Custom Game of Life- Cell Simulator

CS 320: Software Engineering
Software Requirements Specification
January 3, 2022

by:
Justin Flint
Kristine Hess
Alex Hong
Eric Le
Riley Jones
Dennis Balan

Table of Contents

Concept:

We will be implementing a webapp, programmed primarily in python, that can “play” a customized version of Conway’s Game of Life, where the lifecycle of cells is simulated. In our version of the game the user will have the option of using different cell shapes, can change the rules used to determine if cells live, die, or are born, and can set the initial cell population or use one that is randomly generated. The cells will be graphically displayed for the user in an infinite field, which they can zoom in on to focus on a smaller area, as the cells “grow.” The program will calculate the number of generations it takes for the cell population to stabilize and report various information about the system in which the cells live. As some “games” can be methuselahs, taking many generations to stabilize (as many as 50,000 in a bounded region using the original game rules), or infinitely growing “soups”, which never stabilize but can take millions of generations to go “boring”, we will have a save feature which allows users to return to their simulation later. Users will also have an opportunity to witness what the cells are “thinking” as they are born, live, and die in this, our Game of Life.

Epics:

API Creation - Justin Flint

Any product is dependent on interaction from the end user, and building a robust API allows users to change the rules of the simulation down to the last detail, providing them a new way to interact with our simulator. The end goal is to have a plugin manager that allows users to load and unload plugins for their instance of the simulation.

Feature 1: Plugin API
Feature 2: Plugin Manager

Algorithm Implementation - Kristine Hess

The “game” will use user supplied rules, a user chosen cell shape, and either a user chosen or random initial cell population to algorithmically determine the next generation of cells. The total cell population and other statistics will be tracked for a given cell population and displayed for the user.

Feature 1: Determine Next Generation
Feature 2: Cellular Statistics

Cellular Thought Process - Alex Hong

At any moment during the simulation process, the user will be able to select a cell and see information regarding the cell's status, the rules it has undergone, and a history of that cell's status.

Feature 1: Cell Information Gathering
Feature 2: Cell Thought Process Display

Website Implementation - Eric Le

The simulation will be website-based. It will have intuitive navigation, optimized load time, and utility-based features. The front end will be connected to the API for a customizable cell simulation.

Feature 1: Front End Design/Utilities
Feature 2: Login System

Graphical Display - Riley Jones

The user will be able to zoom in and out of a finite view of the infinite grid using a scroll bar or the scroll wheel and pan using four buttons, the arrow keys, or WASD. Users will be able to swap shaders "on-the-fly" to change graphical appearance.

Feature 1: Basic Rendering/Shaders
Feature 2: GUI

Save Game – Dennis Balan

The user will be able to use the save-game feature to save an instance of the game and be able to load it in some time.

Feature 1: Save Game
Feature 2: Load Game

Requirements:

Plugin API - Justin Flint (Major Milestone)

FR: Create a testing mode that will show what simulation element is being interacted with when it is active.
FR: Have hooks for startup that will allow plugins to modify the startup procedures of the simulation.
FR: Have hooks that will allow plugins to modify a single cell, groups of cells, or all cells.

FR: The API will have hooks that will allow plugins to pull information out of the last simulation.
FR: The API will have hooks that will let plugins create new types of cells.
NFR: The API must be well documented, allowing users to quickly figure out how to use it. [accessibility]

Plugin Manager - Justin Flint (Minor Milestone)

FR: The manager will allow users to upload, start, stop, and remove plugins outside of when the simulation is running.
FR: The manager will check for any immediate errors when a plugin is uploaded.
FR: Plugin Manager will store no data server side.
FR: Plugin Manager will have error codes to best approximate the issue should a plugin fails to load.
FR: The plugin manager will not edit the plugins loaded by it in any way.
NFR: Any text the manager uses must be contrasting, allowing it to be accessible to any user. [accessibility]

Determine Next Generation - Kristine Hess (Minor Milestone)

