EJ Lilagan, Neo Argatides, Robert Mahin
Dr. Zheng
4 May 2023
SE461 - Software Testing and Quality

SE461 - Tetris Project

Introduction:
Our group, made up of EJ, Neo, and Rob, are tasked with creating JUnit test cases for a coverage tool called JaCoCo to confirm the branches that have been covered in both the Tetris game and JUnit tests given a Tetris game that is based on Java. Provided below was our application process to complete the Tetris project.

Objective:
The goal for this group project is to utilize our knowledge from all of the material covered in this course and apply it to the Tetris application to access our abilities that have been taught. Additionally, reviewing our prior assignments and applying our expertise onto the project to utilize those concepts on Eclipse, Control Flow Graphs, and test requirements.

With instructions provided, it is mandatory that we must follow the professor's needs and suggestions to successfully complete the group project. Below in the requirements page will go into depth of what is expected to do the group project.

Requirements:
Foremost, we were tasked to complete this project with its entirety by May 7, 2023. The following would consist of having three members in a group to work on the Tetris application. The requirements that were important for us to complete consist of the following:

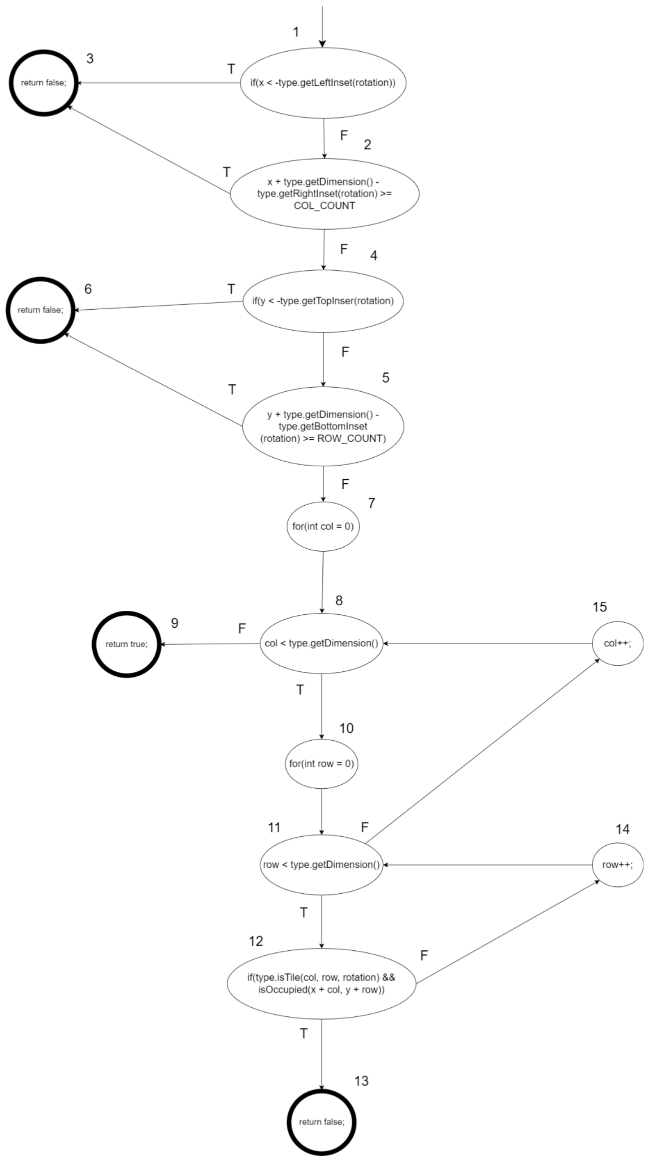
Use at least one test coverage criteria, which can be defined as a rule or collection of rules that impose test requirements on a test set (TR.) This will be needed for developing JUnit test cases that will be written in the Eclipse environment and can be referenced as to how we set up our assignment 3.

While that is in consideration, the team must consider the following aspects for our report. Firstly, it is involving our test design that must satisfy the coverage criterion that we used. The minimum number of examples that we can include are at least three and should include graphs, TR's, test cases (provide an explanation with input and expected output), and test paths. Secondly, it is recording test results and analysis that should specify the problems in the code. An example is finding any bugs that were caught in the code, the coverage score that is recorded from JaCoCo, and providing screenshots of the following that the group has included. Lastly, the group must describe their experience of the project by accumulating and orienting all thoughts that the group has by including their process with passed/failed attempts.

A requirement that was not shown on the project was taking into consideration the user not controlling the game, which indicates that a player should not control the game. Having the user to control the game would easily pass all the following test cases that are needed within the JUnit and JaCoCo, which would give us a challenge for us to research more about other ways to run the code. Not mentioned by the professor, it was mandatory to allow the computer to run our code to test it themselves versus the user typing the keys and checking valid and invalid keys and regions of the board.

<u>Process:</u>
Based on our expertise, we planned on drawing control flow diagrams for each file that was contained in the tetris package. For this portion, EJ did BoardPanel,java, Neo did Clock.java, and Rob did SidePanel.java, TileType.java and Tetris.java. With our combined knowledge of drawing our graphs, TR's, test cases, and test paths, this was the results we came up with for isValidAndEmpty, updateGame, and update.

– public boolean isValidAndEmpty(TileType type, int x, int y, int rotation) –


1
if(x < -type.getLeftInset(rotation))
T
3
return false;
F
2
x + type.getDimension() - type.getRightInset(rotation) >= COL_COUNT
T
F
4
if(y < -type.getTopInser(rotation)
T
6
return false;
F
5
y + type.getDimension() - type.getBottomInset (rotation) >= ROW_COUNT)
T
F
7
for(int col = 0)
8
col < type.getDimension()
F
9
return true;
15
col++;
T
10
for(int row = 0)
11
row < type.getDimension()
F
14
row++;
T
12
if(type.isTile(col, row, rotation) && isOccupied(x + col, y + row))
F
T
13
return false;

TR= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
TR_PPC= {[1, 3], [1, 2, 4, 6],
[1, 2, 4, 5, 7, 8, 10, 11, 12, 14, 11, 12, 14, 11, 15, 8, 10, 11, 12, 14, 11, 15, 8, 9],
[1, 2, 4, 5, 7, 8, 10, 11, 12, 13], [1, 2, 4, 5, 7, 8, 10, 11, 12, 14, 11, 12, 14, 11, 12, 13],
[1, 2, 4, 5, 7, 8, 10, 11, 12, 14, 11, 12, 14, 11, 15, 8, 10, 11, 12, 13]}

1. [1,3]
    a. Test Case: Tile Type = TypeI, x = 10, y = 10, r = 0
    b. Expected Output: false
2. [1, 2, 4, 6]
    a. Test Case: Tile Type = TypeI, x = 2, y = 21, r = 1
    b. Expected Output: false
3. [1, 2, 4, 5, 7, 8, 10, 11, 12, 14, 11, 12, 14, 11, 15, 8, 10, 11, 12, 14, 11, 15, 8, 9]
    a. Test Case: Tile Type = TypeO, x = 5, y = 5, r = 0
    b. Expected Output: true
4. [1, 2, 4, 5, 7, 8, 10, 11, 12, 13]
    a. Test Case: Tile Type = TypeO, x = 5, y = 5, r = 0
    b. Expected Output: false
5. [1, 2, 4, 5, 7, 8, 10, 11, 12, 14, 11, 12, 14, 11, 12, 13]
    a. Test Case: Tile Type = TypeS, x = 3, y = 5, r = 0
    b. Expected Output: false
6. [1, 2, 4, 5, 7, 8, 10, 11, 12, 14, 11, 12, 14, 11, 15, 8, 10, 11, 12, 13]
    a. Test Case: Tile Type = TypeL, x = 5, y = 4, r = 2

***Legend**
Tile Type = piece placed on board
x = row
y = column
r = rotation

As indicated above, here are the results for isValidAndEmpty. This function will serve itself to grab a tile type to be passed onto the board panel to determine if the piece is eligible for any coordinates within the board. Given that it is a boolean value, it will return the following pieces if it is a valid column/row or conflicts with an existing tile on the board.

