CS4398 San Marcos Group 2

Tetris Presentation

By Zach Goldberg, Ryan Dalan, Eddie Dounn, Guillermo Gomez

Tetris, and what we did.

- A simple puzzle game where the user must move a series of "falling" pieces, or Tetrimos.
- The game is endless, meaning that there is no win state, the player keeps playing until they lose.
- If the player fails to clear lines and a piece stops falling while touching the top of the screen the game ends.
- If the user fills a line the line is cleared and the player gains points.
- We recreated the game from the ground up using only base Java (JFrame/Swing), and our custom assets (sprites/background/music). We used no game engines to aid us.

Model View Controller Application

We attempted to design the game via the MVC paradigm that we learned in class. It proved to be simple in theory; however, some application was difficult (mostly due to how Java works regarding GUI's and accessing files).

The idea was for the Model to be the back end logic of the game that detected collision and generated additional pieces via randomizer and also looped to continuously check the gamestate. The View was what the player saw/interacted with aka the User Interface/UI of the game. The Controller was the keyboard actions that the user could take (such as rotating blocks).

Our class creation reflected this paradigm. For example, the Package, View, had classes that dealt with handling the JFrame.

Use Cases

Name: Start
Trigger : User interaction to begin game
Precondition : User starts application and inside Title screen.
While in the application the user selects <START >.
Game play screen displays.

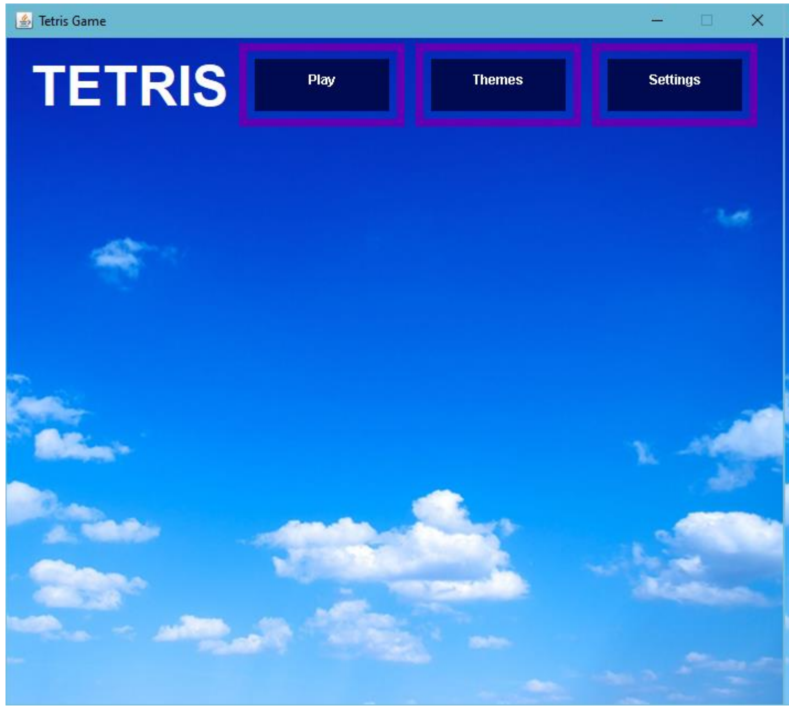
Name: Pause
Trigger : User interaction to pause game
Precondition : User in game
While in the application the user selects <PAUSE >.
Game play stops and displays indication that game is paused.
Continue resumes game play

