Project #5 Design Document: Tetris
A Java Program by Richard Kim and Levi Shusterman
Spring 2016

Overview/Purpose

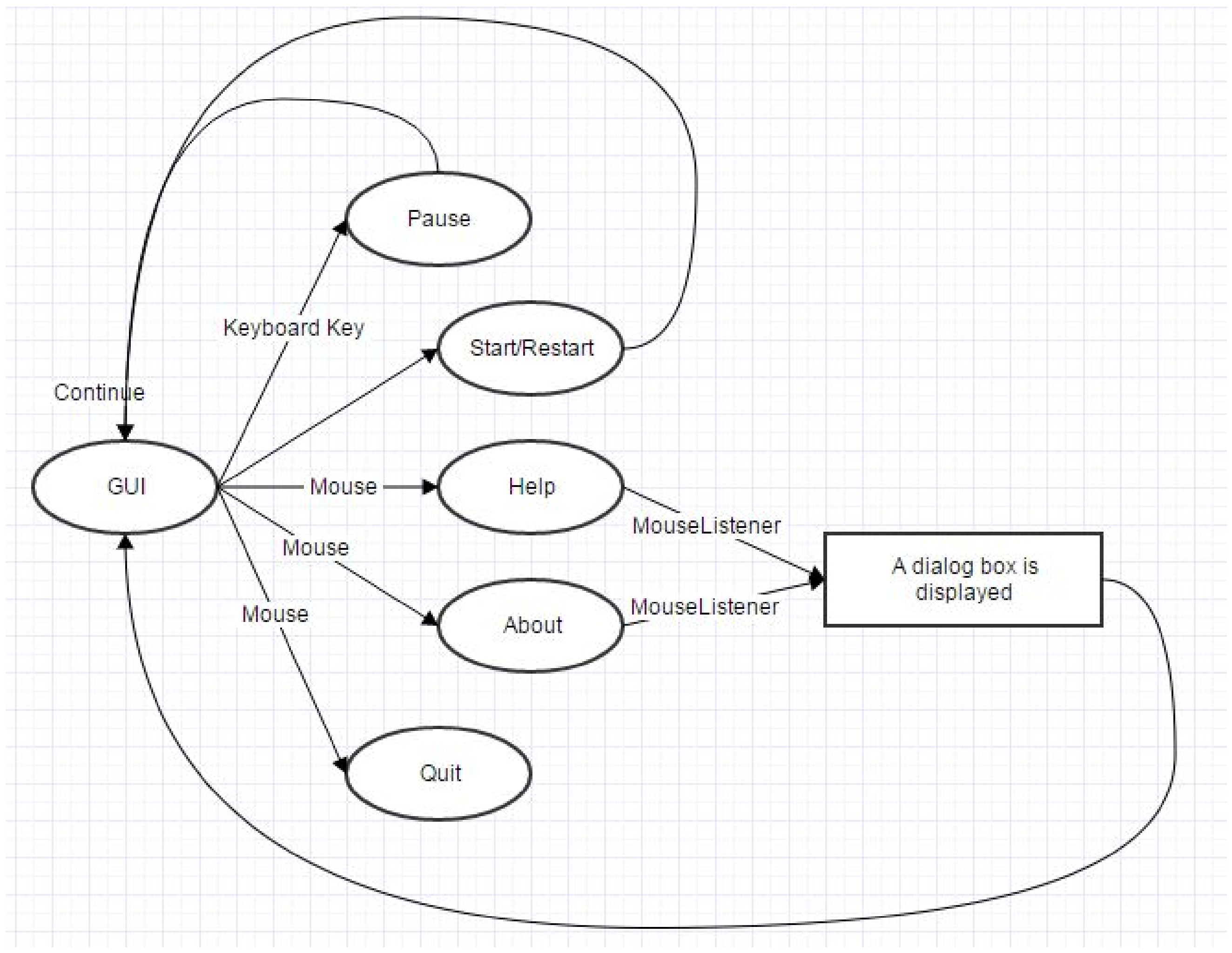
The main focus of this program is to create a version of the game *Tetris*.

The game of *Tetris* is played individually. The objective of the game is to secure as many rows as possible by dropping pieces called Tetrominoes onto a pile. These pieces will continuously spawn at the top and will continue to do so one at a time. This repeats until a stack of blocks reach the top row.