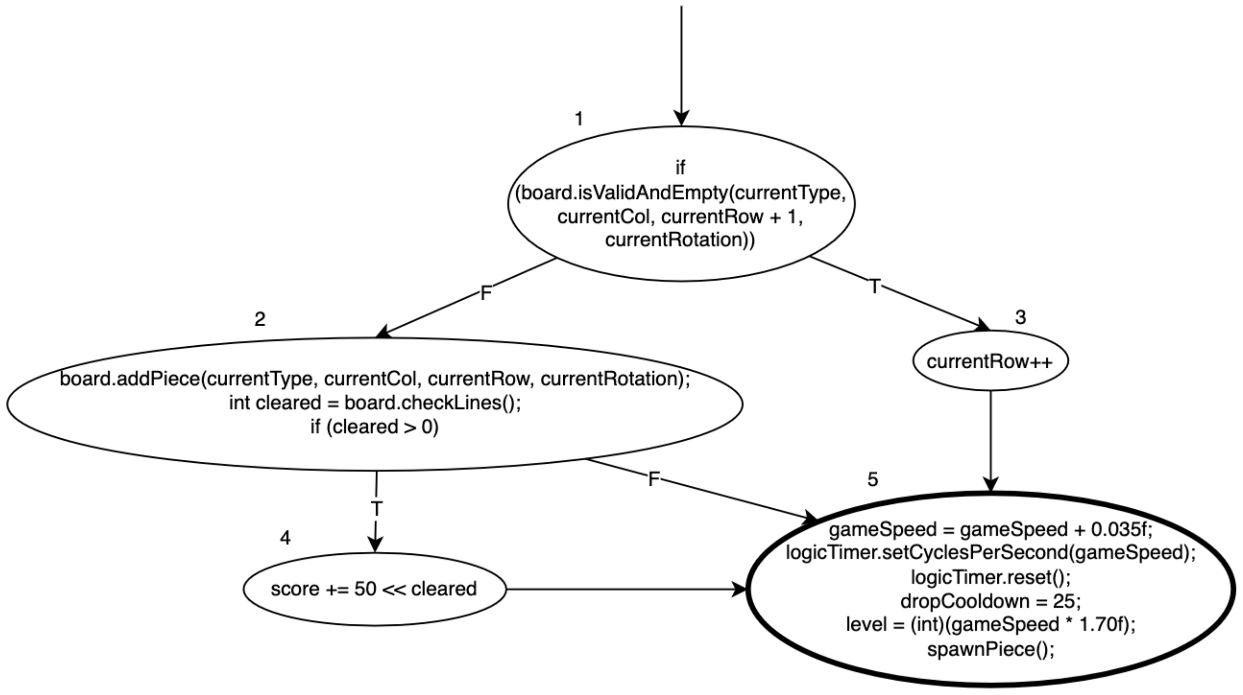
We ended up using PPC (also known as Prime Path Coverage) due to its efficiency and strict nature to find feasible paths from the function. In addition, it would lead to paths that can either be completed or incomplete meaning it will be difficult to not have all PPC be reached, which is why our group was not able to involve all PPC within our JUnit.

Feasible:
[1, 3]
[1, 2, 3]
[1, 2, 4, 6]
[1, 2, 4, 5, 6]

[1, 2, 4, 5, 7, 8, 9]
[1, 2, 4, 5, 7, 8, 10, 11, 15, 8, 9]
[1, 2, 4, 5, 7, 8, 10, 11, 12, 13]
[1, 2, 4, 5, 7, 8, 10, 11, 12, 14, 11, 15, 8, 9]

Infeasible:
[7, 8, 9] ⇒ this loop iterates through every column, however it is unlikely for type.getDimensions() to have a value greater than zero.
[10, 12, 14] ⇒ this loop iterates through every row, however it will face the same result as the above.

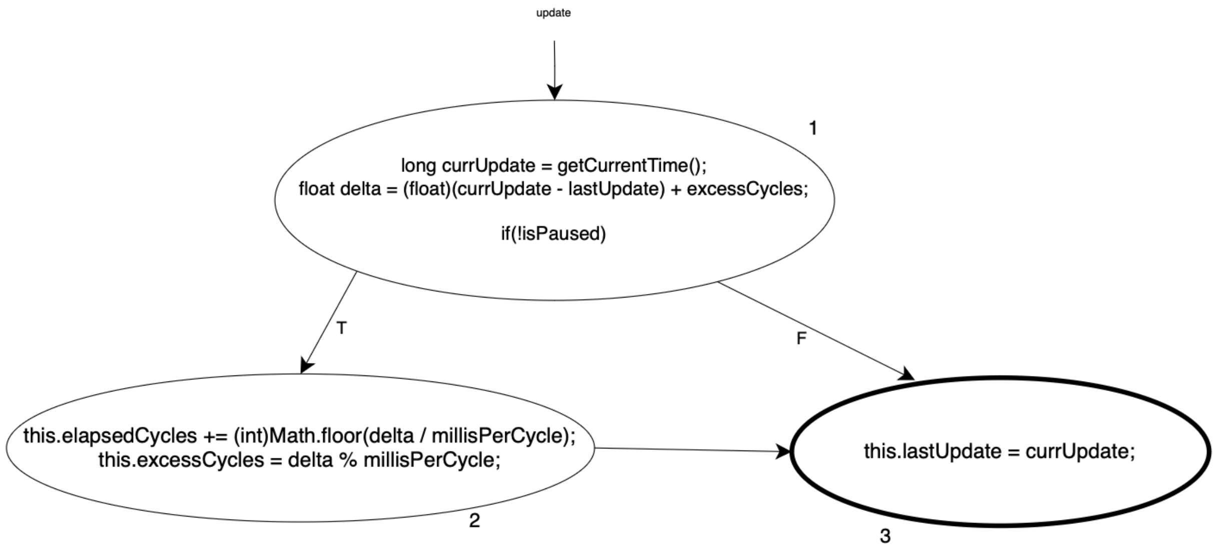
– updateGame() –


1
if (board.isValidAndEmpty(currentType, currentCol, currentRow + 1, currentRotation))
F
T
2
board.addPiece(currentType, currentCol, currentRow, currentRotation);
int cleared = board.checkLines();
if (cleared > 0)
3
currentRow++
F
T
4
score += 50 << cleared
5
gameSpeed = gameSpeed + 0.035f;
logicTimer.setCyclesPerSecond(gameSpeed);
logicTimer.reset();
dropCooldown = 25;
level = (int)(gameSpeed * 1.70f);
spawnPiece();

TRs: {1, 2, 3, 4, 5}
TR_{PPC} = {[1,2,5], [1,2,4,5], [1,3,5]}
Test cases:

1. [1,2,5]
    a. Input: currentType = TypeI, currentCol = 10, currentRow = 9, currentRotation = 0, cleared = 0
2. [1,2,4,5]
    a. Input: currentType = TypeI, currentCol = 10, currentRow = 9, currentRotation = 0, cleared = 1
3. [1,3,5]
    a. Input: currentType = TypeO, currentCol = 5, currentRow = 4, currentRotation = 0

– update() –


update
1
long currUpdate = getCurrentTime();
float delta = (float)(currUpdate - lastUpdate) + excessCycles;
if(!isPaused)
T
F
this.elapsedCycles += (int)Math.floor(delta / millisPerCycle);
this.excessCycles = delta % millisPerCycle;
2
this.lastUpdate = currUpdate;
3

TRs: {1, 2, 3}
TR_{PPC} = {[1, 3], [1, 2, 3]}
Test cases:

1. [1, 3]
   a. Input: isPaused = true
2. [1, 2, 3]
   a. Input: isPaused = false

– checkLine() –

Function is found on BoardPanel/TestBoardPanel classes. Our group decided to use logic coverage to change up our strategies and try something different with PPC. The logic coverage we decided on using is ACoC also known as All Combinations Coverage, which is effective to identify the behavior and outcome of the application. In addition, it was able to determine test cases that are able to run as provided below, and indicates the characteristics that are into consideration for the ACoC for indicating

Characteristic 1: Return ⇒ 1 = full row, 0 = one of the columns are empty
Characteristic 2: Line Place ⇒ F = first row, M = middle row, L = last row
Characteristic 3: Board State ⇒ E = empty board, A = board with tiles

TR_ACoC = {(1, F, E), (1, F, A), (1, M, E), (1, M, A), (1, L, E), (1, L, A), (0, F, E)}

Test Set: (1, F, E) ⇒ Full row, first row, empty board
Recorded on row 0 of the board.
Internal State:

x	x	x	x	x	x	x	x	x	x
..	..	..	..	..	..	..	..	..	..

Expected State would not change since it is unlikely to fill the top row when the rest of the board is empty.

Test Set: (1, F, A) ⇒ Full row, first row, board containing tiles
Recorded on row 0 with full row and random tiles sorted all over the board
Internal State:

x	x	x	x	x	x	x	x	x	x
	x	x		x		x	x		x
..	..	..	..	..	..	..	..	..	..

Expected State would not change since it is unlikely for the top row to fill up with space left underneath the board.

Test Set: (1, M, E) ⇒ Full row, middle row, empty board
Recording the middle row to be full, but empty
***(Not according to scale but used as an example…) → dimensions 10x11**
Internal State

x	x	x	x	x	x	x	x	x	x

🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨

Expected State would change because anywhere besides the top row is clearable. If it were to be anywhere besides the top row, it is valid to clear that row within that board.