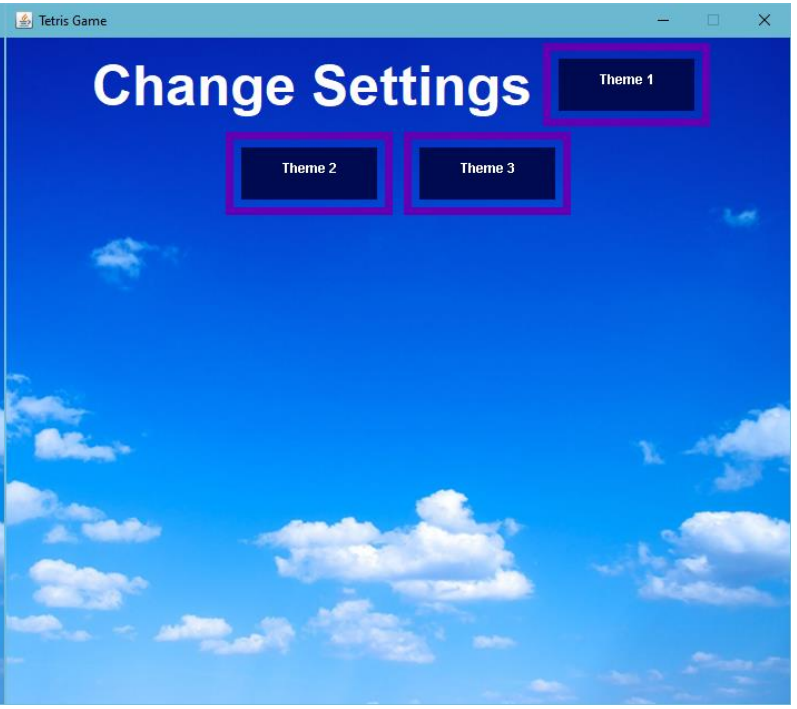
Name: Exit
Trigger : User interaction to quit current game
Precondition : User at title screen or pause screen
While in the title screen or pause screen user selects < EXIT >.
Game is closed.

- Name: Theme Customization
Trigger: User interaction with the game and the blocks.
Precondition: User starts the game.
While in the application the user selects <Theme>.
While in the <Theme> menu, the user tweaks the color of the blocks to their liking.

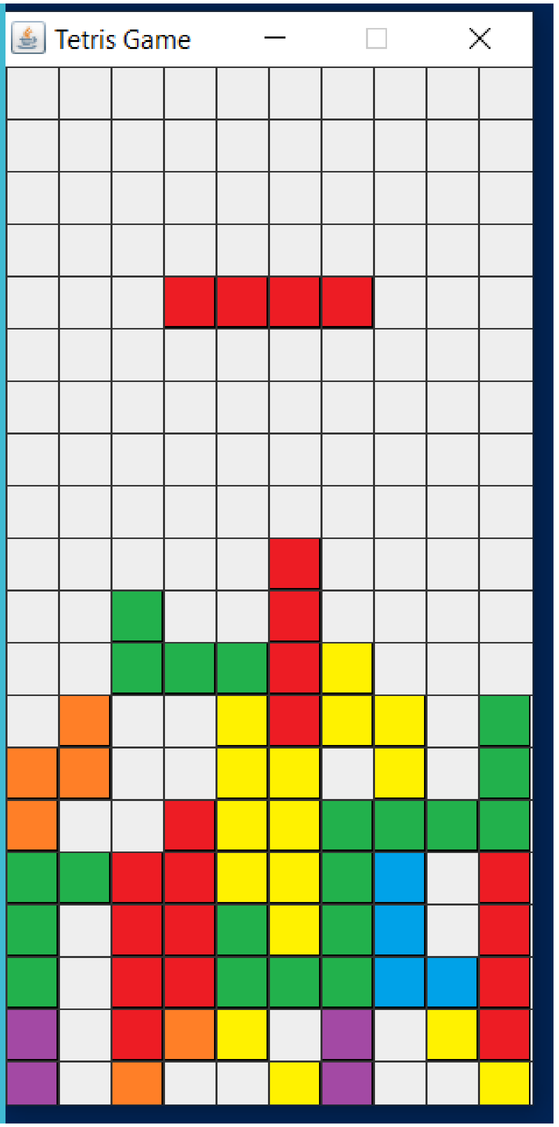
- Name: Option Tweaking
Trigger: User interaction with the game and its options.
Precondition: User starts the game.
While in the application the user selects <OPTIONS>.
While in the <OPTIONS> menu, the user tweaks the options of the game to their liking. This includes keybindings, resolution, and volume.

