A Multiplayer Derivation of Conway's Game of Life

CS 480 - PA#11 Report

Jordan Andrieu, Deev Patel & Braeden Richards

Monday, December 17th, 2018

Contents

Overview

This project is a 2-player adaptation of Conway’s Game of Life. Notably, it is played on a 3-D board that surrounds the players on all sides. The board is shaped like a cube for simplicity in the default version. This can be changed via the configuration file to make any number of shapes. Notably the project makes use of instancing, ray casting, and realistic movement controls to create a usable 2-player game.

Basic Rules

For the most part, the game’s progression is dictated by the 4 rules of Conway's Game of Life:

1. Death by underpopulation

   Any live cell with fewer than two live neighbors dies.

2. Continuation of life

   Any live cell with two or three live neighbors lives on to the next generation.

3. Death by overpopulation

   Any live cell with more than three live neighbors dies.

4. Reproduction

   Any dead cell with exactly three live neighbors becomes a live cell.

However, since this is a multiplayer game, any cell that comes to life takes the dominant color of its neighbors. So, if a cell is surrounded by two cells of Player 1 and one cell of Player 2, it will belong to Player 1 at the end of the round. Additionally, before each round, players are given the ability to either mark 2 of their cells for death & 1 dead cell for life or destroy 1 opposing cell.

Dependencies & Building

This project is built using cmake. To build and run the program in its basic form, go to the project source directory (PA11) and run the following.

```
mkdir build
cd build
cmake ..
make
./PA11_FinalProject
```

One can also the build/run the project with different settings as shown below. Note that all the parameters/variables are shown with their default values. Only change them if you know what you are doing.

```
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
make
./PA11_FinalProject -l launch/DefaultConfig.txt
```

The project was made with OpenGL 3.3, but it should work with most newer versions. In addition, the following external libraries are required. Note that everything was built and tested on Ubuntu 18.04. As such, all installation instructions are meant for Ubuntu 18.04.

- **Assimp** - Open Asset Import Library
  - More Info: https://github.com/assimp/assimp/wiki
  - Installation: `sudo apt-get install libassimp-dev`
  - Primary Usage: loading models of all the objects

- **Bullet**- Bullet Physics SDK
    - More Info: https://github.com/bulletphysics/bullet3
    - Installation: `sudo apt-get libbullet-dev`
    - Primary Usage: detecting collisions during ray cast for clicking on game elements
- **GLEW** - OpenGL Extension Wrangler Library
    - More Info: http://glew.sourceforge.net/
    - Installation: `sudo apt-get install libglew-dev`
    - Primary Usage: Implementing various part of OpenGL graphics pipeline
- **GLM** - OpenGL Mathematics Library
    - More Info: http://glm.g-truc.net/0.9.7/index.html
    - Installation: `sudo apt-get install libglm-dev`
    - Primary Usage: math related with rendering and movement
- **Magick++** - ImageMagick C++ API
    - More Info: http://www.imagemagick.org/Magick%2B%2B/
    - Installation: `sudo apt-get install libmagick++-dev`
    - Primary Usage: loading textures for game elements
- **SDL2** - Simple DirectMedia Layer
    - More Info: https://wiki.libsdl.org/FrontPage
    - Installation: `sudo apt-get install libsdl2-dev`
    - Primary Usage: window creation & user interactions

Note: One can install all the necessary dependencies at once.

```
sudo apt-get install libassimp-dev libbullet-dev libglew-dev
libglm-dev libmagick++-dev libsdl2-dev
```

User Manual

Controls

- Game Controls
  - c: switch to selection mode - can now click on cubes and interact with board
  - g: move one generation forward
  - p: switch turn to next player. Game will progress one generation after Player 2
  - enter/return: begin autoplay mode (only works while in single player)
  - n: switch between a multiplayer and singleplayer game
  - r: reset the board
- Camera Movement
  - Keyboard
    - w/s/a/d: move forward/backward/left/right
    - space: move up
    - l-Shift: move down
  - Mouse
    - look around (left/right/up/down) without moving your position
- Lighting - Global
  - +/-: adjust ambient lighting
- Other
  - t: toggle (open/close) IMGUI menu window
- Exiting
  - ESC: properly close all windows and exit program

IMGUI Menu

There is an IMGUI (https://github.com/ocornut/imgui) menu that runs in a separate window from the main game window. At the top, it shows the current camera position, and focus point. The user has the option to change these values by typing in any number desired. Note that one must press the update button for the new position and focus locations to go into effect. This is helpful for the user to keep track of where they are in the game and move as needed. The menu can be toggled (opened and closed) via the *t* key. An example of what the menu looks like can be seen in Figure 3 in the screenshots section.

