Final Project

Tetris in Processing

Rudd C. Fawcett
COMP-500 Litvin

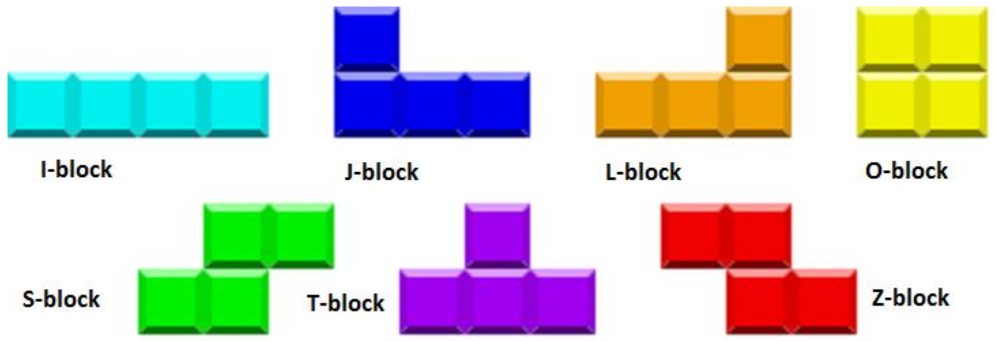
June 2016

Table of Contents

Overview

Introduction

For my COMP-500 project, I made a clone of the classic game Tetris. I built the clone using a combination of pure Java, and Processing. The goal of the game is to get the highest score possible, before you lose. There are no timers, lives, etc. A player gets points by completing a row, thus "clearing it," or by moving a piece downwards, in either a "soft" or "hard" drop.[1] You lose the game by allowing the pieces to stack up all of the way to the top of the Playfield, so no more pieces can drop.



[2]

There are seven different pieces in Tetris, named "Tetrominos."

In Tetris, there are seven different types of pieces, or "Tetrominoes." These pieces can all rotate 360 degrees, and also move right and left. Pieces cannot overlap other pieces. One piece falls at a time, moving down one row per second. All of the pieces rotate differently, and have different pivot points. Because of this, the representations of each piece were hard coded into the game, based on the orientations of each piece.

Challenges

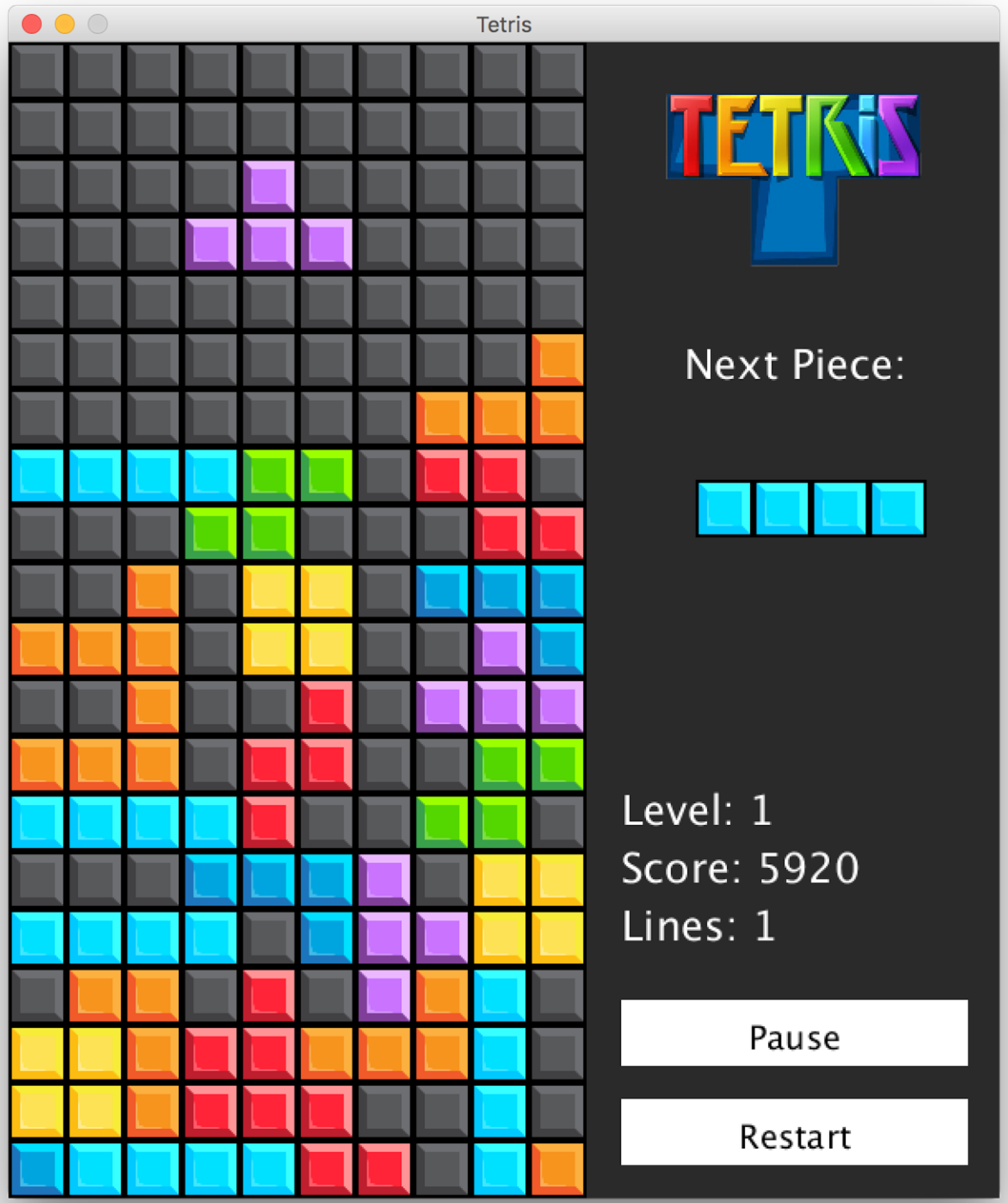
Perhaps the biggest initial challenge was figuring out how to represent the pieces in the block matrix. At first I was thinking of using a matrix of `booleans`, but then realized that I wouldn't be able to distinguish which block was which after going through the array. Thus, I settled on using `ints`, as I could code each `Tetromino` to a different int. Unlike Objective-C, the only other language with which I had used Enums, Java does not map Enum values to integers, but rather keeps them as abstract values. Because of this, I ended up using some `ints` as Constructors, and storing a given numerical value to them (hardcoded).

