Conway's Game of Life

Ana Gorohovschi and Nick Williams
{Ana.Gorohovschi001@umb.edu},{Nicholas.Williams002}@umb.edu
University of Massachusetts Boston

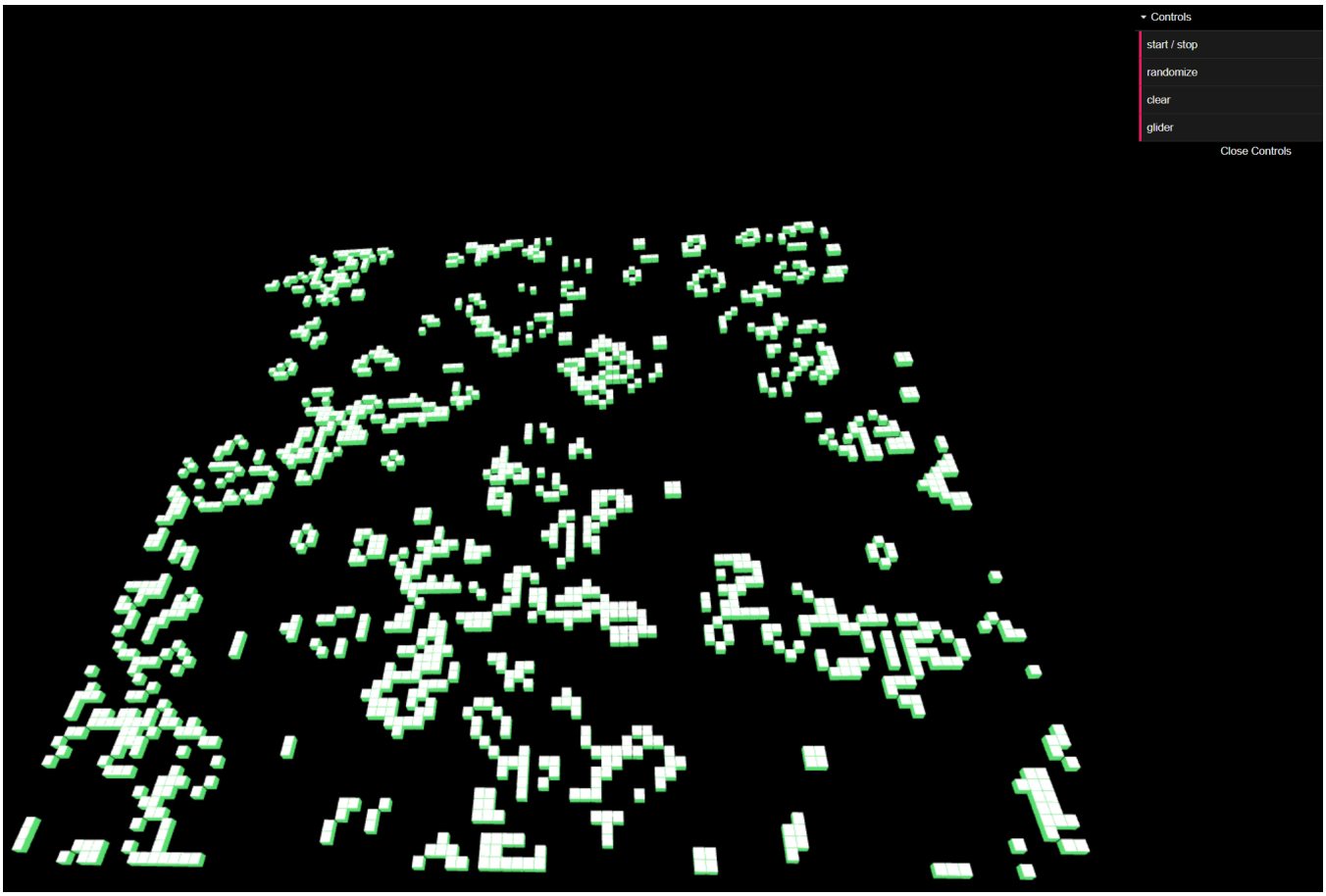

Figure 1: Sample generation.

ABSTRACT

In this project, we recreate Conway's Game of Life using the skill we learned during the course of this semester. This showcases an algorithmic simulation of cells while within a two-dimensional plane, where the generation and reproduction of the cells follow the rules of the game.