Terminal & Progression

The terminal shell from which the game was launched updates periodically with information about the game. Notably, it prints out the generation number of completed simulations. This way, it is easy to tell when the game has progressed. When in multiplayer mode, the terminal prints out whose turn it is. This way, the players know how the game is going and can use the ‘g’ and ‘p’ keys effectively.

The terminal window can also be used the debug the game application. In order to do this, the entire application must be made in debug mode via the cmake -DCMAKE_BUILD_TYPE= Debug command. When this is done, various behind-the-scenes events are displayed to the terminal. This is especially helpful in seeing how the ray casting for clicking is done. At the start, the terminal will show all the colliders being created. Then, when the mouse is clicked, the terminal will shows the results of the ray cast and print the location(s) of any collision.

Screenshots of Game

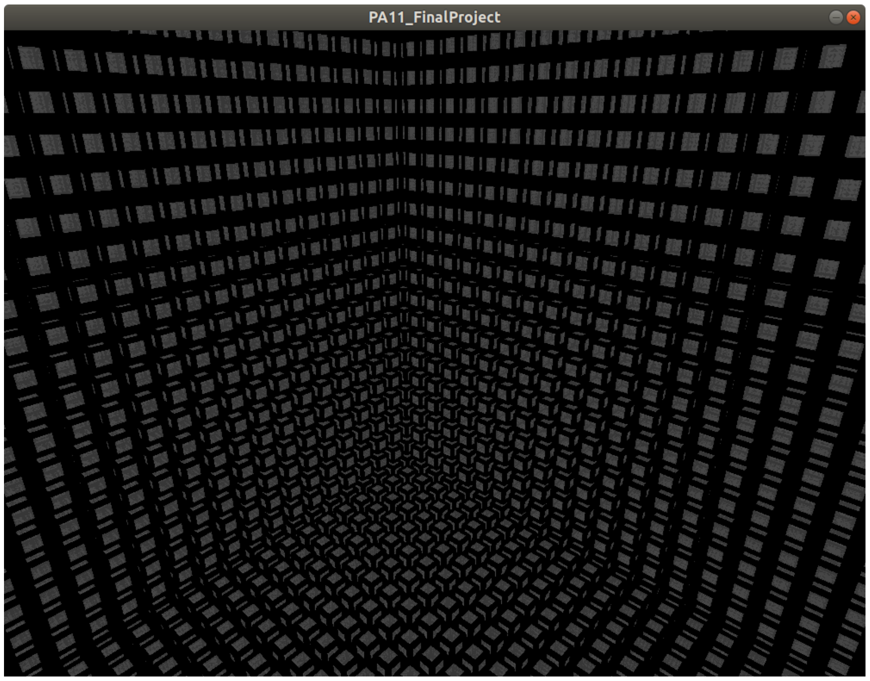


Figure 1: View of empty 3-D game board from top corner.

