final_project – pyrokuna (Ryan Gutierrez)

When producing the source code for the final project (my take on space invaders), I was able to properly implement nearly every task as I had planned. I decided to make the way enemy sprites animated across the screen different than the way I proposed it and I was unable to implement part of the collision detection task—both of which will be detailed in their designated parts. To compile this assignment, you must use visual studio 2019 with CMake GUI: set the source directory to be the directory delivered on github, and the build directory to be a directory you create named "build" which can go wherever you want. Then configure, generate, open project, build the project, and the executable should appear in the build folder. One possible bug that doesn't belong in any of the part descriptions is that I may have possible memory leaks but unfortunately have no time to or no way of checking given my current knowledge/resources.

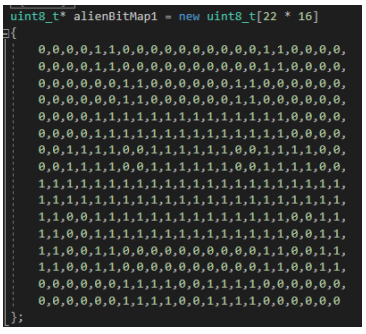
Dependencies

For this project, I only used OpenGL along with GLFW and GLEW with all the code written in C++. Not that I used the starter code provided to us for projects 2 and 3 as a base when creating this project (only I removed everything that had to do Eigen and edited the vertex fragment shaders).

Gameplay Instructions:

Use space bar to fire and the left and right keys to move side to side.

1.1 Drawing Sprites



```
uint8_t* alienBitMap1 = new uint8_t[22 * 16]
{
    0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,
    0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,
    0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,
    0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,
    0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,
    0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,
    0,0,1,1,1,1,0,0,1,1,1,1,1,1,0,0,1,1,1,1,0,0,
    0,0,1,1,1,1,0,0,1,1,1,1,1,1,0,0,1,1,1,1,0,0,
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,
    1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,
    1,1,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,1,1,
    1,1,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,1,1,
    0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,1,0,0,0,0,0,0,
    0,0,0,0,0,0,1,1,1,1,0,0,1,1,1,1,0,0,0,0,0,0
};
```

```
uint8_t* ufoBitMap = new uint8_t[32 * 14]
{
    0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,
    0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,
    0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,
    0,0,1,1,1,1,0,0,1,1,1,1,1,0,0,1,1,1,1,1,0,0,1,1,1,1,0,0,
    0,0,1,1,1,1,0,0,1,1,1,1,1,0,0,1,1,1,1,1,0,0,1,1,1,1,0,0,
    0,0,1,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,1,0,0,
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    0,0,0,0,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,
    0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,
    0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,
    0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0
};
```

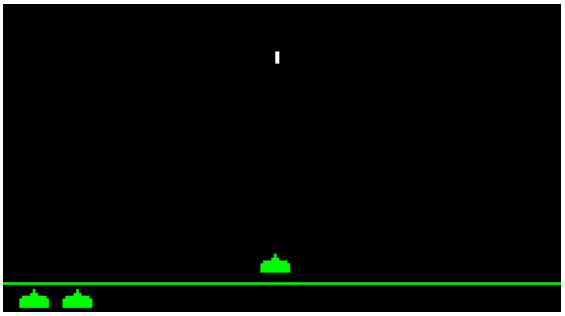
For this task—which I implemented fully—I was able to draw sprites on a by-pixel level. To do so, I created unsigned-8-bit integer arrays to store bit maps (where a 1 meant an on pixel and 0 meant an off pixel) which were then inserted onto a buffer that took up the entire screen. The

buffer was then sent to the vertex shader and fragment shader to be drawn on the window. Note that all sprite information was stored in sprite objects for which I created a class.

1.2 Sprite Animation

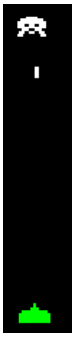
For this task—which I implemented fully but changed the way I had initially intended to do it—I created alternate sprites through the same bitmapping technique used in part 1.1 in order to accomplish the "stationary" animations (in which the enemies change appearance every 40 "frames"). I also animated the UFO from the original game every 1,000 "frames" which I intended to be random but ran out of time. Also, initially I wanted to have the enemies travel down towards the player but due to a lack of time I settled for having them only animated side to side across the screen. Finally, I also created a death sprite which is supposed to look like an explosion that shows for ten "frames" after an enemy or the UFO has been destroyed.
*The reason I put the word "frames" in quotation marks is because my method of simulating frames uses the function glfwSwapInterval(1) which depends on the framerate of the current monitor/computer as I understand it.

1.3 Player Input



For this task—which I implemented fully—I used a key call back function (as we have done in the class for all our past projects) to move the player sprite across the screen from left to right (using the left and right keys) and to have the player sprite shoot (using spacebar). Something to note here is that in trying to create a delay between shots, I found that it is possible to queue up one other shot and I could not figure out why that was happening, which could be considered as a bug or an incidental feature.

1.4 Collision Detection



For this task—which was technically fully implemented properly—I came up short in some respects. I was able to implement collision detection as a whole (through a function in which two sprite objects were sent), but only did it for player bullets hitting enemies. The reason I say I technically completed this section is because collision detection works properly and is independent of who is shooting the bullets. Ultimately, I did not have the time to have the enemy sprites randomly fire at the player and could not check for bullet to player or bullet to bullet collision detection.

1.5 Cosmetics

For this task—which I implemented fully—I was able to add all the cosmetic enhancements I had originally intended to include: a live score and high score (both drawn with bitmaps like the other sprites in the game) that is stored in a file and loaded when a new game is started. Since I did not have the time as I said I would need in my proposal, I did not attempt to implement a start screen or the option to clear the high score—which you can still do by editing the file or deleting it.

*Note for Arthur

Since professor Gao had approved my overview prior to your response to it, I noticed your response much later than I would have liked to and didn't have time to implement any of your recommended additions. I apologize but I figure it shouldn't be that troubling since Gao said my proposal was fine.