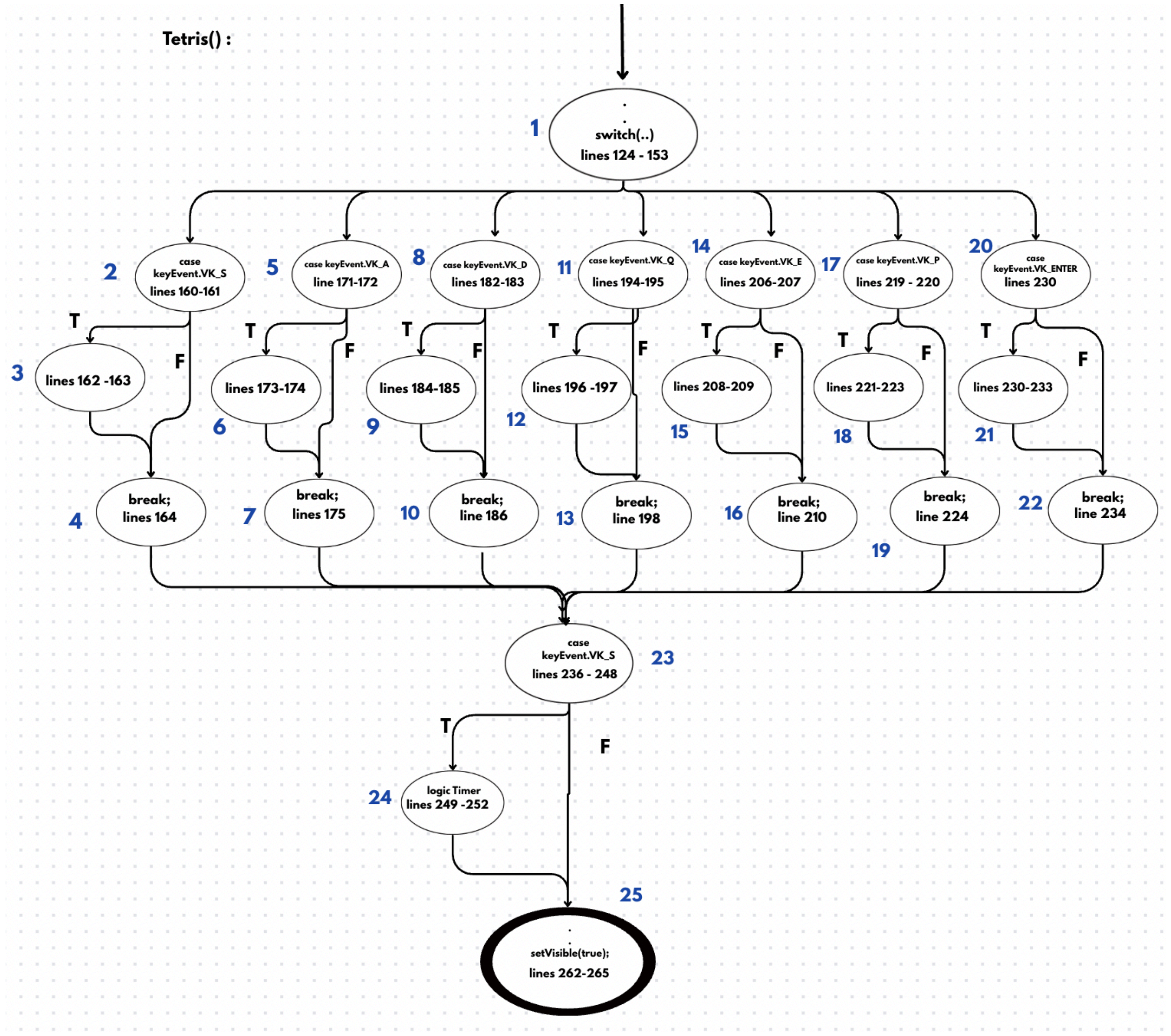
461 Project Report

Beau Dougan
Montserrat Jara

Test Design:

1. Tetris:

CFG:



TR PPC Set: {[1,2,3,4,23,24,25], [1,2,3,4,23,25] ,[1,2,4,23,24,25],[1,2,4,23,25],[1,5,6,7,23,24,25],[1,5,6,7,23,25], [1,5,7,23,24,25], [1,5,7,23,25],[1,8,9,10,23,24,25], [1,8,9,10,23,25], [1,8,10,23,24,25],[1,8,10,23,25],[1,11,12,13,23,24,25],[1,11,12,13,23,25],[1,11,13,23,24,25],[1,11 ,13,23,25],[1,14,15,16,23,24,25],[1,14,15,16,23,25],[1,14,16,23,24,25], [1,14,16,23,25],

[1,17,18,19,23,24,25],[1,17,18,19,23,25], [1,17,19,23,24,25],
[1,17,19,23,25],[1,20,21,22,23,24,25], [1,20,21,22,23,25], [1,20,22,23,24,25], [1,20,22,23,25]}

The highlighted test cases are unfeasible using prime path coverage. Due to the S key not being pressed during those cases it is not feasible to release it in the same case.

testTetris_A1()

Test Case: [1,5,6,7,23,25]
Input:

```
@Test
        void testTetris_A1() throws Exception {

                gameThread.start();

                Thread.sleep(1000);

                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(500);

                robot.keyPress(KeyEvent.VK_A);
                robot.keyRelease(KeyEvent.VK_A);

                Thread.sleep(500);

                int check = tetris.getPieceCol();
                int test = tetris.getPieceType().getSpawnColumn();

                assertEquals(check, --test);
        }
```

Output:
Expect true because test and check should be the same before test is decremented.
Expected output: True
Actual output: True
Test Path: Covers path [1,5,6,7,23,25] and is part of Prime Path Coverage for Tetris()

testTetris_A2()
Test Case: [1,5,7,23,25]
Input:

```
@Test
	void testTetris_A2() throws Exception {
		gameThread.start();

		Thread.sleep(1000);

		robot.keyPress(KeyEvent.VK_ENTER);
		robot.keyRelease(KeyEvent.VK_ENTER);

		Thread.sleep(200);

		robot.keyPress(KeyEvent.VK_P);
		robot.keyRelease(KeyEvent.VK_P);

		Thread.sleep(500);

		robot.keyPress(KeyEvent.VK_A);
		robot.keyRelease(KeyEvent.VK_A);

		Thread.sleep(500);
		int test = tetris.getPieceType().getSpawnColumn();
		int check = tetris.getPieceCol();

		assertEquals(check, test);
	}
```
Output:
Expect true because check and test should be the same because the game is paused.
	Expected: True
	Actual: True
Test Path: Covers path [1,5,6,7,23,25] and is part of Prime Path Coverage for Tetris()

testTetris_Q1()
Test Case: [1,14,15,16,23,25]
Input:
```
@Test
	void testTetris_Q1() throws Exception{
		gameThread.start();

		Thread.sleep(1000);

		robot.keyPress(KeyEvent.VK_ENTER);
		robot.keyRelease(KeyEvent.VK_ENTER);
```

```
                Thread.sleep(500);

                robot.keyPress(KeyEvent.VK_Q);
                robot.keyRelease(KeyEvent.VK_Q);

                Thread.sleep(500);

                int check = tetris.getPieceRotation();
                assertEquals(check, 3);
        }
```

Output:
Expected output should be true because the piece should rotate anti-clockwise and should equal 3.

Expected: True
Actual : True

Test Path: Covers path [1,14,15,16,23,25] and is a part of Prime Path Coverage for Tetris()

testTetris_Q2()
Test Case: [1,14,16,23,25]
Input:

```
@Test
        void testTetris_Q2() throws Exception{
                gameThread.start();

                Thread.sleep(1000);

                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(200);

                robot.keyPress(KeyEvent.VK_P);
                robot.keyRelease(KeyEvent.VK_P);

                Thread.sleep(500);

                robot.keyPress(KeyEvent.VK_Q);
                robot.keyRelease(KeyEvent.VK_Q);

                Thread.sleep(500);

                int check = tetris.getPieceRotation();
                assertEquals(check, 0);
        }
```
Output:
Expected output is true because the piece should not be able to rotate when the game is paused so rotation should be zero.