Another challenge was figuring out how a piece would fit into the matrix. After some trial and error, however, I calculated the potential placement of the new piece, and then looped through each block in the piece to see if it would overlap with any preexisting pieces. I used a similar approach to see if a piece could rotate—rotating it, seeing if it would fit, and then rotating it back.

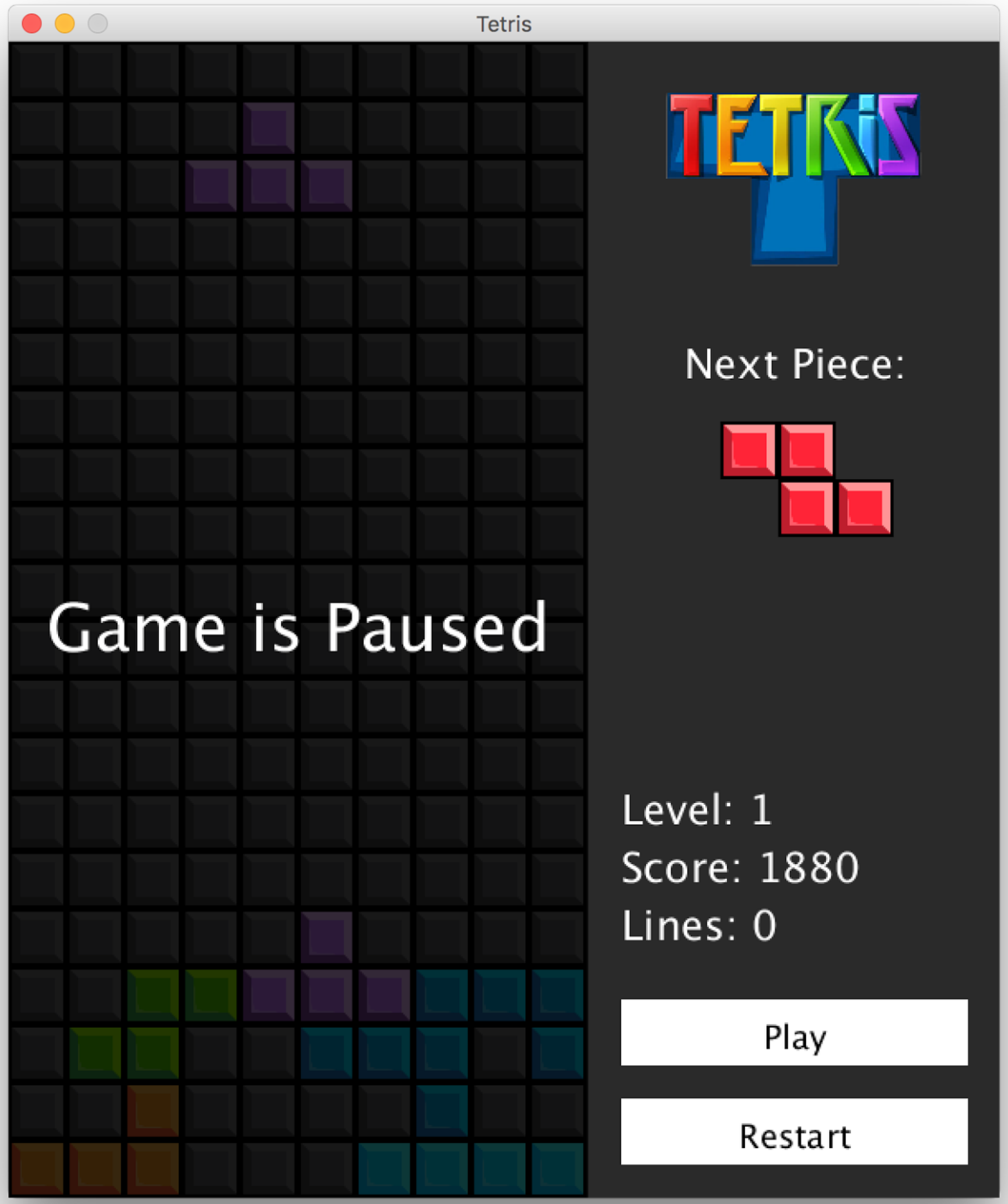
[1] http://tetris.wikia.com/wikiHard_Drop & http://tetris.wikia.com/wiki/Soft_Drop. Last accessed May 30, 2016.
[2] https://i.imgur.com/9Z0oJXe.png Last accessed May 30, 2016.

Game Walkthrough

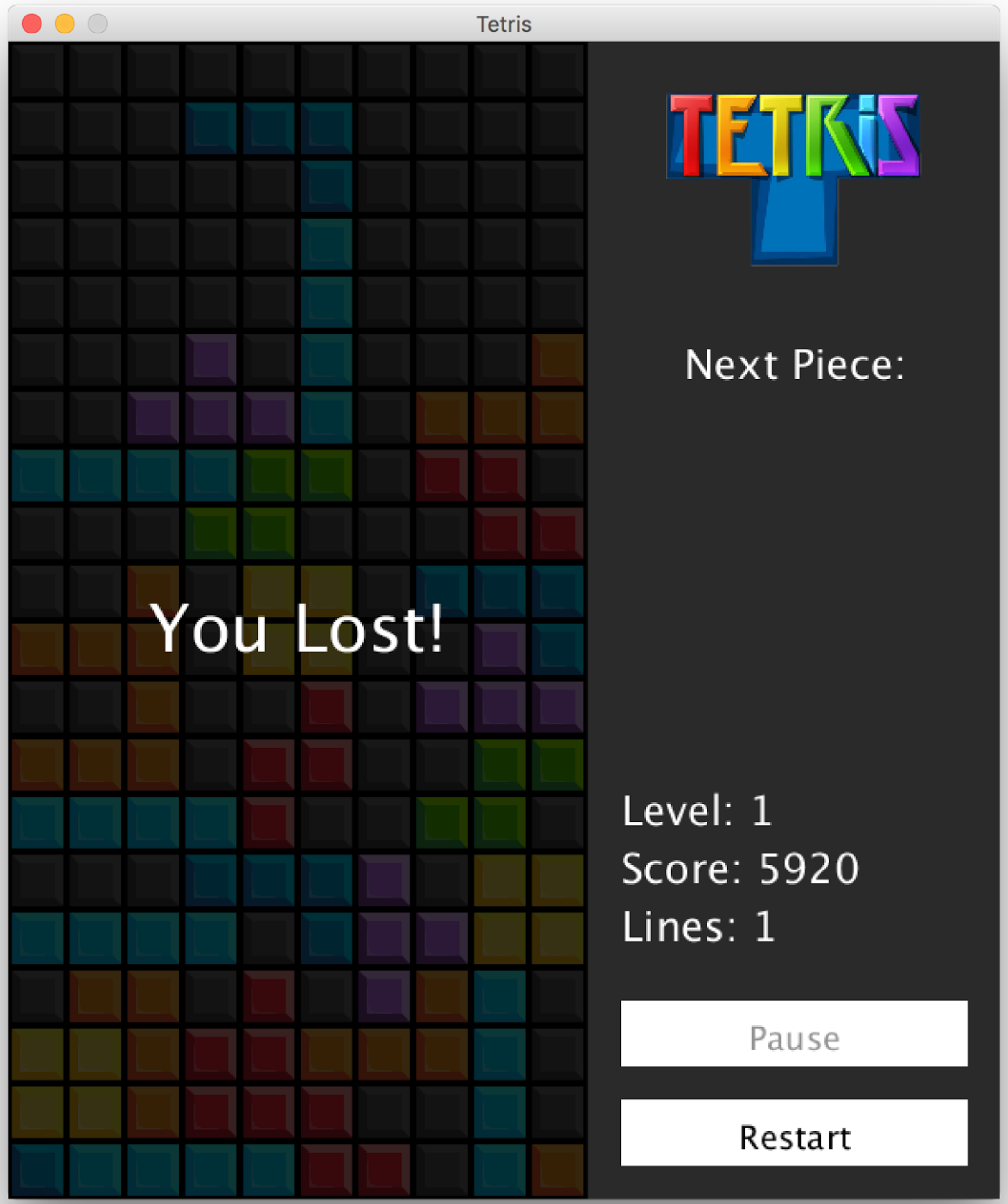
The first screenshot is from a regular played game of Tetris:



The second screenshot is from the game in a paused state:



The third screenshot is from the game when you've lost:



Code Breakdown

While coding Tetris, I struggled with how to best organize the project. I knew that I had to breakdown the various elements of the game into Classes, Enums, etc. but I didn't want (and the assignment description didn't allow for) the entire project to be coded in pure Java. Because of this, I only used `.java` files for Enums, as they are not currently supported in Processing (`.pde`) files as of version 3.0.2. The other classes are all in Processing files, and most of them utilize Processing libraries and types, such as `color` and `PImage`. Processing files also contain standard Java Classes, as you can fully construct and use Classes within a `.pde` file.

The following code breakdown is arranged alphabetically by language, and then filename.

Java

`Direction.java`

`Direction.java` doesn't actually contian any runnable code or methods, but rather contains the `public enum Direction`. Direction is an abstract representation of both direction, and of the four arrow keys, up, right, left down. The concept of direction is therefore represented by `UP, RIGHT, LEFT, DOWN`, respectively in their Enum declaration.

`GameState.java`

`GameState.java`, like `Direction.java` holds an Enum, `public enum GameState`. The possible game states in this version of Tetris are `PLAYING, PAUSED`, and `OVER`. These three game states are used by multiple classes in order to keep track of the game as it is being played. For example, if the game state is `OVER`, there is no need to keep calling the `draw()` method, and if the game is `PAUSED`, the user should not be able to rotate or move any pieces on the `Playfield`.

```
/**
 * The opposite state of the game.
 * Used to show the button text.
 */
public String opposite()
```

`PieceOrientation.java`

`PieceOrientation.java` also contains an Enum, `public enum PieceOrientation`. Much like `Direction, PiecOrientaiton` attempts to abstractly represent what we see as "up," "down," etc. on the screen. The four possible orientations are `UP, RIGHT, DOWN, LEFT` (organized in clockwise order).

`PieceOrientaiton`, unlike the previously described Enums, is a bit "smarter." Each Enum is coded, in constructor-like fashion (`UP(0), RIGHT(1)`, for example), in order to keep track of the orientations. Based on the current orientation (`this`, within the declaration), it can tell the rest of the program what the next orientation is, and what the previous orientation is, through the following constructors:

```
/**
 * Constructs an Enum PieceOrientation, with a given value
 * which is then stored, and used to calculate previous()
 * and next() values.
 */
```
`PieceOrientation(int value)`

```
/**
 * Uses the index of the current orientation to calculate
 * the previous orientation cyclically -- UP's previous
 * rotation is LEFT, even though UP is the first
 * orientation, and LEFT is the last).
 * @return PieceOrientation the previous orientation.
 */
```
`public PieceOrientation previous()`[3]

```
/**
  * Uses the index of the current orientation to calculate
  * the next orientation cyclically -- LEFTS's next
  * rotation is UP, even though UP is the first
  * orientation,and LEFT is the last).
  * @return PieceOrientation the next orientation.
  */
```
`public PieceOrientation next()`[4]