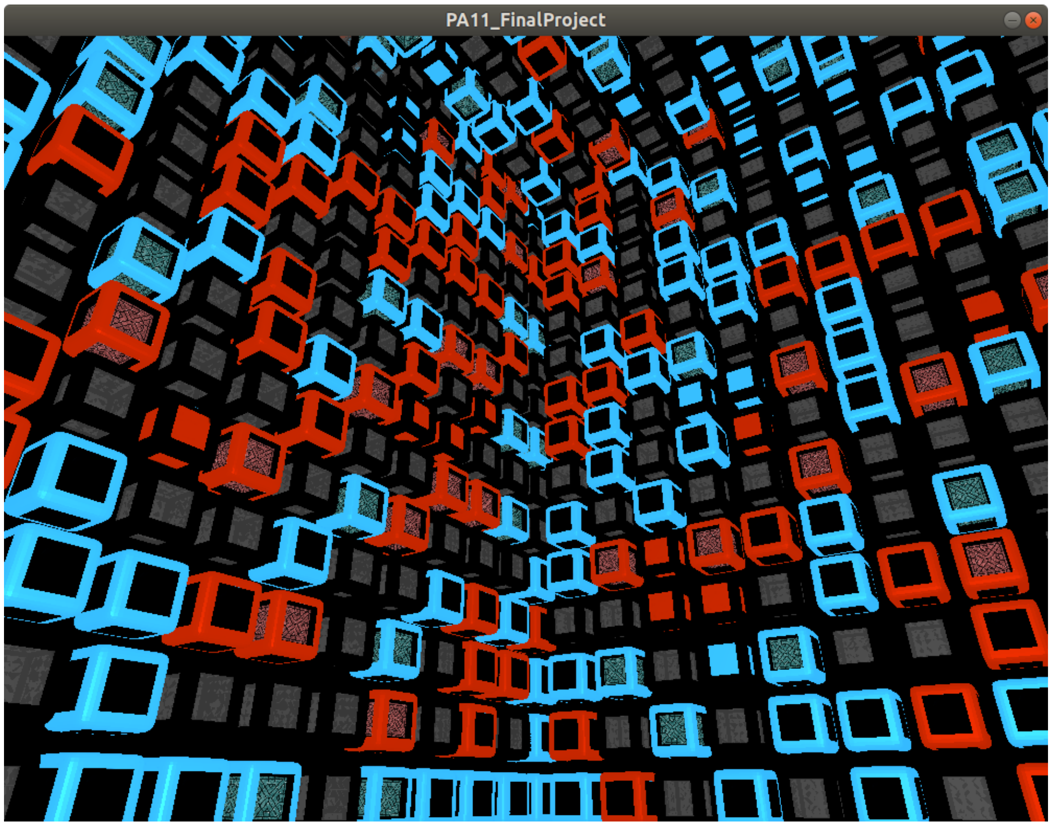
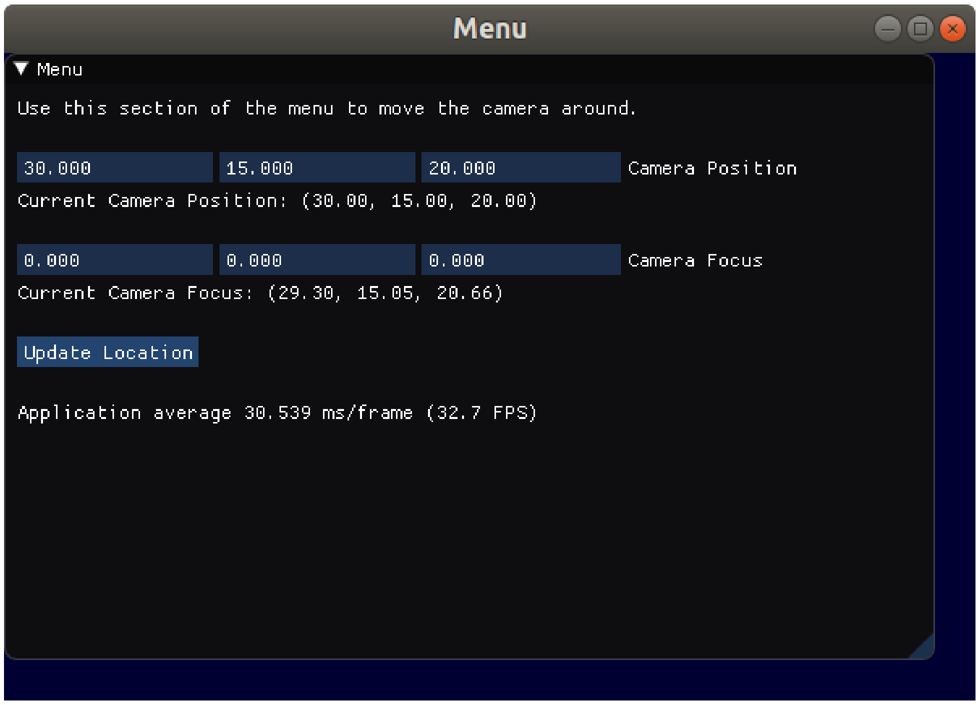


Figure 2: Random 2-player setup for game



```
Use this section of the menu to move the camera around.

30.000          15.000          20.000          Camera Position
Current Camera Position: (30.00, 15.00, 20.00)

0.000           0.000           0.000           Camera Focus
Current Camera Focus: (29.30, 15.05, 20.66)

Update Location

Application average 30.539 ms/frame (32.7 FPS)
```