FR: When the program counts a cell's living neighbors, the algorithms for the specific cell shape will be used.
FR: When determining the next generation of cells, all cells in the nextGeneration matrix will be initially set to 0 (dead).
FR: When a cell is alive in the next generation, a 1 will be placed in the nextGeneration matrix at the cell's location.
FR: When determining the next generation of cells for the cell in matrix position [0][0] the status of the cell in [-1][-1] will be used. This same logic will apply to all edges, causing the field to wrap and create an infinite field.
FR: When the user presses stop on the simulator, the program will complete and return one final generation of cells (the nextGeneration matrix).
NFR: The status of all cells in the nextGeneration matrix must be calculated and returned within a maximum of 500 ms. [performance]

Cellular Statistics - Kristine Hess (Major Milestone)

FR: Once the user starts the cell simulator, the program will record updated cellular statistics with each generation.
FR: Upon reaching stability the cellular statistics cease calculation and are returned with a status of 0 (success).
FR: If the user terminates the simulation before stability is attained the cellular statistics cease calculation and are returned with an exit status of 1 (error).
FR: During the active period of the simulation, spaceships will be classified by the maximum number of cells they contain.

FR: When a pattern is repeated for the second time (the third oscillation cycle begins) it will be classified as stable, and the number of generations required to reach stability will be recorded as the last unique value before repetition began (current cycle – 1 – period of oscillation = last unique value).

NFR: The cellular statistics must accurately report the number of generations required to reach stability (stability means all cells die, a still life is reached, or the second oscillation cycle begins (first repeat)). [accuracy]

Cell Information Gathering - Alex Hong (Minor Milestone)

FR: When a cell is selected, a series of information gathering checks will start being performed on that cell.

FR: When the information check is made, the current tick the program is running on, the number of relevant surrounding cells, and the rule the cell follows will be recorded in an information holding object.

FR: When a selected cell follows a new rule, the rule will be recorded in an array of state changes.

FR: When the program performs a new update on the game, it will check which cell is being selected.

FR: If a new cell has been selected, the array of information holding objects will be refreshed.

NFR: The 10 most recent changes to the selected cell's behavior will be recorded in an array of information holding objects. [reliability]

Cell Thought Process Display - Alex Hong (Major Milestone)

FR: Upon right-clicking a cell, the interface will begin displaying a recorded history of that cell's state changes and surrounding detections.

FR: If a cell is selected, it will be outlined in a color that contrasts with the background of the grid.

FR: When a new cell is selected, the previously selected cell will no longer be outlined, and its recorded history will no longer be displayed.

FR: When a new state change is recorded, the state change will be displayed at the top of a list of recorded changes and will push down any preceding changes.

FR: When any more than 10 recordings have been made, the interface will no longer display the oldest recording so that only 10 recordings are displayed at a time.

NFR: Each change of state displayed in recorded history will have a description of the rule that the cell followed. [accessibility]

Front End Design/Utilities - Eric Le (Minor Milestone)

FR: A navigation bar which will follow the user as they scroll through the page

FR: When the user clicks on the start button, a loading animation will appear.

FR: A "help" or "?" button which will explain rules of the simulation when clicked.

FR: When the user clicks on an item in the navigation bar, the user will be directed to the respective page on the website.
FR: The text on the website will flow as the browser resolution changes.
NFR: The website must load within 2 seconds. [performance]

Login System - Eric Le (Major Milestone)

FR: Login page will connect and send information to a database via backend (For authentication).
FR: When the user creates an account, they will receive an email from the website.
FR: The “Forgot your password” button will send an email to the user for a password reset.
FR: When the user enters the website, there will be a pointer to the login page.
FR: The “Login” or “Create an account” button will redirect the user to the login page.
NFR: The account creation/login process is intuitive and easy for the user. [usability]

Basic Rendering/Shaders - Riley Jones (Minor Milestone)

FR: When the draw function is called, cell data is drawn to the area for the given constraints.
FR: When the draw function calls the cell state function, it will draw white if cell is alive and black if not.
FR: When the draw function is called on ShaderBlock, components will not be redrawn if they have been drawn this frame.
FR: When the draw function is called on ShaderBlock, graphics will be drawn to a buffer.
FR: When a file containing a shader is given to its function, it will be compiled, and its id will be returned.
NFR: The draw function completes within 10 ms for any valid input. [performance]