Tetromino.java

In Tetris, as mentioned above, there are seven different pieces in the game: I, J, L, O, S, T, and Z. Though all of these pieces have only four blocks, it is quite difficult to handle rotation of each piece based on one configuration, as different pieces may have different pivot points. In order to avoid verbose and rather unnecessary code, `Tetromino.java` is an Enum which represents each type of piece, and contains a representation of each piece as an 2D array of `ints`, in a `HashMap`, where the `PieceOrientation` is the key.

Like with `PieceOrientation`, each piece takes a value which assigns it a numerical type. This makes it easy to distinguish the relationship between pieces and blocks in the `Playfield`. Using two getters, the Enum provided the proper configuration of blocks, and where the piece should appear on the grid.

```
/**
 * Constructs an Enum Tetromino and assigns the Enum a value.
 */
```
`Tetromino(int value)`

[3] Formula used from http://stackoverflow.com/a/6826865. Last accessed on May 30, 2016.
[4] Ibid.

```java
/**
 * A HashMap of the possible configurations for a
 * Tetromino for each possible PieceOrientation.
 * @return Map<PieceOrientation, int[][]> The configurations.
 */
public Map<PieceOrientation, int[][]> getConfigurations()

/**
 * The starting coordinates for the type of Tetromino,
 * as some pieces are four columns wide, while others are three.
 * @return int[] the start coordinates (x, y) for a Tetromino.
 */
public int[] getStartCoordinates()
```

Processing

`Launcher.pde`

`Launcher.pde` is a very basic file which houses required and vanilla Processing methods, such as `draw()`, `setup()`, etc. In the file I initialize my `Tetris` class, set up the window bounds, and listen for mouse and keyboard clicks.

```
/**
 * Run once at the beginning of a program session's life-span. Sets
 * up the window bounds, and constructs a new Tetris game.
 */
void setup()


/**
 * The draw method is called directly after setup(), and is called
 * continuously until told to stop. It handles the updates of the GUI.
 */
void draw()


/**
 * A Processing listener which is called whenever a key is released.
 * Passes the keyCode to the Tetris instance in order to handle GUI
 * updates, AKA piece movement.
 */
void keyReleased()


/**
 * A Processing listener which is called whenever the mouse is
 * clicked. Passes the x and y coordinates of the mouse click to the
 * Tetris instance in order to detect button clicks.
 */
void mousePressed()
```

Piece.pde

The `Piece` class keeps track of the `PieceOrientation` of a piece, it's position within the `Playfield` grid, as well as the various configurations for a `Tetromino`, and the piece's start position. `Piece` was created as a class in order to neatly group various related types and variables into one group.

```
/**
 * Constructs a new piece, with default orientation UP
 */
public Piece()
```

```
/**
 * @return int The x position of the piece in the Playfield grid.
 */
public int getX()
```

```
/**
 * @return int The y position of the piece in the Playfield grid.
 */
public int getY()
```

```
/**
 * @return int The width/height of a piece (all are square
 * representations).
 */
public int getSize()
```

```
/**
 * @return int The Tetromino value of the piece.
 */
public int getType()
```

```
/**
 * @return PieceOrientation The current orientation of the piece.
 */
public PieceOrientation getOrientation()
```

```
/**
 * Returns the block at a position in the piece's configurations.
 * @param  int x The x (column) of the block in the configuration.
 * @param  int y The y (row) of the block in the configuration.
 * @return int The int representation of a block type at that point in
 * the grid.
 */
public int getBlock(int x, int y)

/**
 * Rotates the piece clockwise, updating the current orientation.
 */
public void rotate()

/**
 * Rotates the piece counterclockwise, updating the current
 * orientation.
 */
public void rotateBack()

/**
 * Moves the piece in a Direction direction, on the Playfield grid.
 * Updates the piece's x and y positions accordingly.
 */
public void move(Direction direction)
```