Figure 3: IMGUI menu

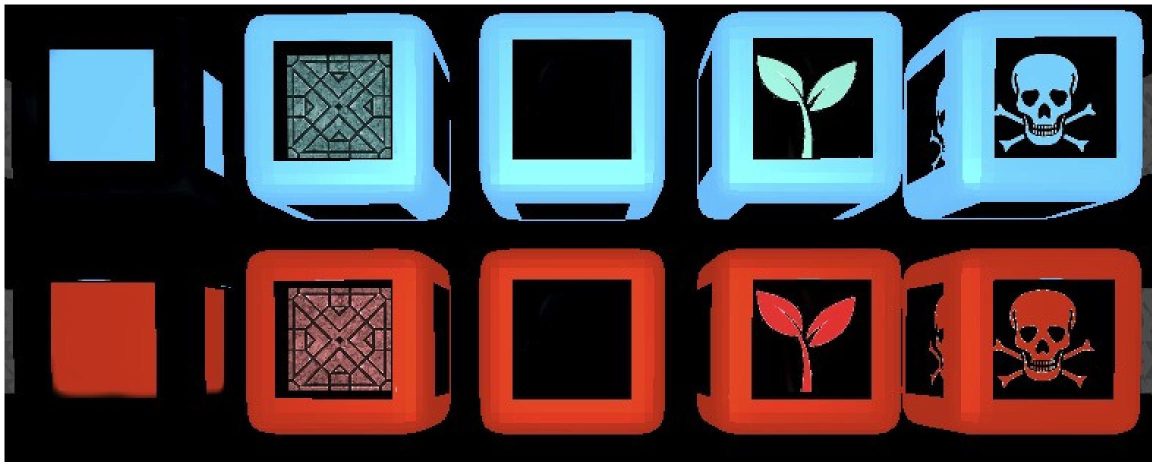


Figure 4: All the special textures a game element can take. Top row shows Player 1 textures. Bottom row shows Player 2 textures. Meaning from left to right: Alive in future, currently alive, dying, marked for life, marked for death.

Technical Manual

Issues

In its current version, the game has few different issues. The most prominent of these is an inability to undo actions or perform them in a different order. In the multiplayer mode, if a player accidentally clicks or selects a cube, they cannot undo their selection. Additionally, a player must perform their actions in the required order (select two of their own cells and then a dead one) or the game will not work. This behavior is not really ideal for a good a game because there are often misclicks, especially when camera rotation is involved. Another related issue with the current version of the game is an inability to move the camera and select an object at the same time. As a result, we had to add the ‘c’ button to switch between clicking and movement modes. Ideally, these two things should be able to be done at the same time. However, in order for our mouse movement calculations to work, the mouse has to be in the center. Since this interferes with clicking, we had to create the two separate modes.

Another issue that is not so obvious stems from the 3-D nature of the game board. Specifically, calculating the neighbors of the game elements along the edges of the board is difficult and the neighbors of certain elements (the corners) had to be hard-coded in. This is an issue because it could affect the game if the board is not changed appropriately. Ideally, we should be able to create any sized/shaped board. But, because finding the neighbors at the corners is hard, the board shapes are limited to cubes/prisms. Thus, the game board is not as scalable as we originally intended.

Things We Could Have Done Differently

There are a few different things that we would have done differently given more time. One of the biggest changes would have included a switch from Euler angles to quaternions when doing the camera movement calculations. We currently use Euler angles to calculate how the camera should rotate when the user moves the mouse. However, looking back, it may have been better to use quaternions to prevent certain weird issues, such as those stemming from gimbal lock. While using quaternions is more computationally complex, it would no doubt produce better results. Another issue regarding movement that we could have addressed is to remove the ability of the player to move outside the board. This could have been implemented by adding checks on the current position based on the dimensions of the board.

Another major change that we could have implemented would have come in regards to the IMGUI menu. Currently, the menu appears in a separate window and does not play a major role in the game. Having the menu appear on top of the main window would have allowed us to make more use of it in the game. For instance, it could have been used to show the generation cycle and display whose turn it is. We could also have added a scoreboard on the menu. The only reason we did not do this is because we could not figure out how to put the menu atop the main window without disrupting the OpenGL rendering of the objects beneath it. Given more time, we could have figured out the exact issue and made better use of the menu. This would have improved the game play quality. Moreover, the players would have more clarity in regards to the progression of the game.

A final change that we would have liked to implement is the ability for players to undo a mouse click. This would make the game more realistic since players would be able to “undo” misclicks.