Test Set: (1, M, A) ⇒ Full row, middle row, board containing with tiles
Recording the middle row to be full with random tiles placed all over the board
***(Not according to scale but used as an example…) → dimensions 10x11**
Internal State:

x		x			x			x	
	x		x		x		x	x	x
	x			x		x	x	x	
x			x	x		x			x
	x	x	x				x		x
x	x	x	x	x	x	x	x	x	x

	x		x		x			x	
x	x		x	x	x		x		x
x		x		x		x			x
x			x				x	x	

						x	x	x	x
x		x			x			x	
	x		x	x	x		x	x	x
	x		x	x		x	x	x	
x				x		x			x
	x	x	x				x		x
	x		x		x			x	
x	x		x	x	x		x		x
x		x		x		x			x
x			x				x	x	

Expected State would change since the row is full, which would drag the rest of the tiles from the top a layer down and allow the board to be reused.

Test Set: (1, L, E) ⇒ Full row, middle row, empty board
Recording the middle row to be full with random tiles placed all over the board
Internal State:

..	..	..	..	..	..	..	..	..	..
x	x	x	x	x	x	x	x	x	x

..	..	..	..	..	..	..	..	..	..

Expected state would change since the bottom row would clear the whole row and it would allow the board to change the whole scene into an empty board.
Test Set: (1, L, A) ⇒ Full row, last row, board with tiles
Recording the last row to be full with random tiles around the board
Internal State:

..	..	..	..	..	..	..	..	..	..
	x	x		x			x		x
x	x	x	x	x	x	x	x	x	x

..	..	..	..	..	..	..	..	..	..
x	x					x	x		x
	x	x		x			x		x

Expected state would be changed since the last row would clear the whole board and the following tiles above the cleared row will fill in the spots that are vacant, however, it would not clear the following row as it has not been full.

Test Set = (0, F, E) ⇒ one of the columns are empty, first row, empty board
Recording the first row to have some tiles covered, but not the whole row
Internal State:

x		x	x		x		x		x
..	..	..	..	..	..	..	..	..	..

Expected state would not be changed since the whole row is not clear.

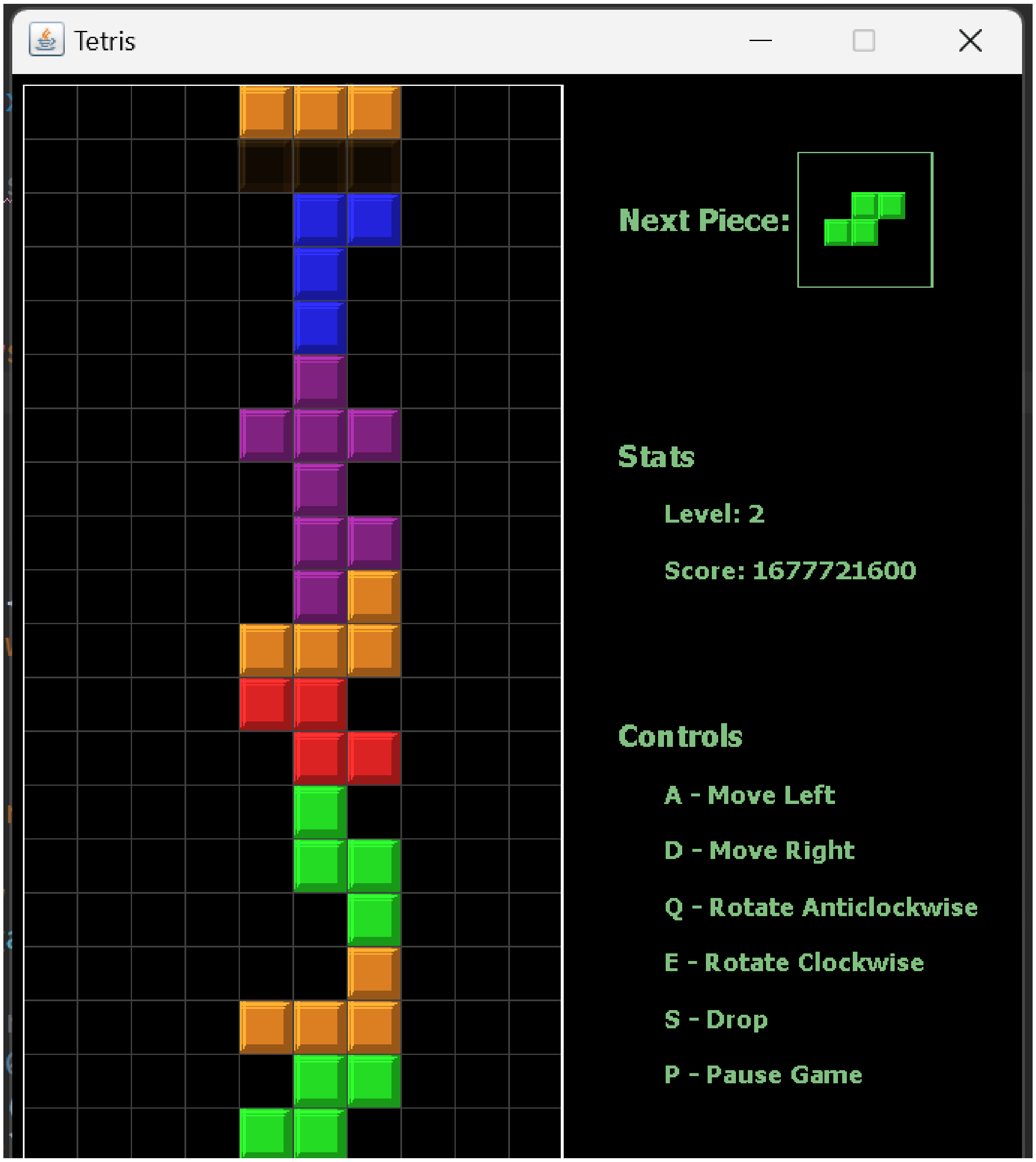
***Legend**
F = first row
M = middle row
L = last row

E = Empty Board
A = Board contains tiles

1 = Full row
0 = One of the columns are empty

2. **Test results and analysis**: what problem(s) did you find in the code? For each problem, further explain how you found it (e.g., using which test case). What is your JaCoCo code coverage score? How did your team improve your coverage score? Include screenshots of JaCoCo test reports.

After various testing and observations, we encountered one bug within the Graphic User Interface (GUI). During testing and playing the game, we all visualized the issue of the score when a piece would fall into the board. It would be clear that the score would be inconsistent during the longevity of the game. Initially our group thought that the score may not be correct due to the checkLines method in board panel with returning the incorrect boolean. Initially we set it as false in the method, however, we found out that the score would not be updated and be set as zero. The only exception that the score would be updated was clearing the whole row that adds up 50 points. The image below shows the bug.


Tetris

Next Piece:

Stats

Level: 2

Score: 1677721600

Controls

A - Move Left

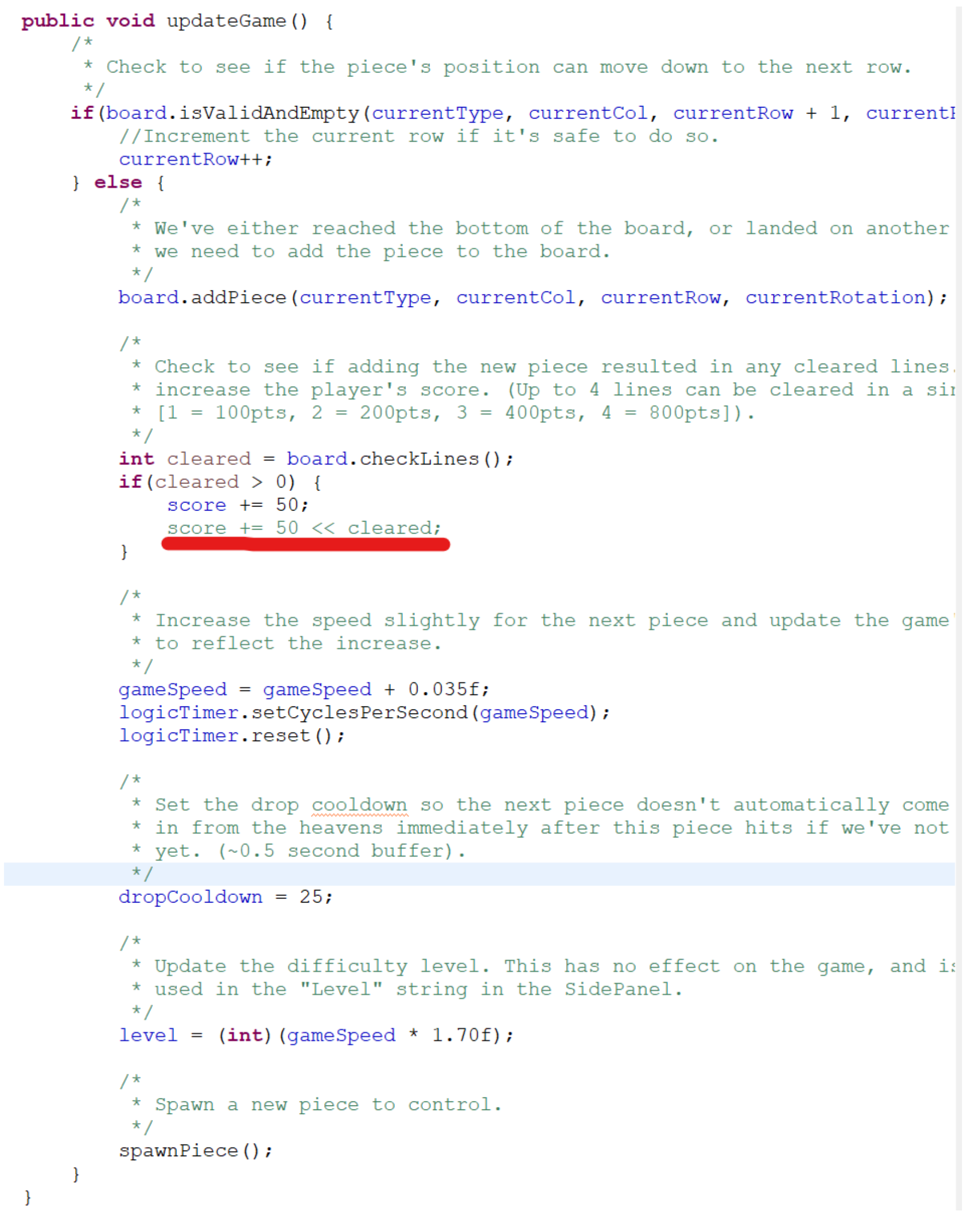
D - Move Right

Q - Rotate Anticlockwise

E - Rotate Clockwise

S - Drop

P - Pause Game

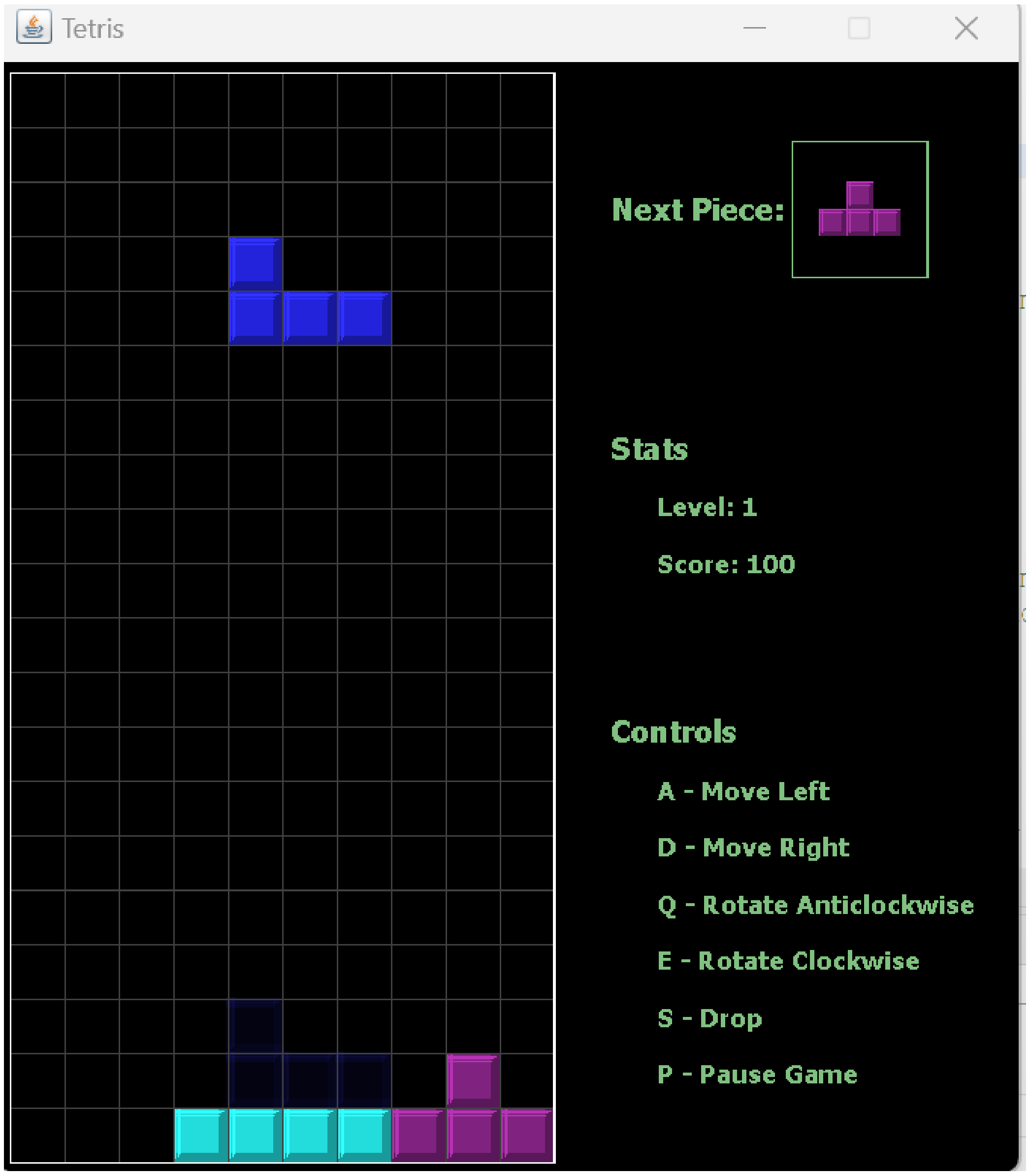


```java
public void updateGame() {
    /*
     * Check to see if the piece's position can move down to the next row.
     */
    if(board.isValidAndEmpty(currentType, currentCol, currentRow + 1, current
        //Increment the current row if it's safe to do so.
        currentRow++;
    } else {
        /*
         * We've either reached the bottom of the board, or landed on another
         * we need to add the piece to the board.
         */
        board.addPiece(currentType, currentCol, currentRow, currentRotation);

        /*
         * Check to see if adding the new piece resulted in any cleared lines.
         * increase the player's score. (Up to 4 lines can be cleared in a si
         * [1 = 100pts, 2 = 200pts, 3 = 400pts, 4 = 800pts]).
         */
        int cleared = board.checkLines();
        if(cleared > 0) {
            score += 50;
            score += 50 << cleared;
        }

        /*
         * Increase the speed slightly for the next piece and update the game
         * to reflect the increase.
         */
        gameSpeed = gameSpeed + 0.035f;
        logicTimer.setCyclesPerSecond(gameSpeed);
        logicTimer.reset();

        /*
         * Set the drop cooldown so the next piece doesn't automatically come
         * in from the heavens immediately after this piece hits if we've not
         * yet. (~0.5 second buffer).
         */
        dropCooldown = 25;

        /*
         * Update the difficulty level. This has no effect on the game, and i
         * used in the "Level" string in the SidePanel.
         */
        level = (int)(gameSpeed * 1.70f);

        /*
         * Spawn a new piece to control.
         */
        spawnPiece();
    }
}
```

A problem we found was in **public void updateGame()** in Tetris.java where the score is increased incorrectly. We found this problem using the testUpdate() method in our test class TestTetris.java when testing the test score as well as remodifying the updateGame() method in Tetris.java.

