The Game of Life on Penrose Tilings: Pentagon Boat-Star

Raghav Goel

Department of Mathematics

Advisor: Dr. May Mei

Introduction

In the 1970s Roger Penrose and John Conway presented the mathematical world with two beautiful and intriguing developments - Penrose with his sets of tiles that tile the plane non-periodically providing us a class of aperiodic tilings known as Penrose tilings [5], and Conway with his Game of Life, a cellular automaton simulated on a lattice of square cells with extremely simple rules governing the future state of the cells that can lead to extremely complex ‘organisms’ being born out of relatively simple starting configurations. [4]

We have investigated the results of playing the Game of Life on a Penrose tiling, and present findings from our simulations.

The Game of Life

Conway's Game of Life is played on an infinite grid of squares, with each square being known as a cell. The fate of a cell in the next generation of the simulation depends on the states of its immediate neighbors as follows: [4]

1. If a cell is alive and has 2 or 3 live neighbors, it survives to the next generation. Otherwise, it dies.
2. If a cell is dead and has exactly 3 live neighbors, it becomes alive in the next generation.

From the application of these rules we observe many figures or *organisms*, which can be classified into a few broad categories. [4] (images from [7])

- A *still life* is an organism in which all cells survive to the next generation and is thus in a state of equilibrium - no births, no deaths.

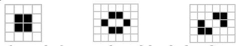

Figure 1: Still lifes, from left to right – block, beehive, aircraft carrier

- An *oscillator* is an organism that starts at some initial state and after a certain number of generations (known as its period) returns to that state in the same position.

Figure 2: An oscillator called a blinker

- A *spaceship* is an organism that translates across the plane by a certain vector in a specific number of generations.

Figure 3: Phases of the Lightweight Spaceship (LWSS)

Penrose Tilings

We use what is referred to as the P1 protoset, which consists of six tiles – a star, boat, rhombus, and three regular pentagons (which differ in how they are substituted).

We use the substitution or inflate-and-subdivide method to generate this tiling. This involves taking a tile, enlarging it by a certain factor (which in our case happens to be the golden ratio $\phi = \frac{1+\sqrt{5}}{2}$), and replace this enlarged tile by a certain arrangement of tiles with the same size as the original tile, and which collectively roughly imitate its shape. [6] (see Fig. 4) Applying this process repetitively n times we are able to produce an arbitrarily large patch which we call the n th level substitution patch.

Implementation

First, we generate the tiling and data regarding the tiles and their neighbors which we then use to apply the Game of Life rules on the tiles. We choose to use the Julia programming language for this purpose due to its easy learning curve and performance benefits, which helps greatly due to the large numbers of tiles we would be taking into consideration.

We begin by defining a `Tile` struct which contains basic properties of a tile such as its type, its “origin” which is the first vertex of the tile to be drawn and from which all other vertices are calculated, the angle between its line of symmetry passing through the origin and the horizontal axis, and a dictionary for storing its neighbors.

We observe that a tile can be formed using just 3 pieces of information - the type of tile, its origin and its angle with the horizontal. Let us suppose we wish to draw a rhombus whose first vertex is at the origin of the plane and makes and angle of 0 with the horizontal. Then going anti-clockwise, we know that the vectors from the tile's origin to the other points make angles of $\frac{\pi}{10}$, 0 and $-\frac{\pi}{10}$, and by calculating the magnitudes of these vectors, we can determine the coordinates for the other vertices. These coordinates can thus be generated on-the-fly later when we plot these tiles for a visual representation of them. After generating these coordinates, we define functions that carry out the substitution rules according to the tile’s type.

Next, we define methods to determine the neighbors of a given tile in a patch. We do this as follows.

1. Choose a tile T and compute the coordinates of its vertices. Then, pick another tile T' and compute its vertices' coordinates as well.
2. We compare every vertex of T in anti-clockwise order to the vertices of T'. If any vertex of T is also a vertex of T', then T' is a neighbor of T. If two such vertices exist, that must mean T and T' share an edge, and we say that T' is an *edge neighbor* of T. Otherwise, T' is said to be a *corner neighbor* of T.

