**CS342 (TROY) Project 5**

**Design Documentation**

**Section 1:**

**Introduction and Purpose:**

Names: Jesus Solorzano, Omaid Khan

NetID: jsolor2, okhan23

Email: [jsolor2@uic.edu](mailto:jsolor2@uic.edu), okhan23@uic.edu

**Project: Tetris v1.0**

We are creating a Tetris game in Java using Java and Swing. The purpose of the project is to see how many components really make up the game itself, and how one may go about designing the internals of an apparent simple game. In case you don’t know how the game works, you have to rotate falling pieces called tetrominoes so that they fit snugly with other pieces. You have to fill an entire row with these pieces so that it vanishes and you gain some points. The amount of points you gain on each row depends on the current level that you are plying on. As you progress in levels, the pieces fall faster and faster and the game becomes more challenging. Also, if a piece that has already been placed down appears like its floating because the row underneath it has been cleared there must be some sort of implementation of gravity that will pull the piece down. Gravity must always work even if the piece is rotating. Users can rotate the Tetrominoes when playing, in order to adjust the blocks to fit into the grid.

**Section 2:**

**High Level Entities:**

**Play area:**

**Grid**: The meat and potatoes. Shows the collection of Tetrominos dropped and the current tetromino that must be dropped.

**Fall timer**:This controls how fast the pieces fall. As the level progresses, the pieces fall faster, until a point (level 24 as in the write-up)

**Side Panel:**

**Time Elapsed:** The amount of time from the start of the game

**Level:**  The current level that the user is playing.

**Score:** Number of points he or she has accumulated.

**Next Piece**: The next randomly decided piece to fall into the playing area.

**Tetromino:**

All information about a Tetromino is contained in this class including rotation and shape of the tetromino.

**Menu:**

Allows the user to start and stop the game, and also displays a help menu that shows the user how to play. Also there is an about menu that displays some information about the programmers.

**Section 3:**

**Low Level Design:**

**Class Diagram to show how everything is related:**

D:\Dropbox\Downloads\Omaid and jesus UML.png

**Play area:**

**Fast Fall:**

We will extend Threading or implement Runnable to make this happen. This piece of code has to be always running so that the game can run smoothly with other logic.

**Grid:**

The user will use the up, down, left, and right arrow keys to manipulate this area. Alternatively, the user can use the WASD keys to rotate the pieces. After a piece has been set, it should no longer be able to move unless gravity pulls it down.

**Side Panel:**

**Level:**

The write-up states that after every 10 lines are cleared, the user advances a level. When this happens, the speed of the falling tetrominos for that level will increase by some amount to make it more difficult to play. In addition to affecting the speed of the falling tetrominos, the user will also accumulate more points for clearing a row in a higher level, so that the satisfaction of accumulating points is greater.

**Score:**

The writeup says that the score gets updated by the formula (40 \* current level). This is always displayed in the side panel

**Menu:**

Menu will end up just being added to the frame as a separate component. It will extend JMenuBar. This is an independent component to the program. Only exception to that is the ‘start game’ button which must call another method inside of the gameplay class.

**Tetromino:**

There must be some logic to handle the rotation of a piece in both directions. The user can opt to use the keyboard or WASD keys to rotate the piece either clockwise or counter-clockwise. We plan on using images provided by the TA to display this piece onto the screen, but if that does not work, we may take advantage of GridLayout to place the different pieces on different sections of the invisible ‘grid’.

**Section 4:**

**Benefits/risks/assumptions:**

**Requirements that may not be so obvious:**

1. We have to make sure that we are able to make the distinction between the time elapsed timer and the fall speed timer. When the fall speed timer is made short and shorter, it should not have any effect on the time elapsed timer speed.
2. We do not yet know how to handle rotating pieces that fall off of the grid.
3. We have to make sure that the game class can communicate with the side panel to make sure that the level and score gets updated correctly.

**Benefits of the design:**

1. All variables have the proper access type so that there is no accidental changing of variables
2. Classes are independent of each other. Only method calls should be accessing other classes
3. Variable names make it very easy to follow and read the program. It also helps to simplify the design so that the programmer knows exactly what needs to be done.