GUI - Riley Jones (Major Milestone)

FR: When the resize slider is moved, percent of cells displayed out of max is set to percent of resize slider downward position.
FR: When a cell is clicked, the cell thought process is displayed close to the cell
FR: When a mouse is held, enter the drag view state.
FR: When in the drag view state, offset the view by the difference between the current mouse position and the initial mouse position.
FR: When a value from the shader select dropdown is clicked, corresponding shaders are compiled and replace current shaders.
NFR: Any interaction should result in a change by the next frame, except shader changes. [performance]

Save Game – Dennis Balan (Minor Milestone)

FR: When the save game button is pressed, the game will enter the save state and a save game file will be created

FR: During the save state, the game will copy all the values of the relevant variables and objects needed to recreate the game instance into the game save file

FR: When all the relevant values have been copied into the save game file, the game save file will be downloaded on to the desktop off the player’s computer

FR: When a game save file is saved, it will be saved as the save number

FR: After a save game file is saved, the save state ends and the save game file is closed

NFR: The save game file will include the names of variables and objects next to the values that they should hold. [debugging, readability]

Load Game – Dennis Balan (Major Milestone)

FR: When the “Previous Saves” button is pressed, game enters load state, and a new dialog box will ask for a save game file

FR: Once the game save file has been inputted/uploaded, the dialog box will close and the save game file will be opened for reading

FR: While in load state, the save game file should be scanned for all the relevant values to input into any variables and objects needed to recreate the saved game instance

FR: When all the values in the save game file are inputted into the valid variables/objects, a new game instance is constructed based on the saved game instance

FR: Once a new game instance is created, the load state ends, and the game save file is closed

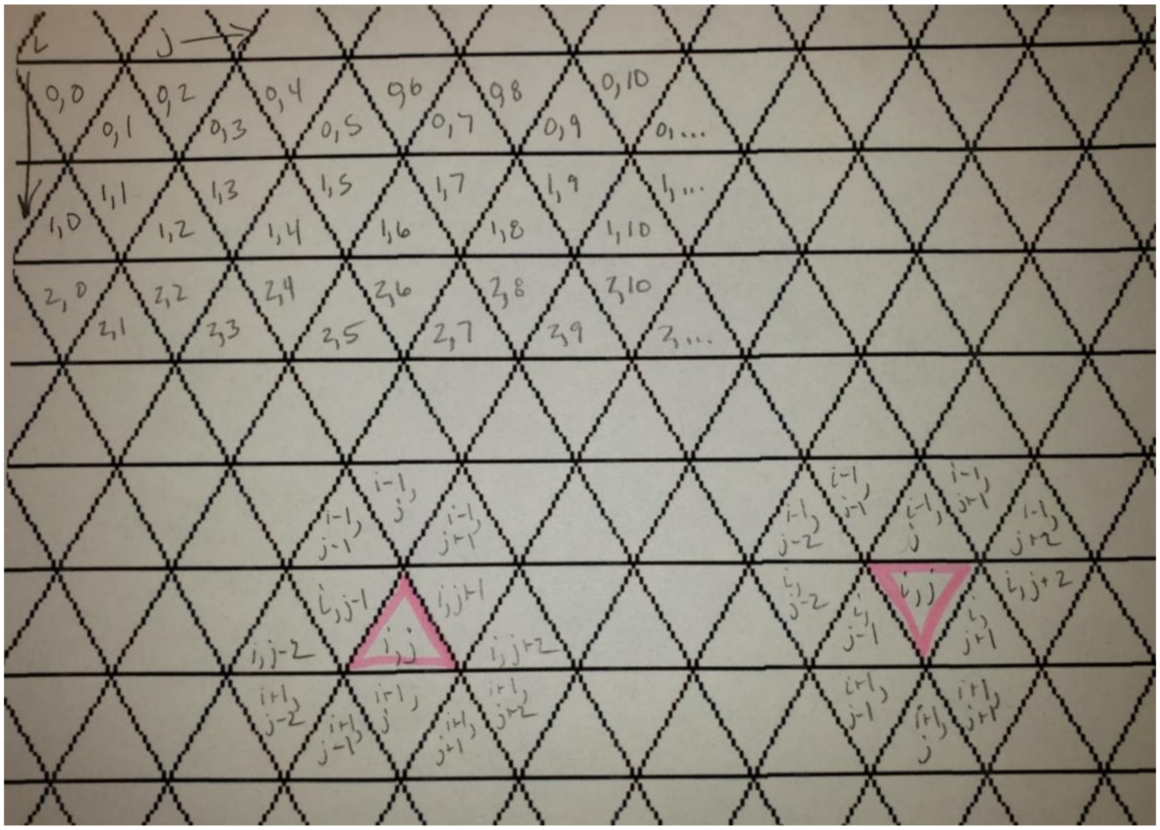
NFR: The save game file name should be checked as a valid name for security purposes and reject any files that do not have a valid name. [security]