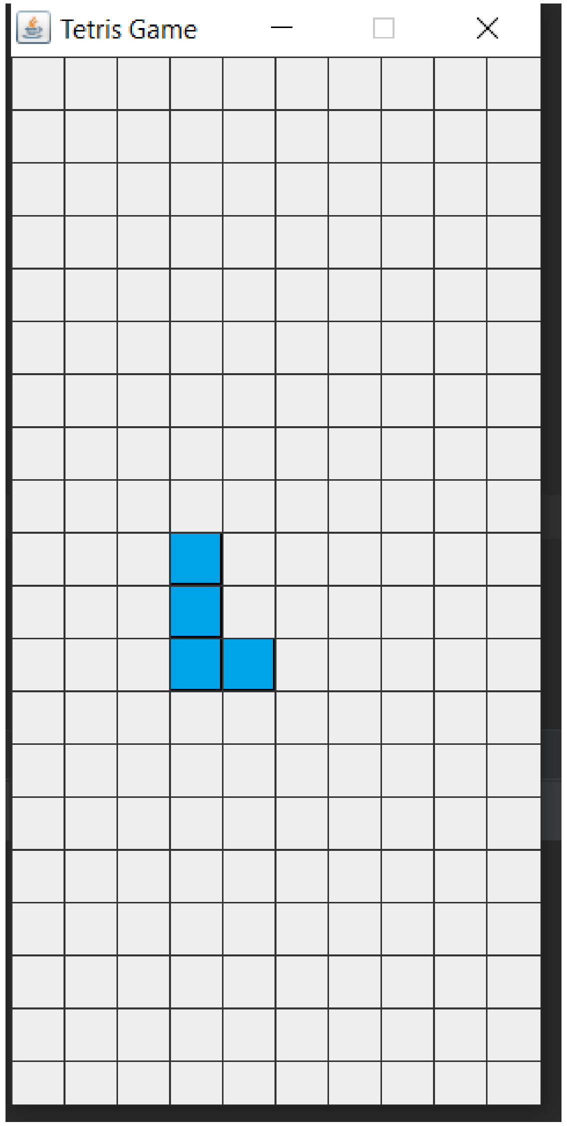
User Interface

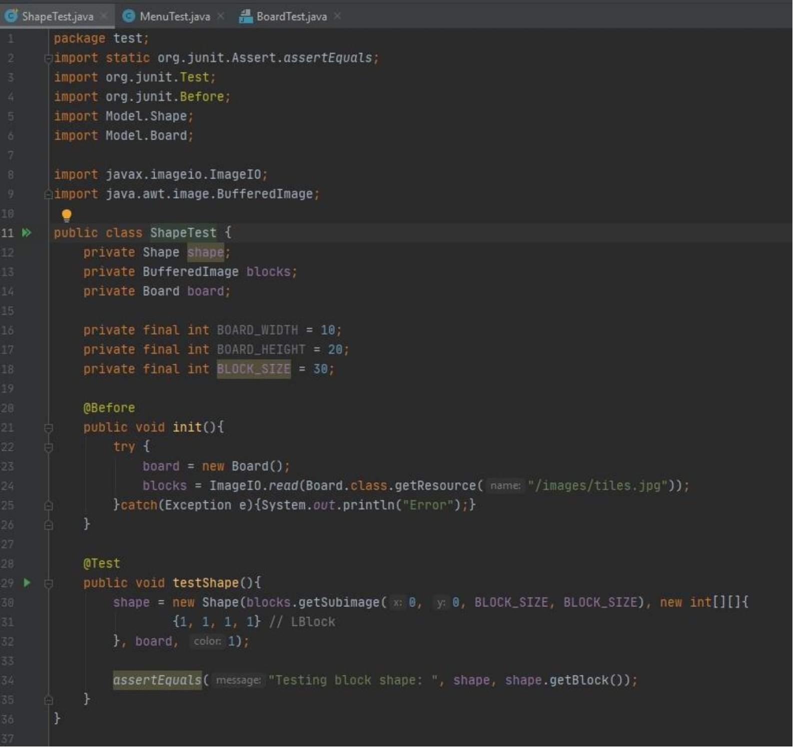




Gameplay

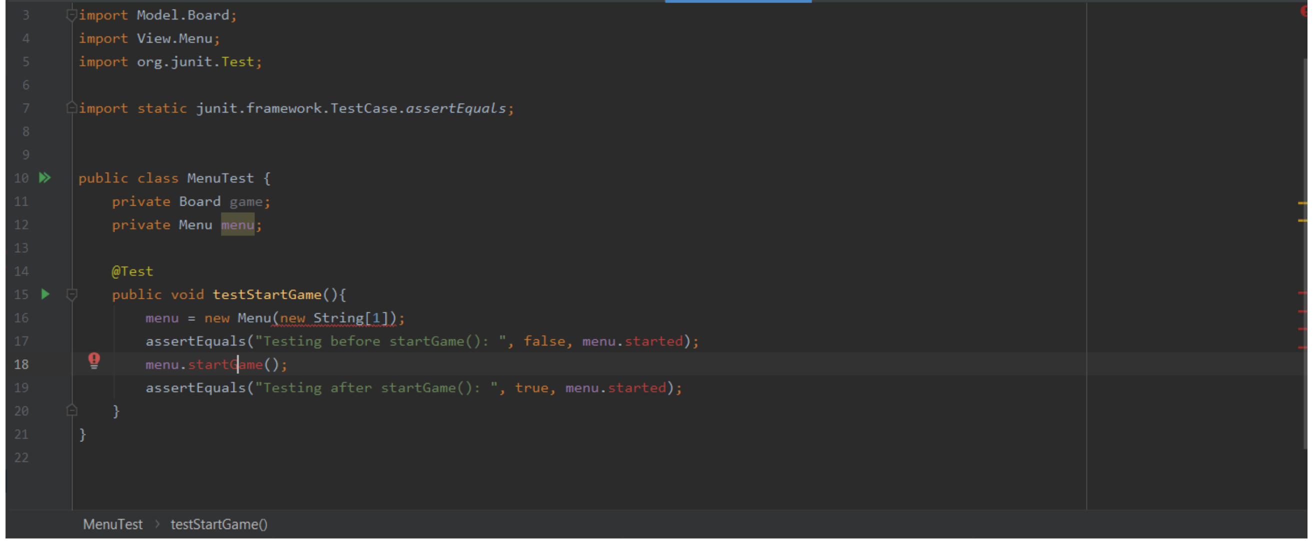






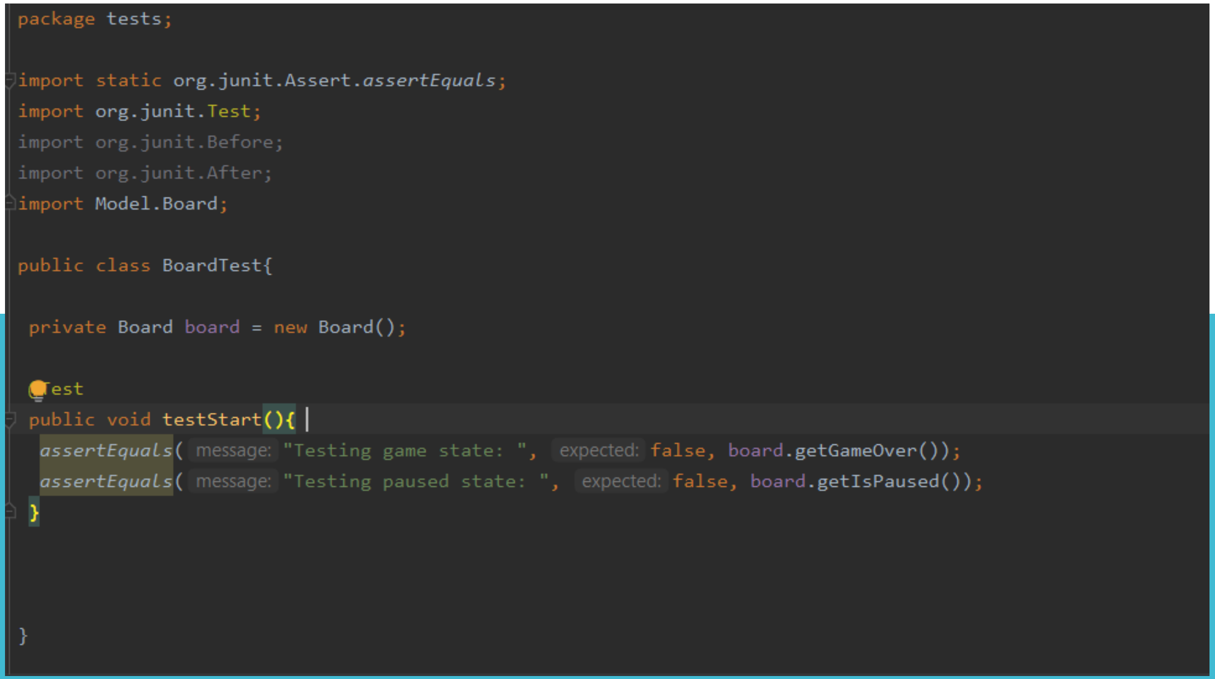
ShapeTest.java | MenuTest.java | BoardTest.java

```java
package test;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
import org.junit.Before;
import Model.Shape;
import Model.Board;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;

public class ShapeTest {
    private Shape shape;
    private BufferedImage blocks;
    private Board board;

    private final int BOARD_WIDTH = 10;
    private final int BOARD_HEIGHT = 20;
    private final int BLOCK_SIZE = 30;

    @Before
    public void init(){
        try {
            board = new Board();
            blocks = ImageIO.read(Board.class.getResource( name: "/images/tiles.jpg"));
        }catch(Exception e){System.out.println("Error");}
    }

    @Test
    public void testShape(){
        shape = new Shape(blocks.getSubimage( x: 0,  y: 0, BLOCK_SIZE, BLOCK_SIZE), new int[][]{
                {1, 1, 1, 1} // LBlock
        }, board,  color: 1);

        assertEquals( message: "Testing block shape: ", shape, shape.getBlock());
    }
}
```