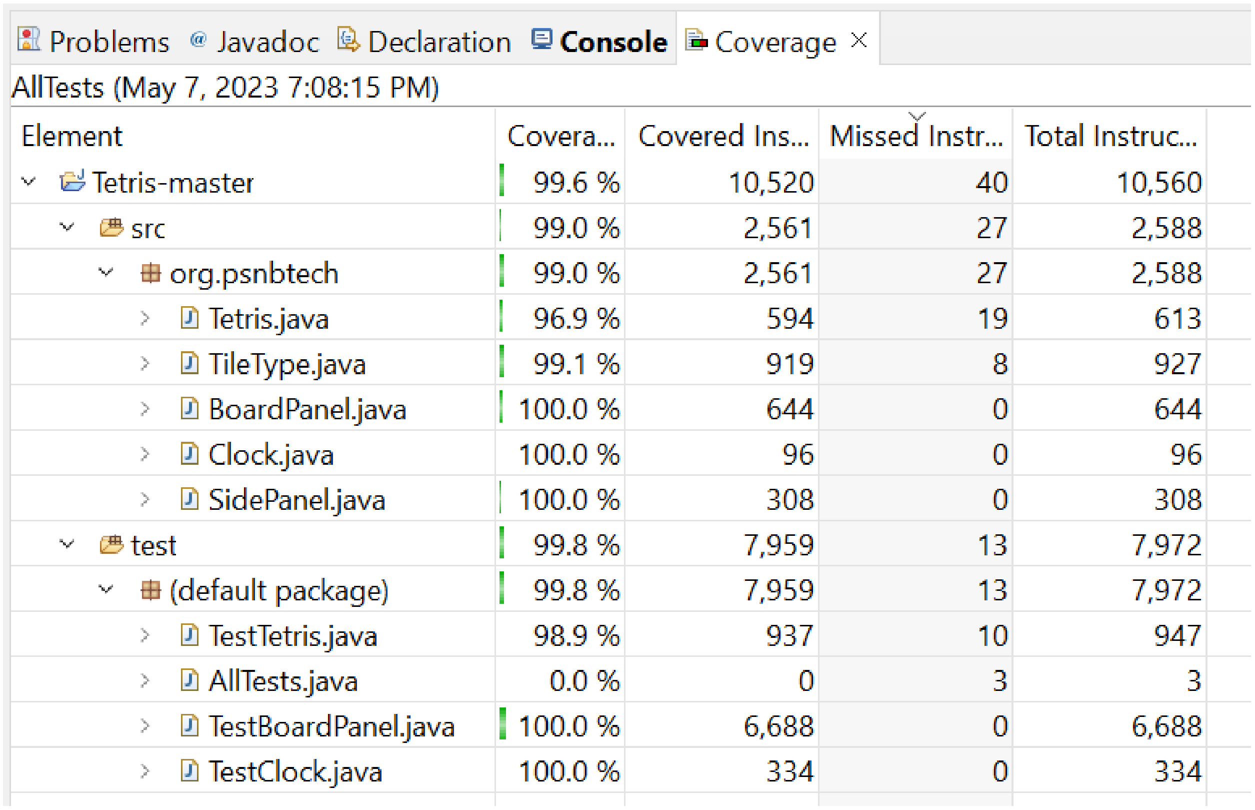
Underlined on red indicated the original code that caused the bug to show the scores to be scaled randomly. The line of code above is the modified version of keeping track of the score in accord.

Below is the GUI output for scores to not be scaled around the millions, but after every piece to record an additional score of 50 points after every drop the piece lands on the board.



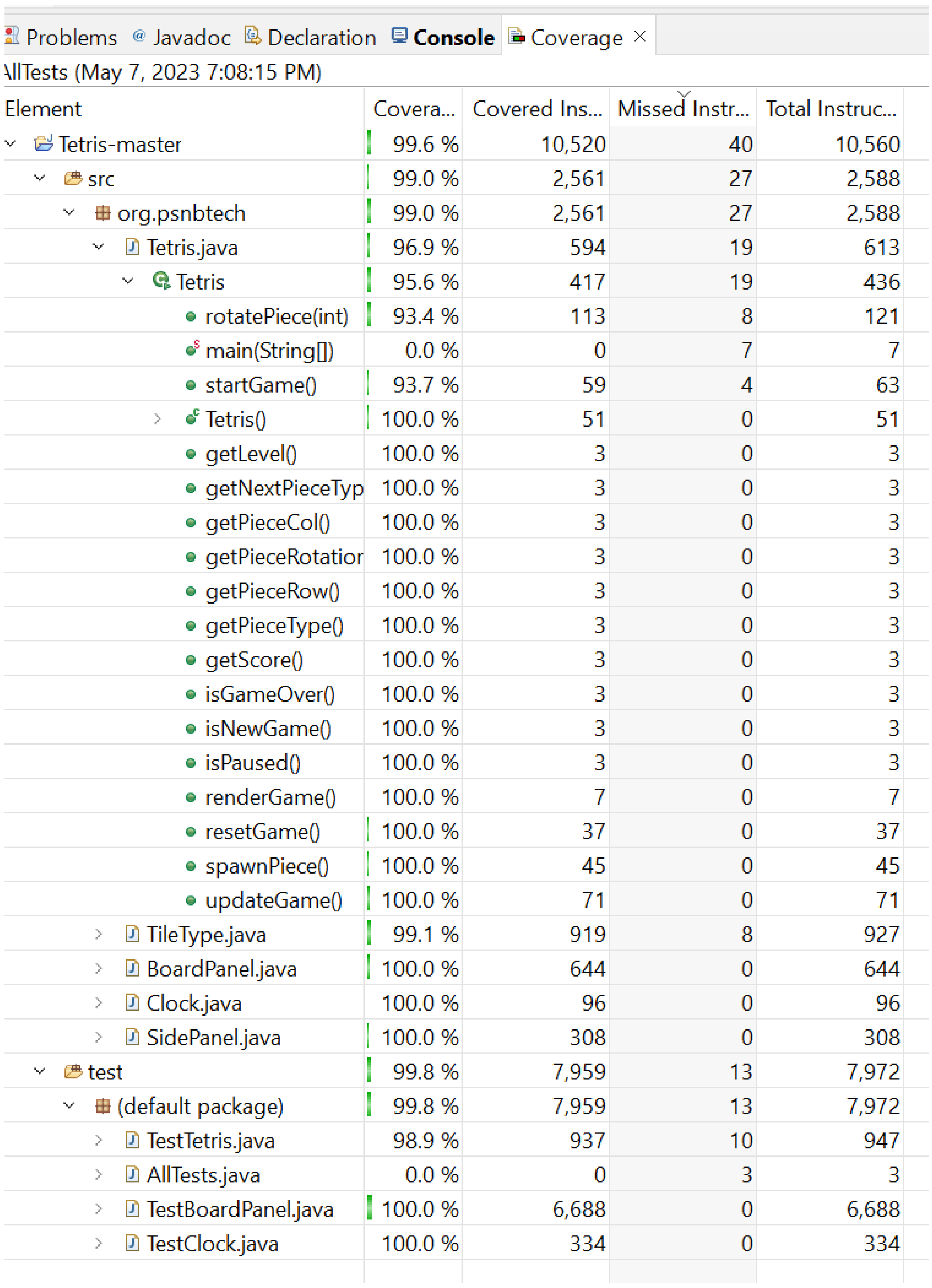
Our JaCoCo code coverage score is around 99.4 to 99.6% or around 99.0% for the src code only. Our team improved our coverage score by using a Java robot to automatically interact with the game and ensure the internal state is updating appropriately.

With multiple attempts, we learned that our robot would not be consistent with our JaCoCo score due to the delay of the robot interaction and as well as the tetris pieces that land onto the board.



AllTests (May 7, 2023 7:08:15 PM)