Appendix:

A: Grid Layout, Matrix Representation, and Neighbors for Cell Shapes

Triangle:

Triangles occur in two types, an upward triangle (point facing up) and a downward triangle (point facing down). Each type has 12 neighbors, but the specific neighbors differ by triangle type.



Upward Triangle: (occurs when i+j is odd) The 12 neighbors of the upward triangle (i,j) are:

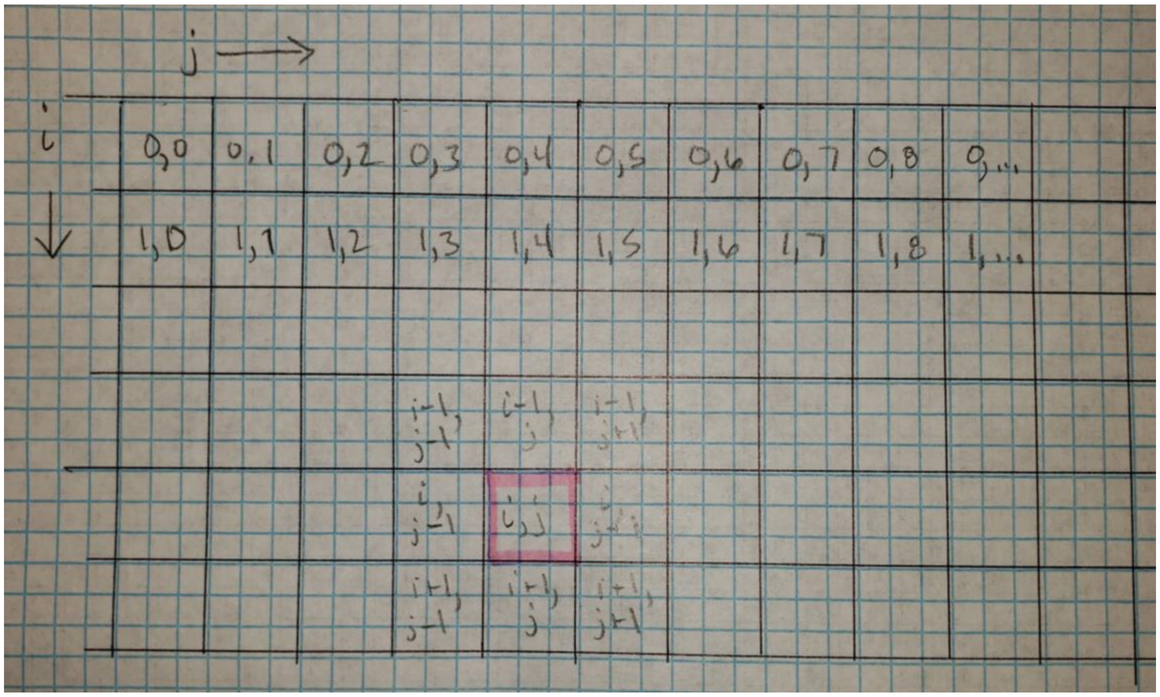
(i-1, j-1), (i-1, j), (i-1, j+1)
(i, j-2), (i, j-1), (i, j+1), (i, j+2)
(i+1, j-2), (i+1, j-1), (i+1, j), (i+1, j+1), (i+1, j+2)

Downward Triangle: (occurs when i+j is even) The 12 neighbors of the downward triangle (i,j) are:

(i-1, j-2), (i-1, j-1), (i-1, j), (i-1, j+1), (i-1, j+2)
(i, j-2), (i, j-1), (i, j+1), (i, j+2)
(i+1, j-1), (i+1, j), (i+1, j+1)

Square:

Squares are the simplest type, existing in only one type and having 8 neighbors.