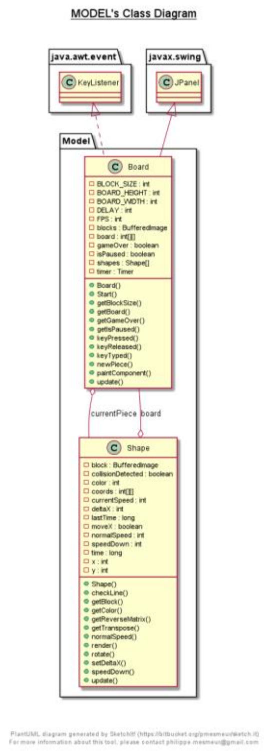
```java
import Model.Board;
import View.Menu;
import org.junit.Test;

import static junit.framework.TestCase.assertEquals;


public class MenuTest {
    private Board game;
    private Menu menu;

    @Test
    public void testStartGame(){
        menu = new Menu(new String[1]);
        assertEquals("Testing before startGame(): ", false, menu.started);
        menu.startGame();
        assertEquals("Testing after startGame(): ", true, menu.started);
    }
}
```

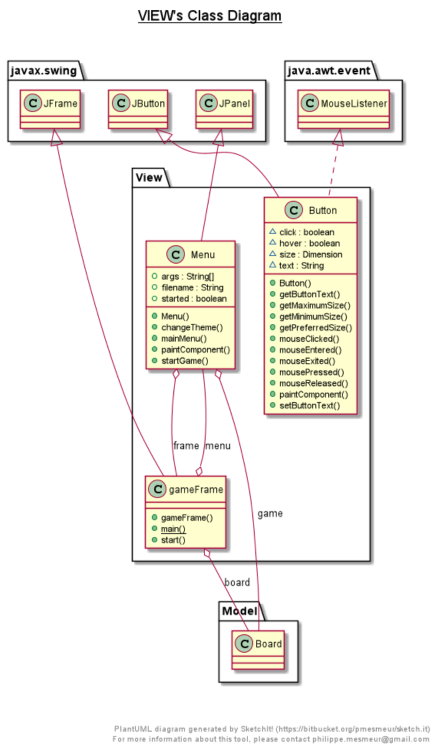
MenuTest › testStartGame()