Expected: True

Actual: True

Test Path: Covers path [1,14,16,23,25] and is a part of Prime Path Coverage for Tetris()

testTetris_E1()
Test Case: [1,14,15,16,23,25]
Input:
```
@Test
        void testTetris_E1() throws Exception{
                gameThread.start();

                Thread.sleep(1000);

                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(500);

                robot.keyPress(KeyEvent.VK_E);
                robot.keyRelease(KeyEvent.VK_E);

                Thread.sleep(500);

                int check = tetris.getPieceRotation();
                assertEquals(check, 1);
        }
```

Output:
Expected true because the piece should rotate clockwise and expected rotation is 1.

Expected:True

Actual:True

Test Path: Covers path [1,14,15,16,23,25] and is a part of Prime Path Coverage for Tetris().

testTetris_E2()
Test Case: [1,14,16,23,25]
Input:
```
@Test
        void testTetris_E2() throws Exception{
                gameThread.start();
                Thread.sleep(1000);

                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(200);

                robot.keyPress(KeyEvent.VK_P);
                robot.keyRelease(KeyEvent.VK_P);

                Thread.sleep(500);

                robot.keyPress(KeyEvent.VK_E);
                robot.keyRelease(KeyEvent.VK_E);

                Thread.sleep(500);

                int check = tetris.getPieceRotation();

                assertEquals(check, 0);
        }
```
Output:
Expected outcome is true because the piece should not rotate when the game is paused so expected rotation is 0.

Expected: True

Actual: True

Test Path: Covers path [1,14,16,23,25] and is part of Prime Path Coverage for Tetris().

testTetris_Enter1()
Test Case: [1,20,21,22,23,25]
Input:
```
@Test
        void testTetris_Enter1() throws Exception{
                gameThread.start();

                Thread.sleep(1000);

                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(500);

                assertFalse(tetris.isGameOver());
        }
```
Output:
Expected outcome is false because after enter is pressed it should be a new game so isGameOver() should be false.

Expected: False

Actual: False

Test Path: Covers path [1,20,21,22,23,25] and is part of Prime Path Coverage for Tetris().

testTetris_Enter2()
Test Case: [1,20,22,23,25]
Input:
```
@Test
        void testTetris_Enter2() throws Exception{
                gameThread.start();

                Thread.sleep(1000);
                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(500);

                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                assertFalse(tetris.isNewGame());
        }
```
Output:
Expected outcome is false because if enter is pressed after a game has started then it is not a new game.

Expected: False

Actual: False

Test Path: Covers path [1,20,22,23,25] and is a part of Prime Path Coverage for Tetris().

testTetris_S1()
Test Case: [1,2,3,4,23,24,25]
Input:
```
@Test
   void testTetris_S1() throws Exception {

      gameThread.start();

      Thread.sleep(500);
```

```
        robot.keyPress(KeyEvent.VK_ENTER);
        robot.keyRelease(KeyEvent.VK_ENTER);

        Thread.sleep(200);

        robot.keyPress(KeyEvent.VK_S);
        robot.keyRelease(KeyEvent.VK_S);

        Thread.sleep(200);

        assertFalse(tetris.isPaused());
        //expects is paused to be false
    }
```

Output:
Expected to be False because the game is not paused when S is pressed.

    Expected: False
    Actual: False

Test Path: Covers path [1,2,3,4,23,24,25] and is a part of Prime Path Coverage for Tetris().

testTetris_S2()
Test Case: [1,2,3,4,23,25]
Input:

```
@Test
    void testTetris_S2() throws Exception {

        gameThread.start();

        Thread.sleep(500);

        robot.keyPress(KeyEvent.VK_ENTER);
        robot.keyRelease(KeyEvent.VK_ENTER);

        Thread.sleep(200);
        robot.keyPress(KeyEvent.VK_P);
        Thread.sleep(500);

        robot.keyPress(KeyEvent.VK_S);

        Thread.sleep(200);

        assertTrue(tetris.isPaused());
    }
```

Output:
Expected true because the game should still be paused meaning that S is never pressed.
    Expected:True
    Actual: True
Test Path: Covers path [1,2,3,4,23,25] and is a part of Prime Path Coverage for Tetris().