There are 7 types of shapes and they are randomly generated each time. Once the piece starts the descend, the user has the option to rotate the piece clockwise or counterclockwise. This can be controlled by pressing the arrow keys on a keyboard. They also have the option to drop the block quicker than the default speed. If the user wishes to pause the game, the keyboard key is ‘p’ for that command. As the game progresses, a timer keeps track of the time, but also dictates how each piece progressively descends. One second relates to a piece moving down one tile. The scores are assigned to the player by how many lines they can clear. As the user clears more and more lines, the difficulty of the game should increase. It can be more challenging and engaging for the user as time goes on, but the replayability of the game Tetris, is a bit limited. Finally, the game will end, as mentioned before, when a stack of blocks reach the top row or when a block is no longer able to spawn at the top.

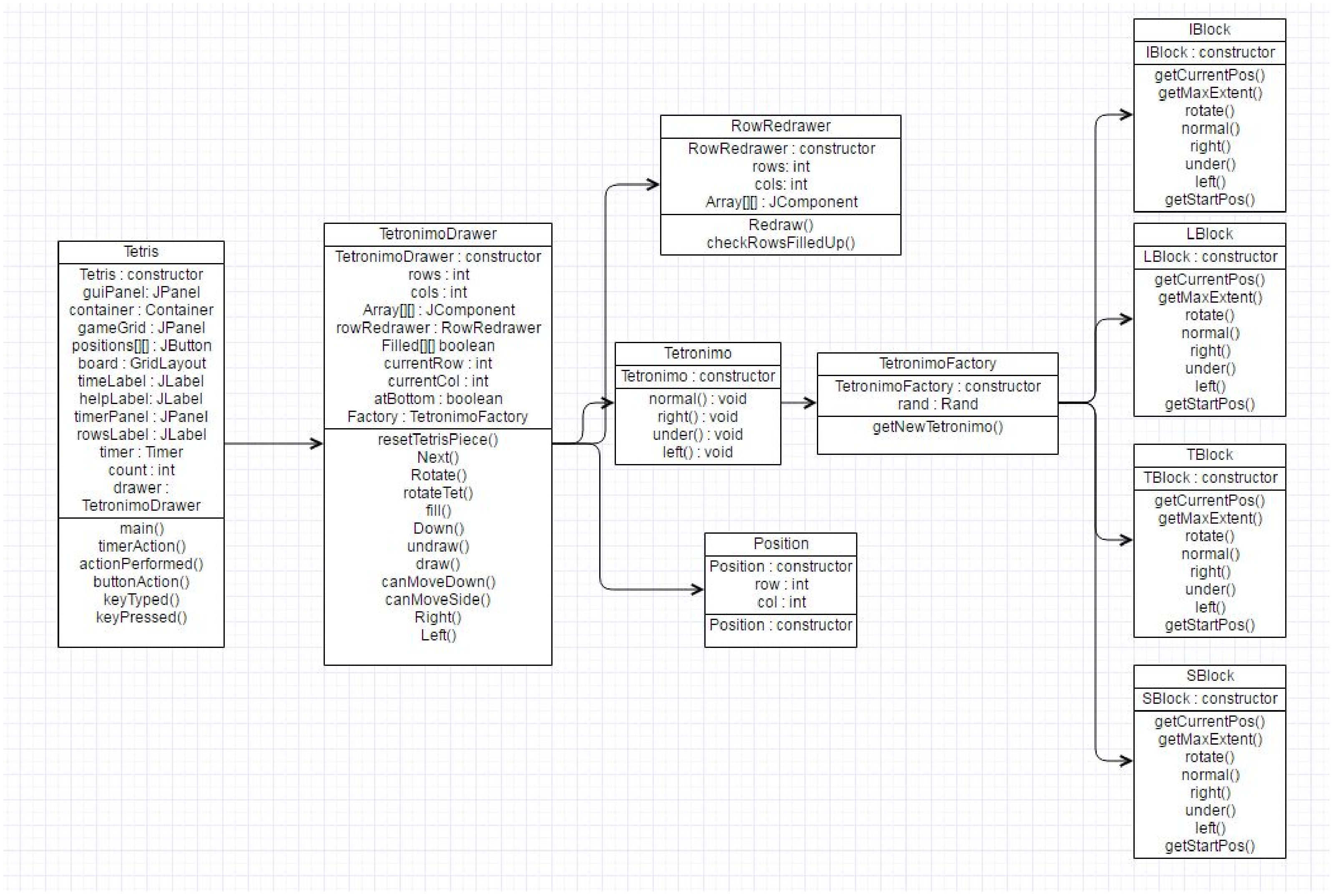
The GUI itself is not so much eye candy, but it should be able to get the job done.
It is our hope that this Java program can serve the needs of those who need entertaining or wish to play to resurge some nostalgia.

