# CSE 165 Final Project Report

Hunter McClellan Higinio Ramirez Kevin Zheng

May 8, 2021

## 1 Project Description

Our project is recreating the famous Tetris game in 3D. Rather than clearing "lines", the player is expected to clear layers. The pieces will be reminiscent of the original pieces, since most of them can be quite naturally generalized.

## 2 Member Contribution

The contribution can be roughly be separated by file as as

- Hunter McClellan
  1. `mainwindow.h`
  2. `mainwindow.cpp`
  3. `Ground.h`
- Higinio Ramirez
  1. `Points.cpp`
  2. `Randomizer.cpp`
- Kevin Zheng
  1. `Tetris.h`
  2. `Tetris_Graphics.h`

## 3 Implementation

### 3.1 Tetris.h

Handles all game logic

- `GameState` enum: has only two members, `PLAYING` and `LOSE`. Sorry, you can't win this game!
- `Moves` enum: has eight members, `DOWN`, `LEFT`, `RIGHT`, `FORWARD`, `BACK`, `PITCH`, `ROLL`, `YAW`.
- `Block` struct: has three integers for red, green, blue which range from 0 to 255. The `Block` struct also has one bool that indicates whether or not the block is falling. Falling in this context is synonymous with "in play." Note that the location of the `Block` is not handled by the `Block`, it is handled by the `Tetris` class.
- `Tetris` class: has a quadruple `Block` pointer denoted by `state`. This can be best understood as three dimensional array of `Block` pointers. This number of pointers is necessary, since want to allow parts of the array to be `NULL`. There are three integers for width, length, and height of the playing field.
  - `ind2sub(int ind, int &x, int &y, int &z)`. Short for "index to subscript". This method accepts a number from 0 to `w*l*h-1`, and "returns by reference" the corresponding `x`, `y`, `z` coordinates. This is used in the cases where we want to loop through every `Block` in `state`, but would rather not use a triple for loop. Instead, we can make one loop, and use `ind2sub`.
  - `int sub2ind(int x, int y, int z)`: This is the reverse operation of `ind2sub`.
  - `GameState control(Moves moves)` accepts a member of the `Move` enum and modifies the `state` accordingly. If the control made by the player results in a loss or not will be reflected in the `GameState` return. The body of this method is simply a `switch` statement calling either `translate_piece` or `rotate_piece`.
  - `GameState translate_piece(Moves move)` handles the translation of pieces. This is called by `advance` as well as `control`
  - `void rotate_piece(Moves move)` rotates the piece. Note that the return is `void` since a rotation can never cause a loss.
  - `void handle_layer_clear()` Checks for cleared layers, and remove them.
  - `GameState spawn_piece()` attempts to spawn in a random piece at the top of the playing field. Failure to due so is the definition of a loss, and such will be returned.
  - `GameState advance()` is an alias for `translate_piece(DOWN)`, as well as a `handle_layer_clear()`.

### 3.2 Tetris_Graphics.h

Has a constructor which accepts a `Tetris*`. Has a `void draw()` method which draws the Blocks in `Tetris`.

### 3.3 mainwindow.h/mainwindow.cpp

Handles keyboard input, mouse input, and graphics. Has class `MainWindow` which extends `QOpenGLWindow`. Implementes the generic methods of `QOpenGLWindow` like in the labs. Adds a `SLOT` for the method `GameAdvance()` which is an alias for `Tetris.advance()` which is bound to a `QTimer` for once a second.

- `void paintGL()`. Starts by clearing the `GL_COLOR_BUFFER_BIT` and the `GL_DEPTH_BUFFER_BIT` and calls `glLoadIdentity`. Then we do a `gl_Translate(0.0, 0.0, -35.0)`, which elevates the camera by `35.0`, this is so the player has a somewhat downward facing view of the playing field. Then we call `glRotatef` for each direction. Then, we can call `Tetris_Graphics.draw()` and `Ground.draw()`, to draw the ground and the game pieces. Afterwards, the controls are handled by checking for if the `W,A,S,D`, $\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$ keys have been pressed, and then make the corresponding call to `Tetris.control()`.
- `void keyPressEvent(QKeyEvent *event)` checks which of the `W,A,S,D`, $\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$ keys have been pressed, and then sets the corresponding member boolean to be true. The escape key calls `qApp->exit()`to exit the game. Note that these member booleans are also set to false after `paintGL` as to prevent the game from doing the move twice before release.
- `void keyReleaseEvent(QKeyEvent *event)` does the opposite of `keyPressEvent`.
- `void mouseMoveEvent(QMouseEvent *event)` modifies the `cam_x_r, cam_y_r, cam_z_r` according to the position of the mouse. This method also resets the position of the mouse to the center. This is to prevent the mouse to leave the window as the player moves the mouse for a different perspective. To regain normal use of the cursor, exit the game.

### 3.4 Points.cpp

### 3.5 Randomizer.cpp

## 4 Lessons/Conclusions

### 4.1 Depth Bit

The depth bit, or depth buffer, is a noter Not all systems will allocate space for the depth bit.