Tile Assembly Simulator in Python using OpenCV

Tharun Mohandoss(tm35848), Surya Sen Dwivedi(ssd982)
May 9, 2020

Abstract

We explore various methods for implementing tile assembly model. We also a implement a visualization for the same in OpenCV.

1 Introduction

We implement a simulator for tile assembly system. The initial baseline implementation is done using fixed grid simulation in C++. We further improve this approach by keeping a dictionary for storing the configuration and also implement a visualization method in C++.

2 Fixed-Grid simulation technique

For the baseline implementation, we define a grid of fixed size and the tile at (0,0) is the seed tile. We further define glue alphabet, tile types and temperature variable. For simulating this model, we choose a random location of the grid and a random tile and see if the tile 'fits' the location (i.e. satisfies constraints of the tile assemble system). This is repeated for until either a new valid location for a tile is found or the max iterations in reached. The advantage of this approach over the next approach is the worst case time complexity of a lookup since it's a direct read from an array. Also this does not require storing any extra data-structures.

3 The dictionary method for storing the configuration

An alternative method to implement is where we allocate space for a square in the grid only when it is filled in the coordinate space. This can be done using a dictionary which maps a 2d coordinate to a Tile object.

Advantages :
1. Space is only allocated for the filled squares : Take the case where we have a long nx3 grid parallel to the x axis, If we preallocate a grid of size nxn, we would be using on 3/n fraction of the space allocated.
2. Tile system can expand in any direction : In case of the fixed grid system, if the tile system tries to grow beyond the borders in any direction, we would need to reallocate the whole grid and copy paste each square even if only a small portion of the grid is currently allocated but in the dictionary method we can grow the tile system in arbitrary direction.
3. This allows us randomize the selection of the next tile efficiently(Explained further in the next section)

Disadvantages :
1. Worst case access time is large : Although dictionary offer an expected O(1) time access, their worst case access times are much worse.

4 Storing potential candidate locations for new tiles

In case of fixed allocation, it is possible to choose co-ordinates to fill next by choosing one of the grid position randomly and filling that coordinate with an appropriate tile if the following conditions hold :
1. That coordinate is empty.
2. There are tiles nearby S.T there is at-least one tile that can be added satisfying the temperature requirements of the tile system.

But this means that we need to keep checking random locations till we find a new tile. There is no bound on the number of tries required to successfully allocate a new tile. The probability of finding a coordinate with a

potential new valid tile can be extremely low especially when there is a large grid with small number of existing tiles. Thus, this method for growing tiles is extremely inefficient.

Instead of this method, we define a dictionary to map valid coordinate locations to the list of valid candidate tiles for that co-ordinate. This dictionary is updated each time a new tile is added. When we add a new tile to coordinate (x,y) we remove it from the potential coordinates dictionary and re-evaluate it's 4 neighbours for addition/modification in the potential coordinate dictionary. When we have to choose a new coordinate and tile for expansion, we can just choose one of the these valid locations and choose one of the tiles from the list of valid tiles that we have already computed and stored. This is a much more efficient way for random expansion of the tile system. Also, this allows us to detect when a terminal configuration is reached so that we can stop immediately.

5 The interface

5.1 Input to the program

The specifications of the tile systems i.e details of glues, different types of tiles, seed tile locations, temperature, maximum iterations before stop and other options such as whether or not to store images of the stepwise progress of the tile system are expected to supplied to the program via a configuration file. An example is shown in Fig 1.

```
#glues, stored as (name, strength) pairs
self.glues = [
        ("a",2),
        ("b",1),
        ("c",2),
        ("d",0)
    ]

#tiles, stored as (name,(glue_list)) where glue_list is stored in west,north,east,south order
self.tiles = [
        ("t1",("a","b","c","a")),
        ("t2",("a","a","a","a")),
        ("t3",("b","b","b","b")),
        ("t4",("c","c","c","c"))
]

#seed tiles stored as (coordinate,tilename) pairs
self.seed_tiles = [
        ((0,0),"t1"),
        ((1,0),"t2")
]

#temperature
self.temperature = 2

#max new tiles to add
self.max_iters = 10000

#save image or not and if so folder
self.save_images = True
self.save_folder = "./NewVideoImages/"
```

Figure 1: An example configuration file

5.2 Classes and functions

1. Class TAS : This is the main class of the program. Given the configuration it initializes initial configuration using the information about seed tiles specified in the config file. It has a simulate function which requires the maximum iterations value as input and grows the tile system until either a terminal configuration is reached or the maximum iterations value is reached.
2. Class Image : When passed the object of tas class and the config, this create a visualization of the growth of the tile system as a sequence of images. This also has a showImage function to display the final state of the tile assembly system.