Classes (High Level Design)



From a high level design perspective, the user will communicate with a GUI that contains multiple options. These options are interacted with by either keystrokes or mouse clicks. There is also a “help” icon in the GUI and also an “about” button that should both display information regarding the authors and also how to play the game. Once a dialog box is displayed, the user can choose to exit out of the box and continue to play. The same goes with Pausing and starting/restarting the game.

Objects (Low Level Design)



From a lower level design, we can see that there is a tetris class that communicates with a tetronimoDrawer. In this drawer, the important thing to note is the TetronimoFactory. It will contain all the different types of tetronimo pieces that we need.

Design Pattern

- **Factory, Singleton**

Tetris.java

The tetris class is where most of the game functionality will take place. In terms of functionality, a lot of the things, such as timer, score, lines cleared, etc, will be kept track here. More importantly, the board will be set up here, along with each tile being initialized. The action handler and key adapter are both also implemented here. All the key events for escape, arrow keys, enter, and ‘p’ are also implemented here.

- **Int lines cleared // keeps track of lines cleared for difficulty purposes**
- **Timer timer // Note: The blocks are lowered for each second on the timer.**
- **Int score // keeps tracks of score after lines cleared**
- **Int level // increases as user progresses**
- **Random num // for shapes**
- **Boolean isFull // for each line.**

This class will be implemented primarily to keep track of information within the current game. It will also be the class that contains the “game_start” method.

TetronimoDrawer.java

Handles how pieces are introduced one-by-one. There is a resetTetrisPiece method that will take the current variables and update them so that we can use a new piece once the previous piece is laid down. Also handles the case where pieces need to stop moving once they’re locked into place. The rest of the class handles how the pieces are drawn and also checks the boundaries of the pieces as well. For example, in the given position, can it move left/right/under/up without conflicting with the border or other pieces?

This class utilizes the Singleton design pattern. A single instance of this crucial class is instantiated to handle, in an encapsulated manner that hides details from the main user interface, all the drawing of moving tetris pieces.

RowRedrawer.java

For RowRedrawer, this will be the maintenance for the rows during active games. It will handle and check to see if a row is full or not. It will also appropriately color and redraw rows if the cleared row disappears.

Tetronimo.java

The Tetronimo class holds an abstract representation of a tetronimo. It simply just contains the vectors and variables that each of the shapes in the TetronimoFactory would have access to. It contains the normal, right, under, and left vectors for each rotation.

Position.java

The position class is just very helpful for returning and keeping track of the row and col that is passed into the constructor.

TetronimoFactory.java

For the shapes of the tetrominoes, we will need to figure out how to get the coordinates of the piece(s). This will be important if we want to rotate it 90 degrees either clockwise or counterclockwise. As for the shape types, we can make either a class or an enum array, similar to how we implemented cards for Kings Corner. Thankfully, using a design pattern made our lives easier.

TetronimoFactory is a java class that makes use of the Factory design pattern. Basically, the factory design pattern is an elegant way of generating objects. Basically, the creation logic of the object is not exposed to the client. It can be thought of as a form of encapsulation. In this class, a random number is generated and used to randomly pick a block to be used for each tetris piece. For each shape, they all have their own unique properties. The method, getStartPos, allows the program to be aware of what the starting rows and columns are. Then, from here, rotations of the blocks, such as under, left, right, etc are assigning values to create the new position of where the block should be. The I, L, T, and S blocks are the first few blocks that we created, we will need to create the remaining blocks later, but these blocks share some similar characteristics.

IBlock.java

Creates the tetronimo shape “I”

LBlock.java

Creates the tetronimo shape “L”

TBlock.java

Creates the tetronimo shape “T”

SBlock.java

Creates the tetronimo shape “S”

Benefits, Risks, and Assumptions

Benefits

In order for this game to be engaging, the level of difficulty needs to be apparent whenever the user progresses through the game. A benefit of increasing the difficulty will be that the user will have a more challenging experience.

Risks

There might be a more optimal design pattern to use for this project. The risk of going with this design, is that there are so many different design patterns to choose from. So, some might be less encapsulated, or maybe this design over complicates things. Also, another risk is that the GUI might be a bit too simple and make the experience a bit bland. The game itself is not that interesting because it can be stale in a matter of minutes. So, holding the user's attention might be a risk with this design.

Assumptions

We must use design patterns in at least 2 spots. The size of the playing grid is set to 20 rows and 10 columns. One of 7 tetris types is randomly generated for each piece. The difficulty for the game will increase with every 10 lines cleared. It is assumed that the piece rotations should not violate other occupied tiles and/or the boundaries that are set. The controls for this game are primarily the arrow keys. It is assumed that key events are the majority of what the user commands/inputs.