KEYWORDS

HTML, Game of Life, Calculation

ACM Reference Format:
Ana Gorohovschi and Nick Williams. 2021. Conway's Game of Life. In *CS460: Computer Graphics at UMass Boston, Fall 2021.* Boston, MA, USA, 4 pages. https://CS460.org

CS460, Fall 2021, Boston, MA

ACM ISBN 1337.
https://CS460.org

1 INTRODUCTION

In this project, we aim to create a replica of Conway's Game of Life. This game, shortened to just 'LIFE', was created by Cambridge mathematician John Horton Conway in 1970, hence the name. It consists of a grid of cells. A cell can either be alive, or dead.The Game of Life is important as it is meant to represent cellular automation, a model of computation that has other applications in other fields of science, such as physics and biology. Both Ana and I contributed

in the coding process and we both found the concept to be fairly interesting for various reasons.

2 RELATED WORK

Three.js [1].

3 METHOD

The project mainly uses THREE.js for the framework with the following rules:

- Confinement: The cells are located within the bounds of the predetermine area aka the board.
- Calculation: Each 250ms a new generation of cell is computed from the old generation based on 2 rules:
  - A cell comes to life if it has exactly 3 live neighbors.
  - A cell stays alive if it has 2 or 3 alive neighbors, and is dead otherwise.
- We have provided an ability to randomize the initial alive cells, as well as manual control by clicking on the grid to either make a cell alive or dead. (SHIFT + CLICK) We have also included a classical pattern of a glider as an example of an interesting pattern.
- Randomization: If one were to start the game with the random button, the board is to fill up with cells in different positions each time. Half of randomly chosen cells will be alive
- Automation: The Start/Stop button starts or stops the calculation

3.1 Implementation

The initial choice for framework was XTK but that later proved difficult to use given that XTK moves the camera and in the case of the glider pattern the user could not see the glider moving. Both the glider and the camera were moving the result was a glider staying in place. Also we learned how to compute mouse clicks on a plane with THREEJS and using both libraries proved difficult.

The project follows the Model-View-Controller design structure.

- The model: is a 2D array of booleans called theWorld which holds either true or false for any particular i, j indices that represents a cell at that location either alive or dead.
- The view: is a 2D array of cubes called theCubes whose 'visible' attribute is turned on or off based on the theWorld 2D array.
- The control: is an onclick call back for the canvas that computes the indicies into theWorld 2D array and toggles the cell. The user must hold SHIFT key and click to toggle a cell

All computations are done on the 2D boolean array theWorld. When the user clicks on "Start/Stop" an update cycle starts. The function 'update' computes the new generation of alive and dead cells and swaps theWorld 2D array for the new one. This is possible a place for improvement. All updates are automatically shown since the "display" function is also in an event loop using the requestAnimationFrame

3.2 Milestones

- 1. Define a 2D array of booleans to represent the cells alive or dead.
- 2. Based on some predefined constants of a cube side size and gap between cubes. Arrange the cubes in 3D space.
- 3. Write and test the rules for the Game of Life
- 4. Provide an ability to toggle the cell in the 2D array

3.2.1 Milestone 1. Define a 2D array of booleans and update it based on the rules of the game. For testing purposes we used a simple HTML table

3.2.2 Milestone 2. Once the rules of the game were working we started working displaying them with XTK cubes. Based on defined constants cube side size (cubeSide) and cubeGap compute where in space would the cube with index [0][0] be then iterate over theWorld array and place cubes where theWorld array is true placing each cube a distance of (cubeSide + cubeGap) away from the previous one. That proved to be inefficient and later when we switched from XKT to THREE.js we decided to add all the cubes to the scene and only toggle their visibility parameter. That has been the most efficient approach so far and switching to THREE.js did not automatically moved the camera when the geometry updated.

3.2.3 Milestone 3. Add a callback method to process mouse clicks on the canvas to figure out the indices into theWorld 2D array and toggle that cell.

3.3 Challenges

The hardest part was dealing with inefficiencies and switching from XTK to THREE.js

- Challenge 1: The update function was fast but the display function was slowing down with time. We realized that the constant "add" then "remove" objects from the scene was the issue. We decided to add all objects and only display the ones we need.
- Challenge 2: Figuring out the indices into theWorld array from mouse click was hard particularly since we initially used the XTK library and tried to use THREE JS for the clicks. That lead us to nowhere so we made the full library switch.
- Challenge 3 (future development): Future improvement could be copy/paste a selection of cells; save/load a world

4 RESULTS

The project perfectly replicated CGL as the cell follow the rules of the game.

5 CONCLUSIONS

This project although simple on it's surface proved to be quite involved. Additionally the Game of Life seems trivial and not deserving of much time, yet people had done amazing work with it. We are sure that given better tools to simulate the game, people could discover additional life patterns that can have real world benefits.

REFERENCES

[1] Ricardo Cabello et al. 2010. Three.js. *URL: https://github. com/mrdoob/three.js* (2010).