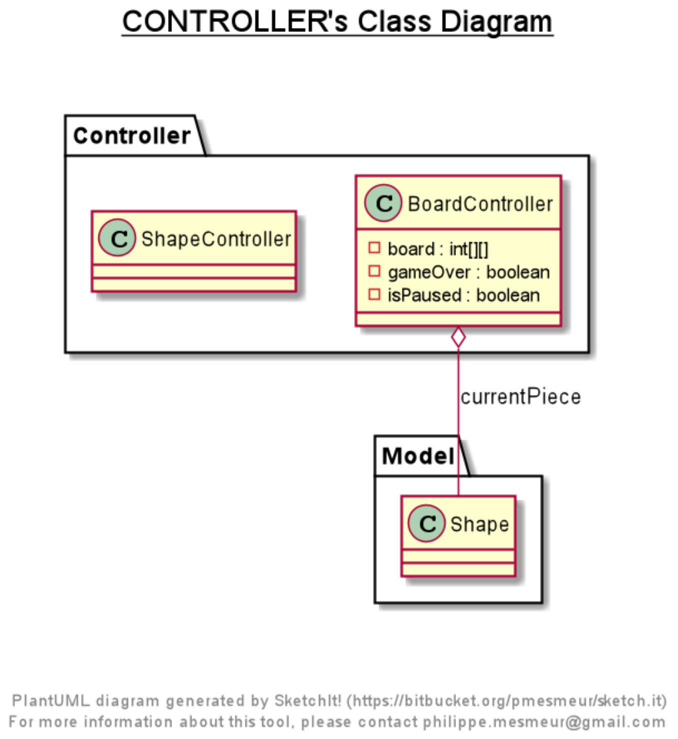


```java
package tests;

import static org.junit.Assert.assertEquals;
import org.junit.Test;
import org.junit.Before;
import org.junit.After;
import Model.Board;

public class BoardTest{

 private Board board = new Board();

 @Test
 public void testStart(){
  assertEquals( message: "Testing game state: ",  expected: false, board.getGameOver());
  assertEquals( message: "Testing paused state: ",  expected: false, board.getIsPaused());
 }



}
```

Unit Tests






CONTROLLER's Class Diagram
Controller
ShapeController
BoardController
board : int[][]
gameOver : boolean
isPaused : boolean
currentPiece
Model
Shape
PlantUML diagram generated by SketchIt! (https://bitbucket.org/pmesmeur/sketch.it)
For more information about this tool, please contact philippe.mesmeur@gmail.com

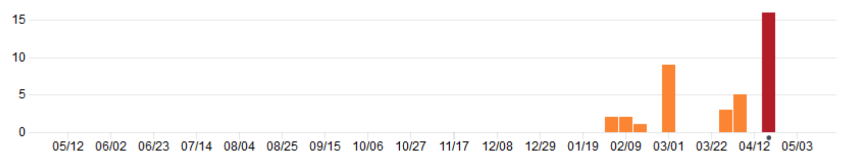
Class Diagrams

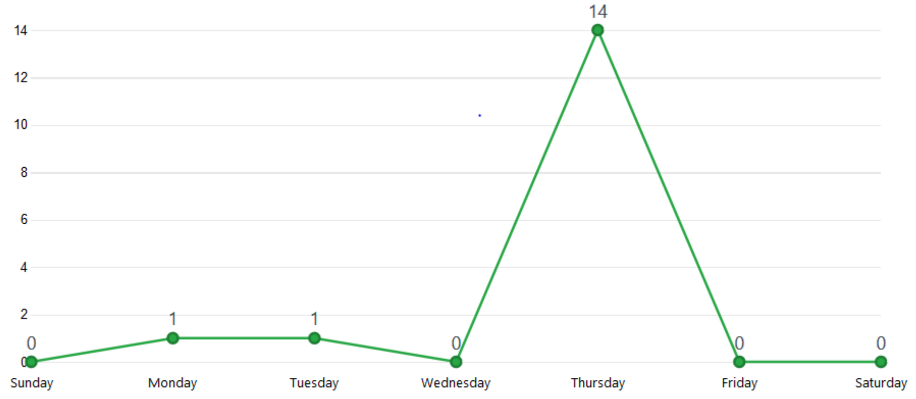
Version Control

Git as version control

We used git through github.com for our version control. Each member kept their own branch and we merged into the master when progress was made.

https://github.com/edounn/cs4398-sm2





Impediments and Future Features

- Impediments
  - A major impediment, though non-technical, was the current state of affairs from COVID. It made real-life communication impossible, thus leading to problems in workflow.
  - Using Java and creating a game-engine from the ground up made for a shunted experience in both polish and introduced problems via bugs. In retrospect, it would have been easier to use established engines i.e. Unity.
- Future Features
  - Internet connected score board
  - Multi-threading to improve performance
  - Additional themes and sounds
  - Increased difficulty as the game progresses