5.3 Visualization

The image of the tile-system is created using OpenCV. An example final state after 1000 steps is shown below.

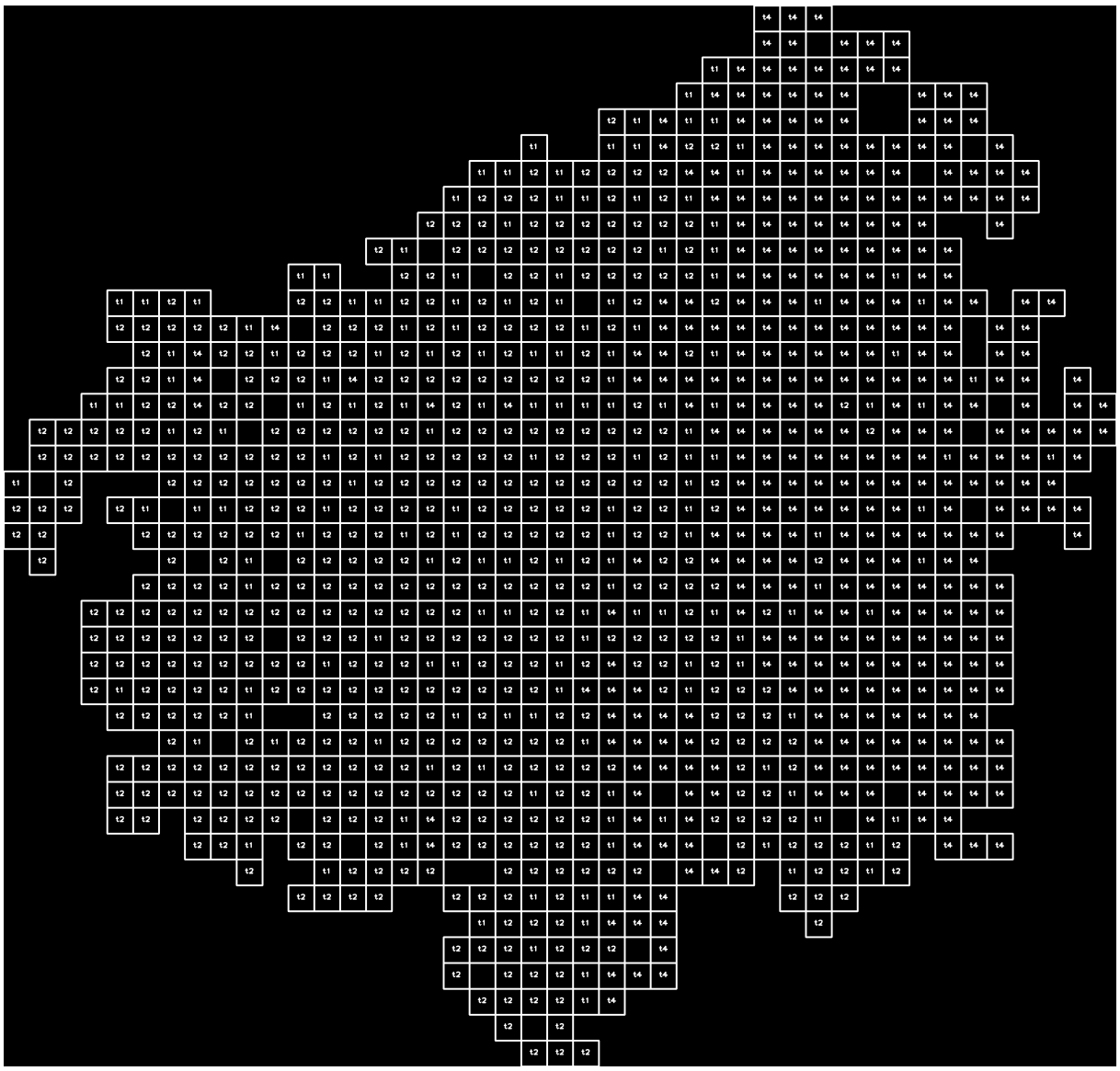

Figure 2: Visualization image