Playfield.pde

The `Playfield` class determines whether or not a piece can move left, right, or down, and is named from the Tetris terminology.[5] It also figures out whether or not a piece can be rotated. The playfield is configured using a block matrix, which is an array of `int` arrays. Using the `Tetromino` Enum, the value of each item in the matrix is coded, and relates to a block, which is how the correct blocks are drawn.

```
/**
 * Creates a new Tetris Playfield.
 * @param Tetris tetris The current Tetris game.
 */
Playfield(Tetris tetris)
```

```
/**
 * Loads all of the images for the block types into an array.
 */
public void loadImages()
```
[6]

```
/**
 * The block images.
 */
public PImage[] getBlockImages()
```

```
/**
 * Adds a new piece to the block matrix.
 * @param Piece piece The piece to add.
 */
public boolean add(Piece piece)
```

```
/**
 * Draws a piece over the actual grid matrix. Gives the illusion the
 * piece is moving, when in actuality it is drawn over the board.
 * @param  Piece piece The piece to draw.
 */
public void draw(Piece piece)
```

```
/**
 * Draws the entire Playfield, with all of the blocks in the matrix.
 */
public void draw()
```

[5] Terminology from the Tetris Wikia. http://tetris.wikia.com/wiki/Playfield Last accessed May 30, 2016.
[6] The images were modified from resources found at http://design.tutsplus.com/articles/create-a-block-game-interface-in-illustrator--vector-5269. Last accessed May 30, 2016.

```
/**
 * Draws an overlay over the Playfield, and shows a message.
 * @param  String text The string to display.
 */
public void drawOverlay(String text)

/**
 * Removes any empty rows from the matrix.
 */
public int removeRows()

/**
 * Remove a row in the matrix.
 * @param  int row The index of the row to remove.
 */
public void removeRow(int row)

/**
 * If the piece can move to a new position based on the block matrix.
 * @param  Piece     piece The piece that the user wants to move.
 * @param  Direction direction The direction the user wants to move
 * the piece.
 * @return boolean Whether or not the piece can move.
 */
public boolean canMove(Piece piece, Direction direction)

/**
 * Rotates the piece, and then sees if the rotation will fit in
 * the matrix using the canMove() method.
 * @param  Piece piece The piece the user wants to rotate.
 * @return boolean Whether or not a piece can rotate.
 */
public boolean canRotate(Piece piece)
```

Scoreboard.pde

The Scoreboard pane is the right pane next to the Playfield. The pane holds information such as the next piece to be played, start/stop/pause buttons, and also the user's current score, level and number of removedLines. The Scoreboard class uses the current Tetris game in order to access all of the values used in presenting the information. It also draws the Tetris logo at the top of the screen.[7]

```
/**
 * Constructs a new scoreboard.
 * @param Tetris the current tetris game, to get all of the
 * information from.
 */
Scoreboard(Tetris tetris)


/**
 * Draws the Scoreboard. Updates the user's score, the next piece,
 * and also draws all of the buttons in the pane.
 */
public void draw()
/**
* Draws the nextPiece from the current Tetris game, and centers it
* horizontally within the pane.
* @param Piece piece The piece to draw.
*/
public void drawNextPiece(Piece piece)


/**
 * Handles a mouse click, to see if the mouse has clicked over
 * any buttons.
 * @param  int mouseX The x coordinate of the mouse click.
 * @param  int mouseY The y coordinate of the mouse click.
 */
public void handleMouse(int mouseX, int mouseY)
```

[7] Image from http://vignette3.wikia.nocookie.net/maditsmadfunny/images/6/6b/Tetris-logo.png/revision/latest?cb=20130223091331. Last accessed on May 30, 2016.

`Tetris.pde`

The `Tetris` class is the command center for the entire game. It draws all of the necessary components, and has functions to pause, restart, and play the game. It also keeps track of the user's level, score, etc. It also uses an algorithm from the Tetris Wikia page to increment the score based on the user's `level`, and number of `removedLines`: $x(n+1)$, where x is a constant based on the user's level, and n is the number of lines removed when a block has been dropped.[8]

```
/**
 * Constructs a new Tetris game.
 * @param PApplet applet The Processing applet, used to play sounds,
 * etc.
 */
Tetris(PApplet applet)

/**
 * Restarts the Tetris game by resetting all values to default.
 */
public void restart()

/**
 * Ends the Tetris game, by setting the game state.
 */
public void end()

/**
 * Sets the game state to PLAYING.
 */
public void play()

/**
 * Pauses the game by setting the game state to paused.
 */
public void pause()

/**
 * The user's current score.
 */
public int getScore()

/**
 * The user's current level.
 */
public int getLevel()
```