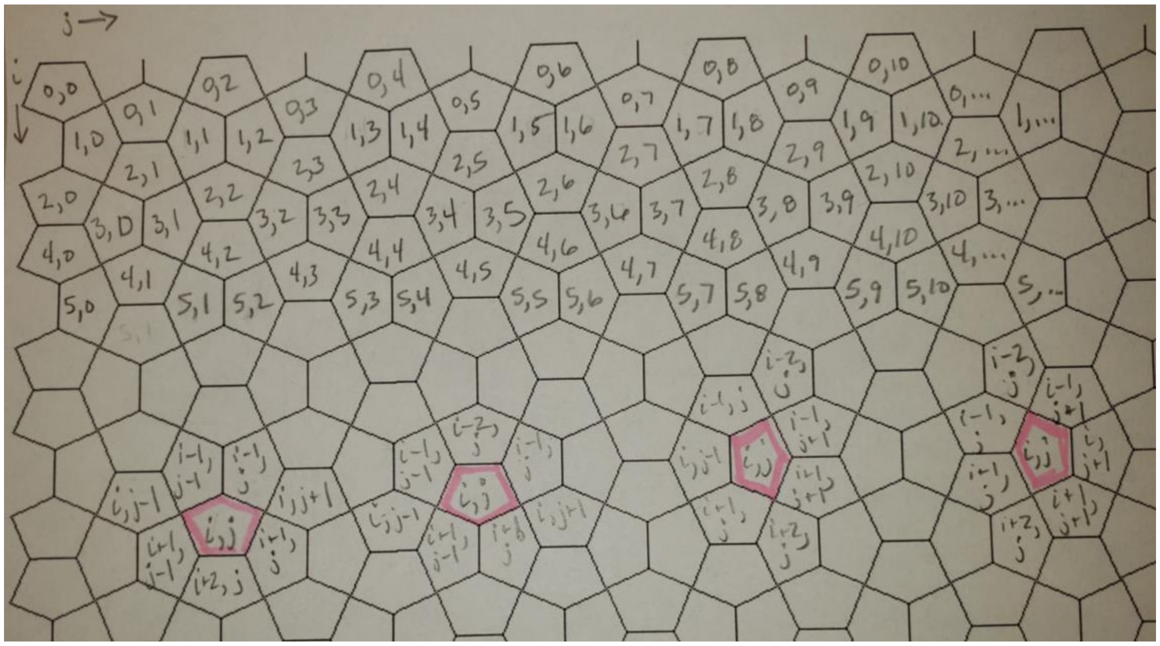


The 8 neighbors of square (i,j) are:

(i-1, j-1), (i-1, j), (i-1, j+1)
(i, j-1), (i, j+1)
(i+1, j-1), (i+1, j), (i+1, j+1)

Pentagon (Irregular):

We will be using an irregular pentagon to create a tessellating pattern. Pentagons are the most complicated shape (that we will be using) to represent as a 2D matrix. The cell pattern is different on each of the first 4 rows and then the pattern begins to repeat. In this pattern, there are 3 distinct pentagon types, we will call them A, B, and C which are diagrammed from left to right on the included image (C appears twice). Each pentagon has 7 neighbors, but the neighbors change depending on which type the pentagon is. Type A pentagons occur when i % 4 = 0 and j is odd or i % 4 = 2 and j is even, type B when i % 4 = 2 and j is odd or i % 4 = 0 and j is even, and type C (pictured twice) when i % 4 is odd.