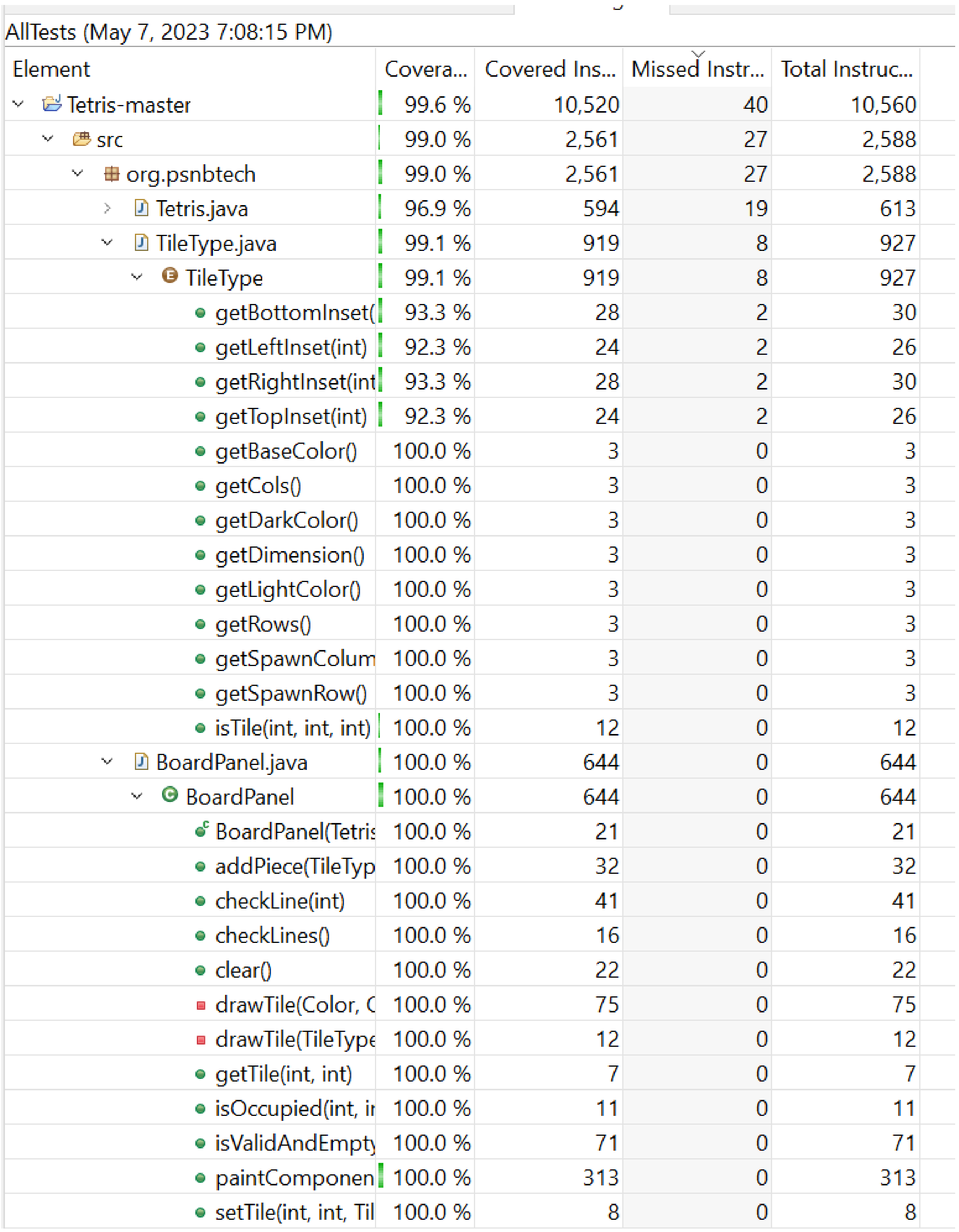
Element	Covera...	Covered Ins...	Missed Instr...	Total Instruc...
Tetris-master	99.6 %	10,520	40	10,560
src	99.0 %	2,561	27	2,588
org.psnbtech	99.0 %	2,561	27	2,588
Tetris.java	96.9 %	594	19	613
TileType.java	99.1 %	919	8	927
BoardPanel.java	100.0 %	644	0	644
Clock.java	100.0 %	96	0	96
SidePanel.java	100.0 %	308	0	308
test	99.8 %	7,959	13	7,972
(default package)	99.8 %	7,959	13	7,972
TestTetris.java	98.9 %	937	10	947
AllTests.java	0.0 %	0	3	3
TestBoardPanel.java	100.0 %	6,688	0	6,688
TestClock.java	100.0 %	334	0	334



Problems @ Javadoc Declaration **Console** Coverage ×

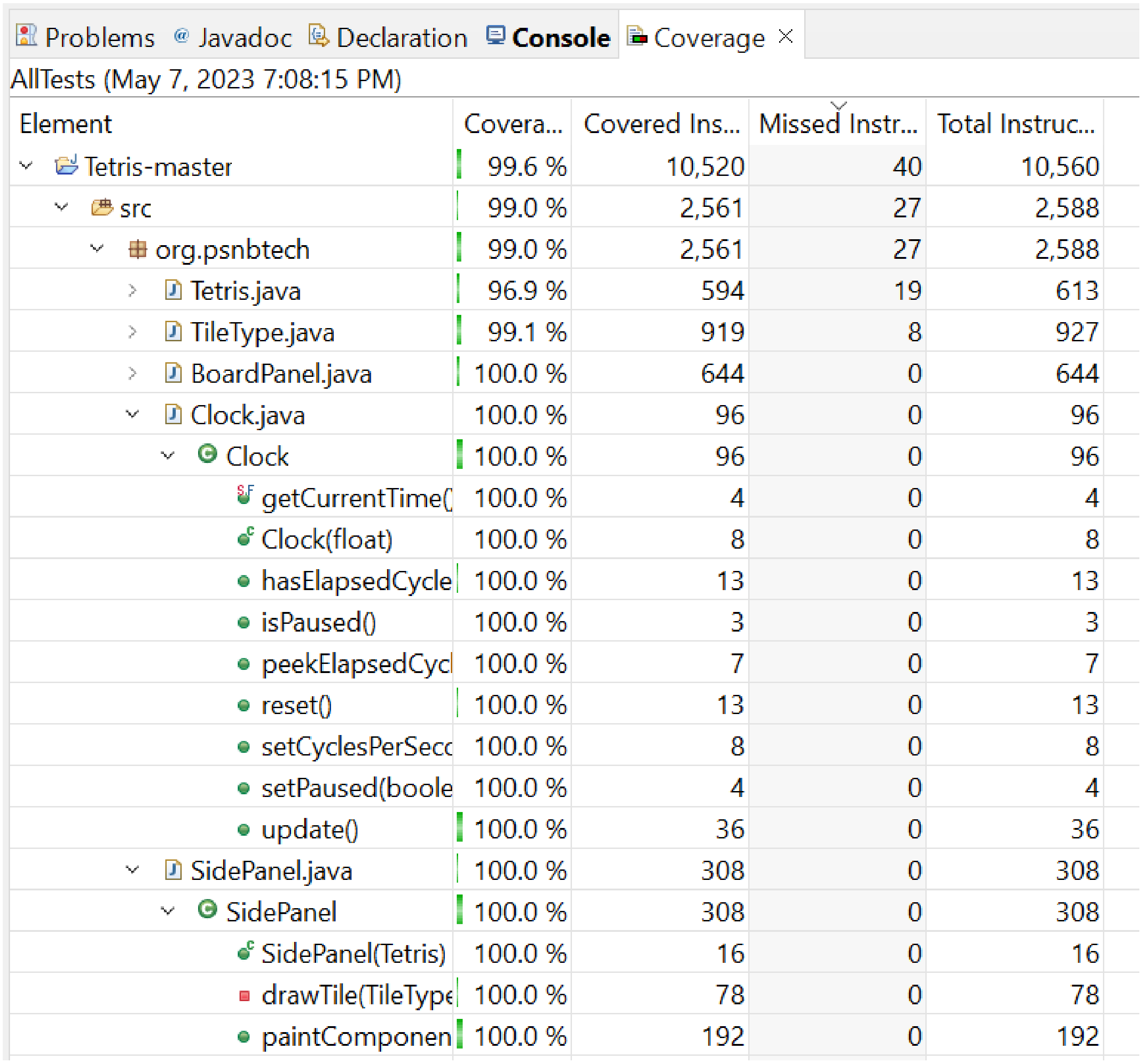
AllTests (May 7, 2023 7:08:15 PM)

Element	Covera...	Covered Ins...	Missed Instr...	Total Instruc...
Tetris-master	99.6 %	10,520	40	10,560
src	99.0 %	2,561	27	2,588
org.psnbtech	99.0 %	2,561	27	2,588
Tetris.java	96.9 %	594	19	613
Tetris	95.6 %	417	19	436
rotatePiece(int)	93.4 %	113	8	121
main(String[])	0.0 %	0	7	7
startGame()	93.7 %	59	4	63
Tetris()	100.0 %	51	0	51
getLevel()	100.0 %	3	0	3
getNextPieceTyp	100.0 %	3	0	3
getPieceCol()	100.0 %	3	0	3
getPieceRotatior	100.0 %	3	0	3
getPieceRow()	100.0 %	3	0	3
getPieceType()	100.0 %	3	0	3
getScore()	100.0 %	3	0	3
isGameOver()	100.0 %	3	0	3
isNewGame()	100.0 %	3	0	3
isPaused()	100.0 %	3	0	3
renderGame()	100.0 %	7	0	7
resetGame()	100.0 %	37	0	37
spawnPiece()	100.0 %	45	0	45
updateGame()	100.0 %	71	0	71
TileType.java	99.1 %	919	8	927
BoardPanel.java	100.0 %	644	0	644
Clock.java	100.0 %	96	0	96
SidePanel.java	100.0 %	308	0	308
test	99.8 %	7,959	13	7,972
(default package)	99.8 %	7,959	13	7,972
TestTetris.java	98.9 %	937	10	947
AllTests.java	0.0 %	0	3	3
TestBoardPanel.java	100.0 %	6,688	0	6,688
TestClock.java	100.0 %	334	0	334



