**GROUP FINAL SUBMISSION**

**GROUP #3**

Group Members:

* Salvador Ochoa
* Kelvin Lok
* Cristian Cortez

1. **OpenGL Installation: Setup for Our Project**

For the installation of OpenGL, we used many sources including following the instructions in the book *Computer Graphics Programming: In OpenGL With C++*. We also used YouTube tutorials to help with the install including:

* [Setting Up OpenGL in Visual Studio 2017 C++ | Fresh Install 🡪OpenGL Window](https://www.youtube.com/watch?v=k9LDF016_1A&t=787s)
* [Setting Up OpenGL and Creating A Window in C++](https://www.youtube.com/watch?v=OR4fNpBjmq8)

To compile and run our program, the user needs to place the “*glew32.dll*” within the same file where the “*projectName.exe*” is.

It is also important to place the “*fragment\_core.glsl*” and “*vertex\_core.glsl*” files within the *resource files* in the Solution Explorer for the project in Visual Studio. If, after placing the “*.glsl*” files within the resource files the program does not compile or errors are shown, the user might need to recreate these files.

To recreate these two “*.glsl*” files, add two new items in the resource files with the same name as “*fragment\_core.glsl*” and “*vertex\_core.glsl*”, with the same extension. Now copy and paste the information of each file into the two newly created files with the same name respectively.

1. **Algorithm and Steps Leading Towards Implementation:**

Our OpenGL project begins by using a *struct Vertex* data structure that calls the *vec3* and *vec2* from *glm*. Each vertex within the Vertex Array has a *vec3 position, vec3 color,* and *vec2 texcoord.* This struct is declared within the *Vertex.h* header file, where it is later initialized within the *Vertex.cpp* source file. These two files are used to store and build each object dimensions within the Vertex Array. Our Klotski game uses 13 objects, where two are dedicated for the board, 10 for the blocks, and one an exit block.

Once the 13 objects are built, the *movement.h* header file declares functions for movement, selecting, collision and draw. Briefly, movement requires a block to be selected depending on the position of the mouse. If the mouse Xposition and Yposition is clicked when within the bounds of a block, then that block’s vertices are saved. When the mouse button is released, there is a check using a Boolean to see if the new space is empty, and if yes, then the block moves to the new space the where the mouse X and Ypositions are. When the block is moved, the whole bard is then redrawn. This process requires functions for selecting a block, checking direction moved, if the space is empty, 4 functions for moving in either up, down, left and right and redraw.

Next, as the block is moved, the previous location of the block (original location) is stored using a stack. This stack can be found in the *MovementStack.h*. This movement stack requires two structs; one struct for a StackNode and the other for Stack that has StackNode data, int index for the index of the block object and Stack next, members. This movement stack is used for undoing a move which is mapped to the backspace button.

All these files are then used to implement a *driver.cpp*.

**INSTRUCTIONS:**

* + To move a block, click and hold on a block and drag to next available open space. Let go of the click to move the block. The game will then redraw with the updated block positions.
  + To undo a move, press the “backspace” button on the keyboard.
  + To reset the game press “r” on the keyboard.

1. **Screenshots:**

Start:

A picture containing monitor

Description automatically generated

*Starting Position of Klotski Game Board.*

Before / After undo:

A picture containing monitor

Description automatically generated

*From left to right: 2 Moves In, Undo Once, Undo Twice (Start).*

Before / After reset:

A picture containing screenshot

Description automatically generated

A screenshot of a computer screen

Description automatically generated

*From Top to Bottom: Reset Before (With 4 Moves In), Reset After (Circled on Console).*

Winning Board / Console:

A screenshot of a cell phone

Description automatically generated

*Note: The console prints “Congrats you win” within the game loop, so it never ends.*

1. **Implementations / Not Implemented:**

In our project we have implemented movement, collision, undo, and reset.

What we have not implemented was move count, textures, menus / buttons, and display winning window.

1. **Completing the Not Implemented:**

To complete the non-implemented parts, we would need access to other functions found within the glm, glfw, gl and SOIL libraries. For example, to use textures for buttons, we would need to call on functions specific to importing and updating textures.

1. **Difficulties / Issues / Help:**

One major issue we had with this project was selecting the block. Particularly, our group had difficulties with handling the different sizes of the blocks. There are 10 block objects that have three different sizes: 4 blocks are 1x1, 5 blocks are 1x2 and one block is 2x2. Including in these 10 blocks are two blocks used for the game board, and one block used for the win condition. The algorithm for selecting a box is briefly as follows, check if mouse cursor Xposition and Yposition are within the vertices of a block and if they are, select the block.

This proved difficult because of the differences between the vertices of each block. Each block has specific vertices that are unique to it, making binding the mouse cursor to the block difficult. A way to fix this is to assign an index to each block. Using an index means that when a check is done on Xpostions and Ypositons of the mouse cursor and vertices, then there is assurance that the correct block is selected. This index can be implemented by finding the beginning index of the block’s 4 vertices within the vertex array. Since each block as 4 vertices, then finding the first vertex’s index within the array would allow us to add 1 to the index to find the next vertex and so on until we find 4 vertices.

To help with this issue of selecting a block, our group members used internet searches and had discussions with other groups to compare the functions used. Combing multiple examples found online and from these groups helped to form proper functions for the block selecting. Specifically, using the idea of finding the blocks first vertex index to identify the block.

1. **Additional Features:**

Some features we would like to add to this game would be implementing sounds and voice overs. For example, we would like to utilize sound effects when blocks are moved or play a jingle when the game is won. Or perhaps implementing a system that plays a “good” sound when a move is correct and a “bad” sound when a move is incorrect. However, upon research, there is no sound compatibility designed in the current install of OpenGL. We would need an additional installation of a platform sound engine called Irrklang which would allow sound implementation.

Another feature we would like to include would be a real time self-solver algorithm. This specific board design of Klotski can take over 110 moves to win. If the player begins to lose interest or wishes to see the solution, there should be some way within in the game design to give the player hints or solve the entire puzzle.

1. **Future Features:**

Some futuristic features that we would like to add would be the addition of Augmented Reality. Specifically, we could look at how to implement this Klotski game so that it can be projected from our phones, through the phone screen, on to a surface. Imagine Pokémon Go but instead Klotski Go.