Pentagon A: (occurs when i % 4 = 0 and j is odd || i % 4 = 2 and j is even) The 7 neighbors of pentagon A (i,j) are:

(i-1, j-1), (i-1, j)
(i, j-1), (i, j+1)
(i+1, j-1), (i+1, j)
(i+2, j)

Pentagon B: (occurs when i % 4 = 2 and j is odd || i % 4 = 0 and j is even) The 7 neighbors of pentagon B (i,j) are:

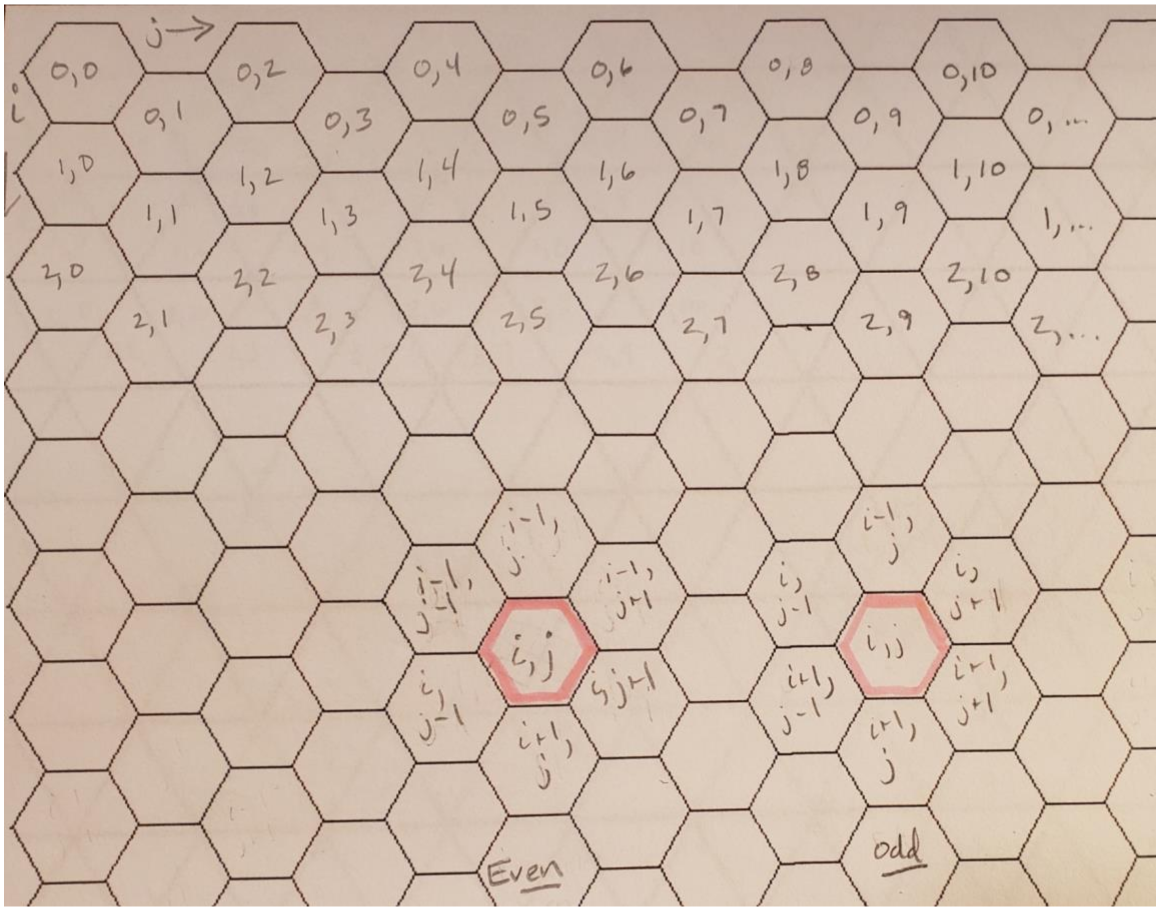
(i-2, j)
(i-1, j-1), (i-1, j)
(i, j-1), (i, j+1)
(i+1, j-1), (i+1, j)

Pentagon C: (occurs when i % is odd) The 7 neighbors of pentagon C (i,j) are:

(i-2, j)
(i-1, j), (i-1, j+1)
(i, j-1)
(i+1, j), (i+1, j+1)
(i+2, j)

Hexagon:

Hexagons have two types, an even hexagon (i+ j is even) and an odd hexagon (i + j is odd). Each type has 6 neighbors.



