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 Ramierz
  1. Point System
- 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

4 Lessons/Conclusions