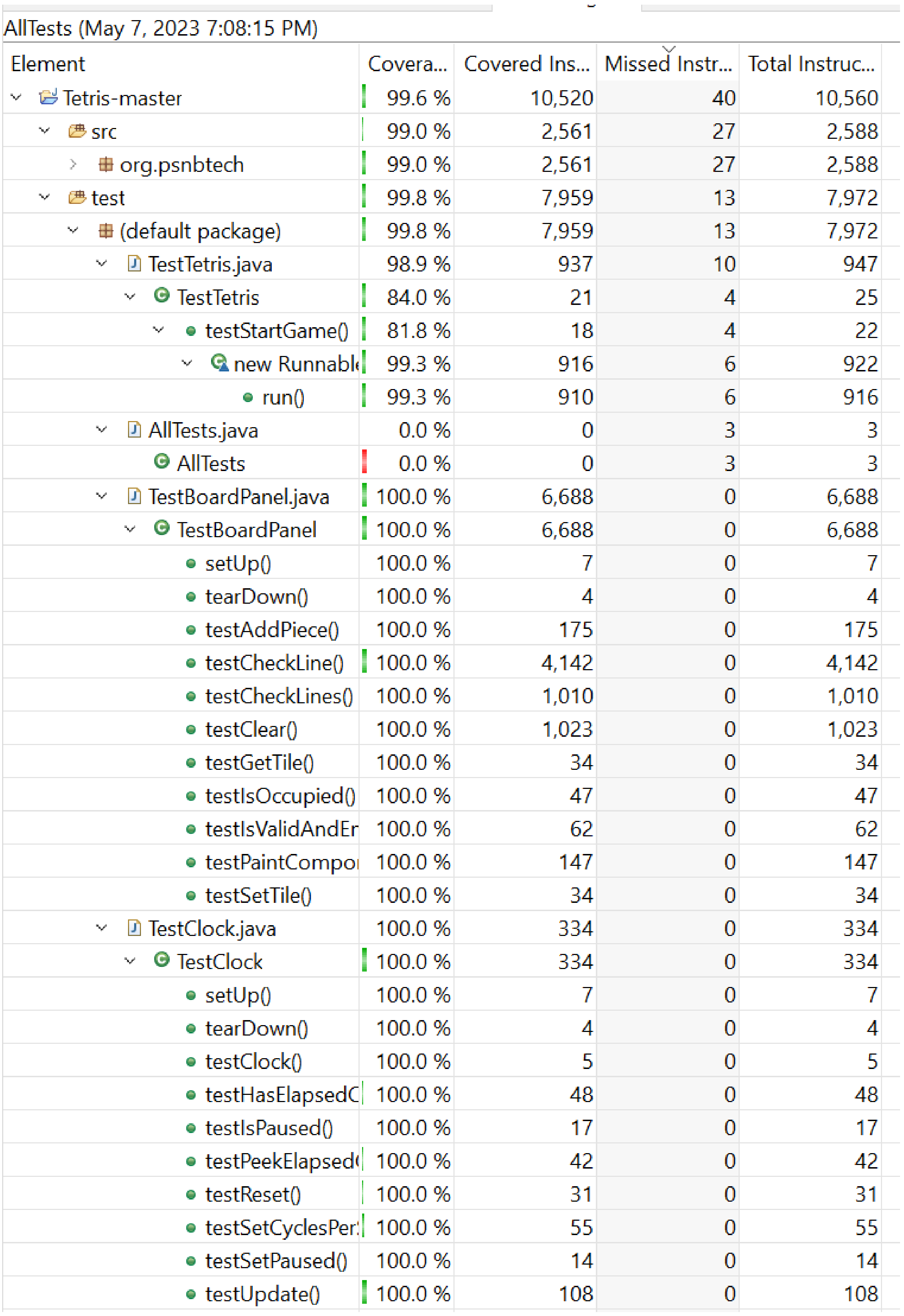
AllTests (May 7, 2023 7:08:15 PM)

Element	Covera...	Covered Ins...	Missed Instr...	Total Instruc...
Tetris-master	99.6 %	10,520	40	10,560
src	99.0 %	2,561	27	2,588
org.psnbtech	99.0 %	2,561	27	2,588
Tetris.java	96.9 %	594	19	613
TileType.java	99.1 %	919	8	927
TileType	99.1 %	919	8	927
getBottomInset(	93.3 %	28	2	30
getLeftInset(int)	92.3 %	24	2	26
getRightInset(int	93.3 %	28	2	30
getTopInset(int)	92.3 %	24	2	26
getBaseColor()	100.0 %	3	0	3
getCols()	100.0 %	3	0	3
getDarkColor()	100.0 %	3	0	3
getDimension()	100.0 %	3	0	3
getLightColor()	100.0 %	3	0	3
getRows()	100.0 %	3	0	3
getSpawnColum	100.0 %	3	0	3
getSpawnRow()	100.0 %	3	0	3
isTile(int, int, int)	100.0 %	12	0	12
BoardPanel.java	100.0 %	644	0	644
BoardPanel	100.0 %	644	0	644
BoardPanel(Tetris	100.0 %	21	0	21
addPiece(TileTyp	100.0 %	32	0	32
checkLine(int)	100.0 %	41	0	41
checkLines()	100.0 %	16	0	16
clear()	100.0 %	22	0	22
drawTile(Color, C	100.0 %	75	0	75
drawTile(TileType	100.0 %	12	0	12
getTile(int, int)	100.0 %	7	0	7
isOccupied(int, ir	100.0 %	11	0	11
isValidAndEmpty	100.0 %	71	0	71
paintComponen	100.0 %	313	0	313
setTile(int, int, Til	100.0 %	8	0	8



AllTests (May 7, 2023 7:08:15 PM)

Element	Covera...	Covered Ins...	Missed Instr...	Total Instruc...
Tetris-master	99.6 %	10,520	40	10,560
src	99.0 %	2,561	27	2,588
org.psnbtech	99.0 %	2,561	27	2,588
Tetris.java	96.9 %	594	19	613
TileType.java	99.1 %	919	8	927
BoardPanel.java	100.0 %	644	0	644
Clock.java	100.0 %	96	0	96
Clock	100.0 %	96	0	96
getCurrentTime(	100.0 %	4	0	4
Clock(float)	100.0 %	8	0	8
hasElapsedCycle	100.0 %	13	0	13
isPaused()	100.0 %	3	0	3
peekElapsedCycl	100.0 %	7	0	7
reset()	100.0 %	13	0	13
setCyclesPerSecc	100.0 %	8	0	8
setPaused(boole	100.0 %	4	0	4
update()	100.0 %	36	0	36
SidePanel.java	100.0 %	308	0	308
SidePanel	100.0 %	308	0	308
SidePanel(Tetris)	100.0 %	16	0	16
drawTile(TileType	100.0 %	78	0	78
paintComponen	100.0 %	192	0	192



AllTests (May 7, 2023 7:08:15 PM)

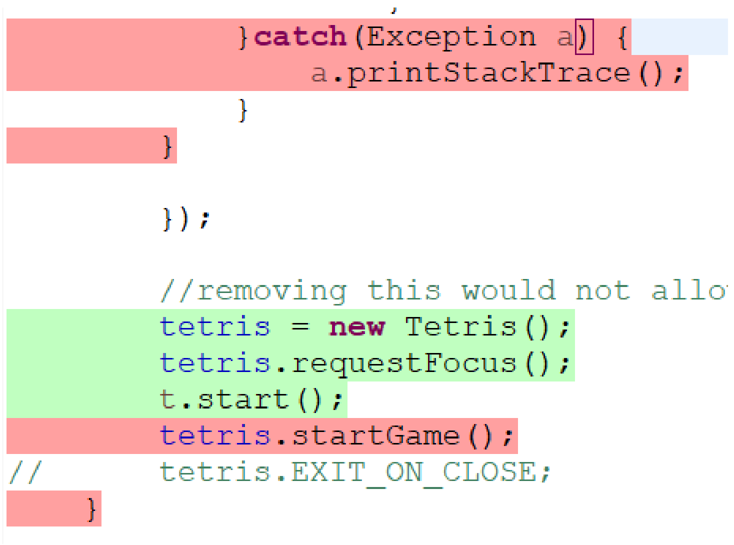
Element	Covera...	Covered Ins...	Missed Instr...	Total Instruc...
Tetris-master	99.6 %	10,520	40	10,560
src	99.0 %	2,561	27	2,588
org.psnbtech	99.0 %	2,561	27	2,588
test	99.8 %	7,959	13	7,972
(default package)	99.8 %	7,959	13	7,972
TestTetris.java	98.9 %	937	10	947
TestTetris	84.0 %	21	4	25
testStartGame()	81.8 %	18	4	22
new Runnabl	99.3 %	916	6	922
run()	99.3 %	910	6	916
AllTests.java	0.0 %	0	3	3
AllTests	0.0 %	0	3	3
TestBoardPanel.java	100.0 %	6,688	0	6,688
TestBoardPanel	100.0 %	6,688	0	6,688
setUp()	100.0 %	7	0	7
tearDown()	100.0 %	4	0	4
testAddPiece()	100.0 %	175	0	175
testCheckLine()	100.0 %	4,142	0	4,142
testCheckLines()	100.0 %	1,010	0	1,010
testClear()	100.0 %	1,023	0	1,023
testGetTile()	100.0 %	34	0	34
testIsOccupied()	100.0 %	47	0	47
testIsValidAndEr	100.0 %	62	0	62
testPaintCompo	100.0 %	147	0	147
testSetTile()	100.0 %	34	0	34
TestClock.java	100.0 %	334	0	334
TestClock	100.0 %	334	0	334
setUp()	100.0 %	7	0	7
tearDown()	100.0 %	4	0	4
testClock()	100.0 %	5	0	5
testHasElapsedC	100.0 %	48	0	48
testIsPaused()	100.0 %	17	0	17
testPeekElapsed	100.0 %	42	0	42
testReset()	100.0 %	31	0	31
testSetCyclesPer	100.0 %	55	0	55
testSetPaused()	100.0 %	14	0	14
testUpdate()	100.0 %	108	0	108

