take for example a rule that states that next to a particle in state A there must be a particle in state B. If we have a state space of two particles in the initial superposition: $[(A, B), (A, B)]$ and collapse the first to state B: $[(B), (A, B)]$, the rule forces us to choose A for the second $[(B), (A)]$ particle. The algorithm keeps collapsing and propagating until all particles are in a single state.

The WFC algorithm has already been applied to problems in PCG such as building generation [Cha], where the individual particle states need to be thought of as building structure elements like walls, windows and doors, or 2D level generation [Don], where the states are reusable level fragments. It has also been applied to village generation in Minecraft [Mif] using individual buildings and infrastructure components instead of the particle states.

Structure

The report is structured into independent sections, containing the method and results for the respective component. First, a method for the building placement is described, results are shown and limitations are discussed. The next section focuses on the building generation using the WFC algorithm. Third, a method do add interior decoration is shown. Fourth, results for the combination of building placement, building generation and interior decoration are shown. Finally, the report concludes with a discussion of the results, the limitations of this approach and directions for improvement.

2 Building Placement

3 Building Generation

This section describes the generation of buildings using the wave function collapse (WFC) algorithm.

3.1 Structure Building Blocks

In order to use the WFC algorithm a set of small structures, sometimes called tiles or prefabs, is required to combine them to a larger structure, i.e. a building. These structures can in general be of any granularity from single blocks to larger groups such as walls, rooms or even whole corridors. However, there is a trade-off between variation and believability. There are more combinations of smaller structures in a fixed area than there are combinations of larger structures but many of these combinations of smaller structure combinations result in unrealistic buildings, e.g. five doors placed next to each other. Therefore, building structures with a size of $11 \times 6 \times 11$ and $11 \times 9 \times 11$ blocks were used, where each represents a single room of a building. This size gives enough possible variations in the typical size of a Minecraft house while believability, accessibility and architectural style. A selection of the designed structures can be seen in Figure 1.

(a) From left to right: building entrance corner, balcony and big window corner.

(b) Doors between each building structure.

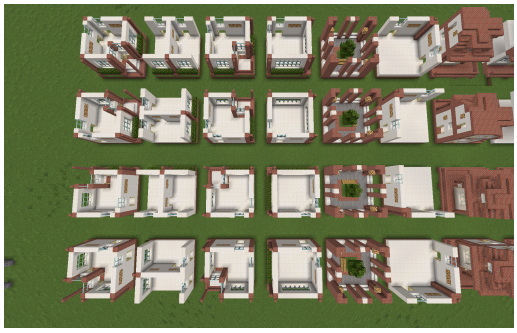
Figure 1: Some manually designed building structures which can be combined by placing them next to each other. Doors between them are aligned to ensure accessibility of the whole building.

In order to duplicate these structures, and apply translation and rotation onto them, they need to be constructed using a computer program. It was chosen to follow a scanner-builder approach over manually translating these structures into code that constructs them using simple geometry like cuboids because the scanner-builder approach is easier to implement and allows for faster iterations. The scanner[1] works by serializing the user defined build area to a Python pickle file. Afterwards, the builder[2] can be used to duplicate the scanned structure with arbitrary rotation and translation. Every structure was scanned from the bottom left corner in positive XZ direction, by standing on the golden blocks shown in Figure 1a.

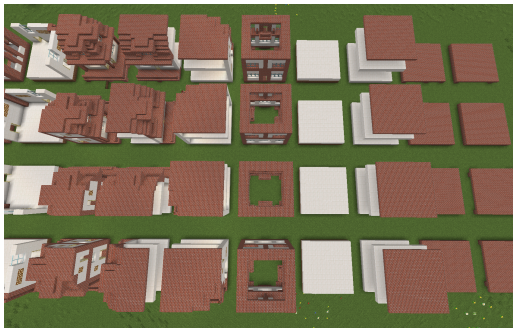
In total 14 building structures were designed, which are shown in figure Figure 2. They are split between first floor ($11 \times 6 \times 11$) and roof house ($11 \times 9 \times 11$) elements. There are simple corner elements and middle wall elements to allow building $n \times 2$ rectangular shaped buildings. Additionally, center elements like `center` and `courtyard` were added to support $n \times m$ shapes. Later, the concept of an inner corner was added to allow for more complex shapes like L, O and $+$.

With the introduced building structures one can already build whole houses by deterministically placing them next to each other, while making sure that all doors

[1] `structure_scanner.py`
[2] `structure_builder.py`



(a) From left to right: building entrance corner, middle wall, balcony corner, big window corner, courtyard and inner corner.



(b) From left to right: corner roof house, middle wall roof house, inner corner roof house, courtyard roof house, center, center roof house, corner flat roof and big window flat roof.

Figure 2: All 14 designed building structures in all rotations. Each column represents a single structure and each of the four rows represent a rotation of that structure. The lowermost row is the baseline rotation 0.

align as they are supposed to. Two examples are shown in Figure 3.

(a) A 3×3 house with courtyard and a second floor (roof house).

(b) A 2×2 house with a flat roof.

Figure 3: Two deterministically built buildings.

3.2 Wave Function Collapse

With the previously introduced set of structures it was only possible to generate deterministic buildings so far. Using the wave function collapse (WFC) algorithm it will be possible to generate randomized buildings which are still believable because they are only built out of valid combinations of structure building blocks.