[8] Algorithm from http://tetris.wikia.com/wiki/Scoring. Last accessed May 30, 2016.

```
/**
 * The number of removed lines by the user.
 */
public int getRemovedLines()


/**
 * The game's current state.
 */
public GameState getGameState()


/**
 * The Tetris Playfield.
 */
public Playfield getPlayfield()


/**
 * Handler for when a key is pressed. Uses if and switch statements to handle
 * the course of action.
 * @param  int keyCode The key code of the key pressed (Processing type).
 */
public void handleKey(int keyCode)


/**
 * Calculates the score increment, based on the number of lines cleared.
 * @param  int lines The number of lines cleared.
 * @return int The calculated score based on the lines.
 */
public int calculateScore(int lines)


/**
 * Passes on the mouse click event to the Scoreboard in order to detect if a
 * button was clicked.
 * @param  int mouseX the x coordinate of the mouse click.
 * @param  int mouseY the y coordinate of the mouse click.
 */
public void handleMouse(int mouseX, int mouseY)


/**
 * Updates the Playfield and Scoreboard. Called from the Launcher.
 */
public void update()
```

File Breakdown

Besides the physical code for the project the following files were used to finish the final product.

`blocks/`

`tetris-block-x.png`

Each piece is composed of an image which is then drawn in a location based on the `x` and `y` positions of the piece. There is a block image of each `Tetromino` type (1-7) and an empty block (0).[9]

`graphics/`

`tetris-logo.png`

This image is a Tetris logo that is used at the top of the `Scoreboard` pane, in order to provide some more graphic variety.[10]

`sounds/`

`blip.mp3`

The blip sound effect is played whenever a piece lands—hits the bottom of the `Playfield`, or hits the top of another piece.[11]

`swoosh.mp3`

The swoosh sound effect is played whenever a row is removed from the `Playfield`.[12]

[9] Blocks designed from http://design.tutsplus.com/articles/create-a-block-game-interface-in-illustrator--vector-526931 Last accessed May 30, 2016.
[10] Image from http://vignette3.wikia.nocookie.net/maditsmadfunny/images/6/6b/Tetris-logo.png/revision/latest?cb=20130223091331 Last accessed May 30, 2016.
[11] Sound from http://www.flashkit.com/soundfx/Interfaces/Blips/. Last accessed May 30, 2016.
[12] Sound from http://soundbible.com/706-Swoosh-3.html. Last accessed May 30, 2016.

Citations

All of the following links were last accessed on May 30, 2016.

Scoring

The following resources were used in order to calculate the scoring throughout the game.

- http://tetris.wikia.com/wiki/Scoring
- http://tetris.wikia.com/wiki/Hard_Drop
- http://tetris.wikia.com/wiki/Soft_Drop

Gameplay

The following resources were used in order to learn more about Tetris, and the gameplay throughout making the game.

- http://tetris.wikia.com/wiki/Playfield
- http://tetris.com/play-tetris/
- https://en.wikipedia.org/wiki/Tetris
- http://tetris.wikia.com/wiki/Gameplay_overview
- http://tetris.wikia.com/wiki/Category:Rotation_Systems

Processing/Java

The following resources were used when looking up various Java/Processing methods and functions.

- https://docs.oracle.com/javase/tutorial/essential/exceptions/catch.html
- https://processing.org/reference/noLoop_.html
- https://processing.org/reference/loop_.html
- https://processing.org/reference/PImage.html
- https://processing.org/reference/textLeading_.html
- https://processing.org/reference/noStroke_.html
- https://forum.processing.org/one/topic/how-to-display-text-in-the-title-of-the-display-window.html
- https://stackoverflow.com/questions/6826826/how-could-i-have-the-index-of-an-array-roll-over-when-incrementing
- https://stackoverflow.com/questions/12417937/create-a-simple-countdown-in-processing
- https://stackoverflow.com/questions/14149733/clone-method-for-java-arrays

Design

The block images were taken from the following resource in the making of the game.

- http://design.tutsplus.com/articles/create-a-block-game-interface-in-illustrator--vector-5269

Sounds

The following sounds were used in the making of the Tetris game.

- http://soundbible.com/706-Swoosh-3.html
- http://www.flashkit.com/soundfx/Interfaces/Blips/