With our expertise, we were unsuccessful to cover all possible branches on the tetris due to the total of instructions and coverages we had to consider. While testing, we were able to identify that the rotatePiece method in Tetris to be inconsistent for satisfying all possible branches that is played in the game. We came to the conclusion that the reason why rotatePiece was inconsistent was due to the possibility of the pieces not landing correctly when rotated on a certain angle. An example of this is when there would be a piece rotated 90 degrees, however when it is about to be place down, the rotation would be set to go to a different angle, which would then give us a yellow coverage. In addition, the speed of the game would begin to affect the course of the game since the objective for the game is to complete all possible tests in the game versus the robot knowing how to play the game.

We begin to create tests cases for the following game by developing two games to allow the robot to satisfy more test cases needed to be covered within the game. While we let the robot play with the game, we learned that with the KeyEvents, it would be called and typed from the browser we are currently on which would confuse all of us at first. In addition, we would have to use different devices to work on our report/slides while the game was active. Later on we adjusted the following visibility in the main code from private to public to allow ourselves to test the code properly with JUnit and read the branch coverages to cover the rest of the instructions.

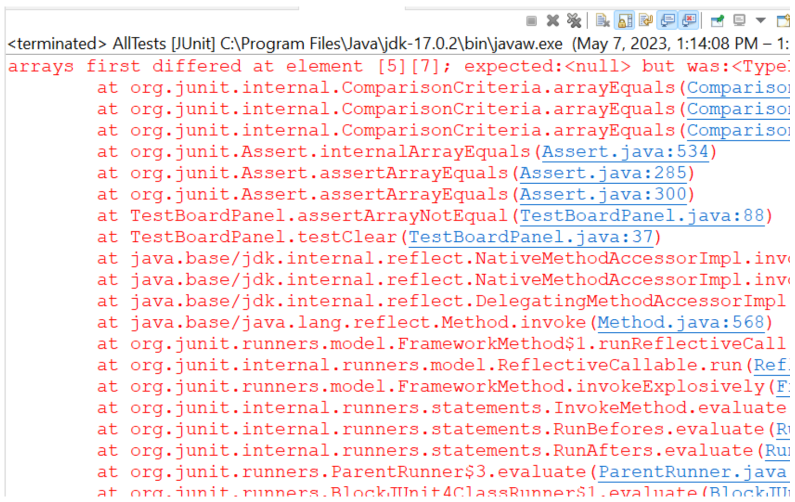
We would begin to learn that the game would begin to not work after a multitude of tests ran. We could not learn why we would not be able to run JUnit consistently given the changes we have not changed, but we would have to close out the IDE for us to reuse, which then resolved our issue.

Final Remarks:



```
                        }catch(Exception a) {
                            a.printStackTrace();
                        }
                }

                });

                //removing this would not allo
                tetris = new Tetris();
                tetris.requestFocus();
                t.start();
                tetris.startGame();
//              tetris.EXIT_ON_CLOSE;
        }
```

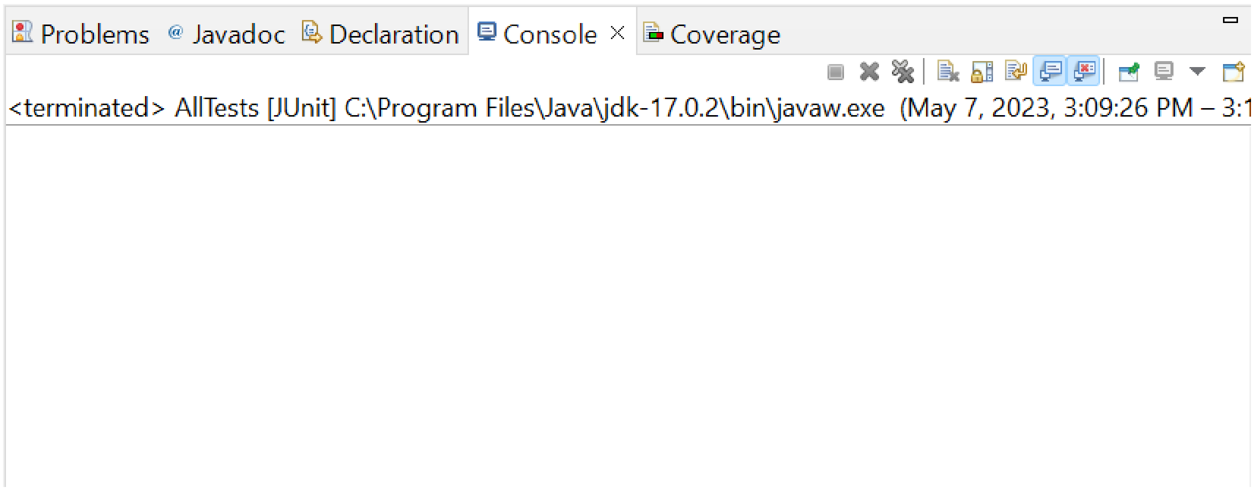
We were unable to resolve the issue for creating test cases from the main function of tetris. Doing so would require us to remodify the whole source code and redo the JUnit test cases. In addition, we came to the conclusion that we should only modify code that are faults rather than working code. Given our ultimate efforts to fully cover all possible branches in the code, we ended up keeping the score as it is.



```
<terminated> AllTests [JUnit] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe  (May 7, 2023, 1:14:08 PM – 1:
arrays first differed at element [5][7]; expected:<null> but was:<Type
	at org.junit.internal.ComparisonCriteria.arrayEquals(Compariso
	at org.junit.internal.ComparisonCriteria.arrayEquals(Compariso
	at org.junit.internal.ComparisonCriteria.arrayEquals(Compariso
	at org.junit.Assert.internalArrayEquals(Assert.java:534)
	at org.junit.Assert.assertArrayEquals(Assert.java:285)
	at org.junit.Assert.assertArrayEquals(Assert.java:300)
	at TestBoardPanel.assertArrayNotEqual(TestBoardPanel.java:88)
	at TestBoardPanel.testClear(TestBoardPanel.java:37)
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.inv
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.inv
	at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl
	at java.base/java.lang.reflect.Method.invoke(Method.java:568)
	at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall
	at org.junit.internal.runners.model.ReflectiveCallable.run(Ref
	at org.junit.runners.model.FrameworkMethod.invokeExplosively(F
	at org.junit.internal.runners.statements.InvokeMethod.evaluate
	at org.junit.internal.runners.statements.RunBefores.evaluate(R
	at org.junit.internal.runners.statements.RunAfters.evaluate(Ru
	at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java
	at org.junit.runners.BlockJUnit4ClassRunner$1.evaluate(BlockJU
```

We would also bump into console errors along the way due to JUnit reading every Tetris piece with the board to check the range of the piece within the game. In addition, we initially thought we had to debug our code, however, the JaCoCo would still record its coverage and we began to work on adding a variety of test cases to progress our coverage.

Later on we figured out that the console output would record the JUnit array to check if one value does not equal the other. We would comment out the comment and below shows the console output not displaying any errors, however keeping that would keep a great reference to track when the game reads the results as for when we see if the array would have the same values. Below is the following result.



```
<terminated> AllTests [JUnit] C:\Program Files\Java\jdk-17.0.2\bin\javaw.exe  (May 7, 2023, 3:09:26 PM – 3:1
```

References:
Down below are the sources we used to accumulate our ideas to complete the project.

- Projects that contained all CFG
- Robot Class Source 1
- Robot Class Source 2

- STQ02 (Chp 1 & 2.1)
- STQ03 (Chp 5)
- STQ05 (Chp 7.2.1)
- STQ07 (Chp 7.3.1)
- STQ09 (Chp 9.2) → a little bit with changing score
- STQ11
- STQ13 (Ch 6.2)
- JUnit Tutorial ⇒ LunarLandar/SpaceController
- https://www.vogella.com/tutorials/JUnit4/article.html
- Assignment 3
- https://www.w3schools.com/java/java_threads.asp
- https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html
- https://github.com/nagalil-honaj-hteal3/SE471/tree/main/Lab0 (My Repository)
- SE461 Project Slides