3.2.1 Rule Set

To achieve this, it is necessary to define the adjacency rules for each structure. These define how structures can be placed next to each other, whereby rotations of the structures must also be taken into account. For the 14 structures with six sides and four rotations (rotation around the Y-axis only), this would mean that $14 \cdot 6 \cdot 4 = 336$ rotated sides would have to be considered. For each of them one would have to define whether they may be placed next to each other, i.e. $\frac{336(336-1)}{2} = 56280$ rules.

Obviously, this is not very practical to define so many rules manually. Therefore, one can make use of the symmetries of the structures and define the adjacency rules only for one rotation of the structure. The others can be derived from this. In addition, one can only define the rules that are allowed, and assume for all rules that are not defined that they are not allowed.

The total of 320 rules[3], that were necessary for the 14 structures, were realized directly in Python. A small excerpt from this is shown in Listing 1. Direct relationships between the objects were specified directly, i.e. if A is allowed next to B, there must also be a rule that B is allowed to be next to A. This explicit symmetry makes it easier to create the rules, as errors can be avoided in this way. Additionally, a consistency check `check_symmetry` for the rule set was implemented along with some other unit tests because the process of defining the rule set and the implementation to drive the implicit rules are both nontrivial.

```
structure_adjecencies = {
    brickhouse_middle: StructureAdjacency(
        structure_name=brickhouse_middle,
        x_plus=[
            StructureRotation(brickhouse_middle, 0),
        ],
        x_minus=[
            StructureRotation(brickhouse_middle, 0),
        ],
        z_plus=[
            StructureRotation(brickhouse_middle, 2),
            *all_rotations(brickhouse_center),
        ],
    ),
    brickhouse_center: StructureAdjacency(
        structure_name=brickhouse_center,
        x_plus=[
            *all_rotations(brickhouse_center),
            StructureRotation(brickhouse_middle, 1),
        ],
        x_minus=[
            *all_rotations(brickhouse_center),
            StructureRotation(brickhouse_middle, 3),
        ],
        z_plus=[
            *all_rotations(brickhouse_center),
            StructureRotation(brickhouse_middle, 2),
        ],
        z_minus=[
            *all_rotations(brickhouse_center),
            StructureRotation(brickhouse_middle, 0),
        ],
    ),
}
```

Listing 1: A (very) short excerpt from the adjacency rules limited to the middle wall and center structure. All rules are defined from the respective element's perspective in rotation 0 (lowermost row in Figure 2).

3.2.2 Wave Function Collapse Algorithm

The wave function collapse algorithm [Gum] is used to generate random valid combinations of the previously introduced building structures. A simplified version of

[3] `structure_adjacency.py`

the algorithm[4] is shown in Listing 2. The algorithm starts with a state space where each cell is in a superposition of all available building structures. If the minimal overall entropy is zero, there is at least one cell for which we can't determine any fitting building structure anymore and the whole WFC process restarts. Otherwise, a random cell among all cells with that entropy is selected. Afterwards, that cell is collapsed to a single building structure out of the still possible building structures for that cell (it's superposition). The cell state collapse is propagated to update all other cell's superposition using the previously defined set of rules. This is repeated until all cells are collapsed, i.e. exactly one building structure is selected for them.

```
def wfc(self):
    self.self._initialize_state_space_superposition()

    while not self.collapsed():
        min_entropy = self.min_entropy()
        if min_entropy == 0:
            # the current state is unsolvable now. Restart.
            self.self._initialize_state_space_superposition()
            continue

        next_cells_to_collapse =
        ↪  list(self.cells_with_entropy(min_entropy))
        x,y,z = random.choice(next_cells_to_collapse)

        state_superposition = list(self.state_space[x][y][z])
        collapsed_state: StructureRotation =
        ↪  random.choice(state_superposition)

        self.collapse_cell(cell_xyz, collapsed_state)
```

Listing 2: A simplified version of the implemented wave function collapse (WFC) algorithm.

To ensure that the generated building is closed, in the sense of only outer walls from the outside, it was necessary to introduce the concept of an air building structure. That structure is essentially the same as any other structure, except that is does not contain any blocks and is only composed of air. Using this structure and the appropriate rules that define which sides of an actual building structure need to be placed next to air, one can enforce closed buildings by collapsing the outermost rectangle of cells in the state-space to air.

4 Interior Design

The buildings that can be constructed using the techniques and assets presented so far are missing interior design like decoration, furniture and lights. To add this to the buildings each building structure can be built with different interior designs, yielding a set of variations of that structure that can be used as a drop-in replacement for each other. After the WFC algorithm finishes, another algorithm may replace any number of building structures with such a drop-in replacement to add interior design to the building.

As the focus of this assignment was not on interior design of Minecraft houses, only a few building structures were equipped with these elements. The results can be seen in Figure 4, Figure 5 and Figure 6.

[4] `wave_function_collapse.py`

Figure 4: Entrance interior design elements.

Figure 5: Middle and center interior design elements.

5 Overall Results

6 Discussion

Figure 6: Center and bedroom interior design elements.

References

[Cha] Eleni Chasioti. Gameplay with encoded architectural tilesets: A computational framework for building massing design using the wave function collapse algorithm.

[Don] Martin Donald. Superpositions, sudoku, the wave function collapse algorithm. https://www.youtube.com/watch?v=2SuvO4Gi7uY.

[Gum] Maxim Gumin. Wave function collapse algorithm. https://github.com/mxgmn/WaveFunctionCollapse.

[Mif] Jakub Mifek. Procedural generation of minecraft villages utilizing the wave function collapse algorithm.