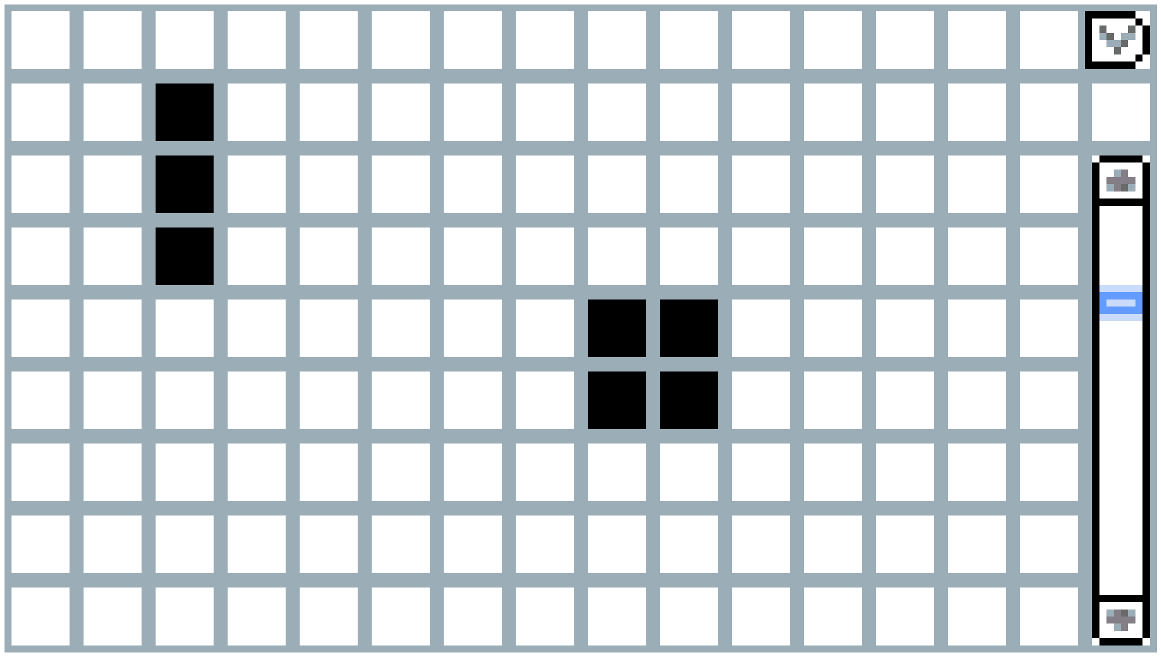
Even Hexagon: (occurs when (i + j) is even) The 6 neighbors of the even hexagon (i,j) are:

(i-1, j-1), (i-1, j), (i-1, j+1)
(i, j-1), (i, j+1)
(i+1, j)

Odd Hexagon: (occurs when (i + j) is odd) The 6 neighbors of the odd hexagon (i,j) are:

(i-1, j)
(i, j-1), (i, j+1)
(i+1, j-1), (i+1, j), (i+1, j+1)

B: Visual Mock-up of Simulator Display



Participation

Justin Flint
Wrote descriptions of FRs, NFRs, and the epic revolving personal cool cam.

Kristine Hess
Wrote rough draft of concept, set up formatting for documentation, and created appendix A, diagramming the grid layout, matrix representation, and neighbors for cell shapes. Was also responsible for personal epic, algorithm implementation, and its feature requirements.

Alex Hong
Wrote the epic description and feature requirements for the personal cool cam of displaying cellular thought processes.

Eric Le
Wrote epic for website implementation including FRs and NFRs of its features.

Riley Jones

Wrote about own cool cam and created appendix B, the visual mockup.

Dennis Balan

Wrote about own epic, the save/load cool cam.