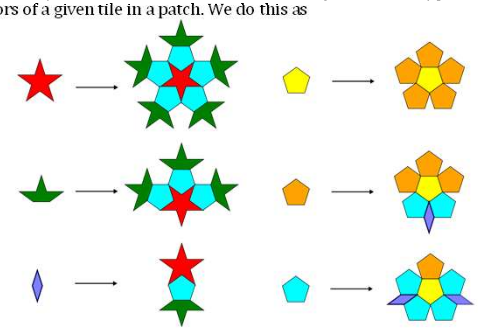

Figure 4: Substitution rules for the tiles of the P1/Pentagon Boat Star prototile set

This process clearly take a lot of computation time due to the large number of comparisons between points taking place. Thus, we compile the neighbor information of all tiles in a given patch into a dictionary whose keys are the indices of the tiles within the patch, and the values are their respective neighbor dictionaries. We then proceed to store this dictionary into a `JLD2` file which is a file format designed for saving and loading Julia data structures.

Since we have the number of neighbors for each tile in the patch, we can implement the Game of Life rules for the same. Firstly, we pass a list of live indices to initialize our simulation. Next, we extract the lists of neighbors for each tile and check if any of those neighbors' indices are alive. Then, according to the number of live neighbors the tile is determined to have, we decide whether or not to add the tile's index to the list of live indices for the next generation which is returned by our function. We can then repeat this process for as many generations as we like, passing in the output of the previous generation as an input for the next one.

To visualize the tiling and the Game of Life being played on it, we utilized the `Luxor` package. After getting the tiles in a patch we proceed to compute the vertices for each of them. We then use Luxor's `poly` method to join those vertices to form a closed polygon. When playing the Game of Life, we add a black, filled polygon in the same manner as we did for the tiling at the coordinates corresponding to the tile at the live index. We generate an SVG image for every generation using `Luxor's @svg` macro and animate and store it to a GIF file using the `@animate` macro.

We limited our testing space to six patches pulled from the 5th substitution patch of a type-1 pentagon tile, centered roughly around the center of the patch. We did so in an effort to get a hold of a reasonably large enough patch for our simulations while minimizing the effects of considering tiles on the boundary of the patch, which lead to inaccurate results due to having an inaccurate number of live neighbors counted. After running the Game of Life simulation multiple times on these patches we have observed numerous still lifes and oscillators. The smallest still lifes and oscillators found comprised of 3 tiles each. Oscillator periods vary between 2 and 6. We have presented some of the organisms we have identified here.

Findings

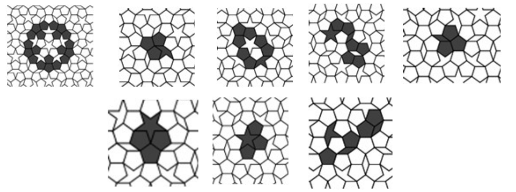

Figure 5: Still lifes observed

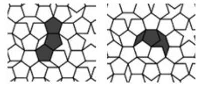

Figure 6: A period 2 oscillator

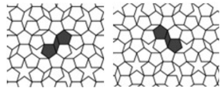

Figure 7: Another period 2 oscillator

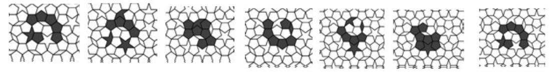

Figure 8: A period 6 oscillator

Acknowledgements

I would like to extend my gratitude to the J.R. and P.V. Endowment for funding this research, and to Dr. Mei, the Lisska Center and the Anderson Science Program for providing me with an opportunity to undertake this endeavour. Furthermore, I would like to thank my teammates Zheng Fang and Mandy Hong for providing useful input and collaboration throughout this project.

References

[1] C. Adams, The tiling book. (Unpublished)

[2] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, Julia: A fresh approach to numerical computing, SIAM review, 59 (2017), pp. 65–98.

[3] D. Frettlöh and F. Gähler, Penrose Pentagon Boat Star, Tiling Encyclopedia , https://tilings.math.uni-bielefeld.de/substitution/penrose-pentagon-boat-star/.

[4] M. Gardner, Mathematical games, Scientific American, 223 (1970), p. 120–123.

[5] M. Gardner,, Mathematical games, Scientific American, 236 (1977), p. 110–121.

[6] B. Grünbaum and G. C. Shephard, Tilings and Patterns, Second Edition, Dover Publications, New York, 1987 (Reprint 2016).

[7] N. Johnston and D. Greene, Conway’s Game of Life: Mathematics and Construction, Self-published, 2022