testTetris_S3()
Test Case: Test Case: [1,2,4,23,24,25]
Input:
```
@Test
   void testTetris_S3() throws Exception {

       gameThread.start();

       Thread.sleep(500);

       robot.keyPress(KeyEvent.VK_ENTER);
       robot.keyRelease(KeyEvent.VK_ENTER);

       Thread.sleep(200);
       robot.keyPress(KeyEvent.VK_P);
       Thread.sleep(500);

       robot.keyPress(KeyEvent.VK_S);
       robot.keyRelease(KeyEvent.VK_S);

       Thread.sleep(200);

       assertTrue(tetris.isPaused());
   }
```
Output:
Expected true because the game is paused and s is unable to be pressed.
    Expected: True
    Actual: True
Test Path: Covers path [1,2,4,23,24,25] and is part of Prime Path Coverage for Tetris().

testTetris_S4()
Test Case: Test Case: [1,2,4,23,25]
Input:
```
@Test
   void testTetris_S4() throws Exception {
        gameThread.start();

        Thread.sleep(500);

        robot.keyPress(KeyEvent.VK_ENTER);
        robot.keyRelease(KeyEvent.VK_ENTER);

        Thread.sleep(200);

        robot.keyPress(KeyEvent.VK_S);

        Thread.sleep(200);

        assertFalse(tetris.isPaused());

    }
```

Output:
The Expected output is False because the game is paused and you are able to press on the s button without releasing.
Test Path: covers path [1,2,4,23,25] and is part of prime path coverage for Tetris()

testTetris_D1()
Test Case: [1,8,9,10,23,25]
Input:

```
@Test
    void testTetris_D1() throws Exception {

        gameThread.start();

        Thread.sleep(500);

        robot.keyPress(KeyEvent.VK_ENTER);
        robot.keyRelease(KeyEvent.VK_ENTER);

        Thread.sleep(200);


        robot.keyPress(KeyEvent.VK_D);
        robot.keyRelease(KeyEvent.VK_D);

        Thread.sleep(200);
```

```
      int test = tetris.getPieceType().getSpawnColumn();

      assertTrue(tetris.getPieceCol() == ++test );

   }
```

Output: The expected outcome is True because the column should be updated.
Expected: True
Actual: True
Test Path: This path covers [1,8,9,10,23,25] and is part of Prime Path Coverage for Tetris()

testTetris_D2()
Test Case: [1,8,10,23,25]
Input:

```
@Test
   void testTetris_D2() throws Exception {

      gameThread.start();

      Thread.sleep(500);

      robot.keyPress(KeyEvent.VK_ENTER);
      robot.keyRelease(KeyEvent.VK_ENTER);

      Thread.sleep(200);
      robot.keyPress(KeyEvent.VK_P);

      robot.keyPress(KeyEvent.VK_D);
      robot.keyRelease(KeyEvent.VK_D);

      Thread.sleep(200);

      int test = tetris.getPieceType().getSpawnColumn();
      assertTrue(tetris.getPieceCol() == test );

   }
```

Output:
The expected outcome is True because the column shouldn't change since the game is paused.
Expected: True
Actual:True
Test Path: This path covers [1,8,9,10,23,25] and is part of Prime Path Coverage for Tetris().

testTetris_P1()

Test Case: [1,17,18,19,23,25]
Input:
```
@Test
   void testTetris_P1() throws Exception {

       gameThread.start();

       Thread.sleep(500);

       robot.keyPress(KeyEvent.VK_ENTER);
       robot.keyRelease(KeyEvent.VK_ENTER);

       Thread.sleep(200);
       //robot.keyPress(KeyEvent.VK_P);

       robot.keyPress(KeyEvent.VK_P);
       robot.keyRelease(KeyEvent.VK_P);

       Thread.sleep(200);
       //4

       assertTrue(tetris.isPaused());

   }
```
Output:
The expected output is true because the game is paused.
Expected:True
Actual:True
Test Path: This covers path [1,17,18,19,23,25] and is part of Prime Path Coverage for Tetris()

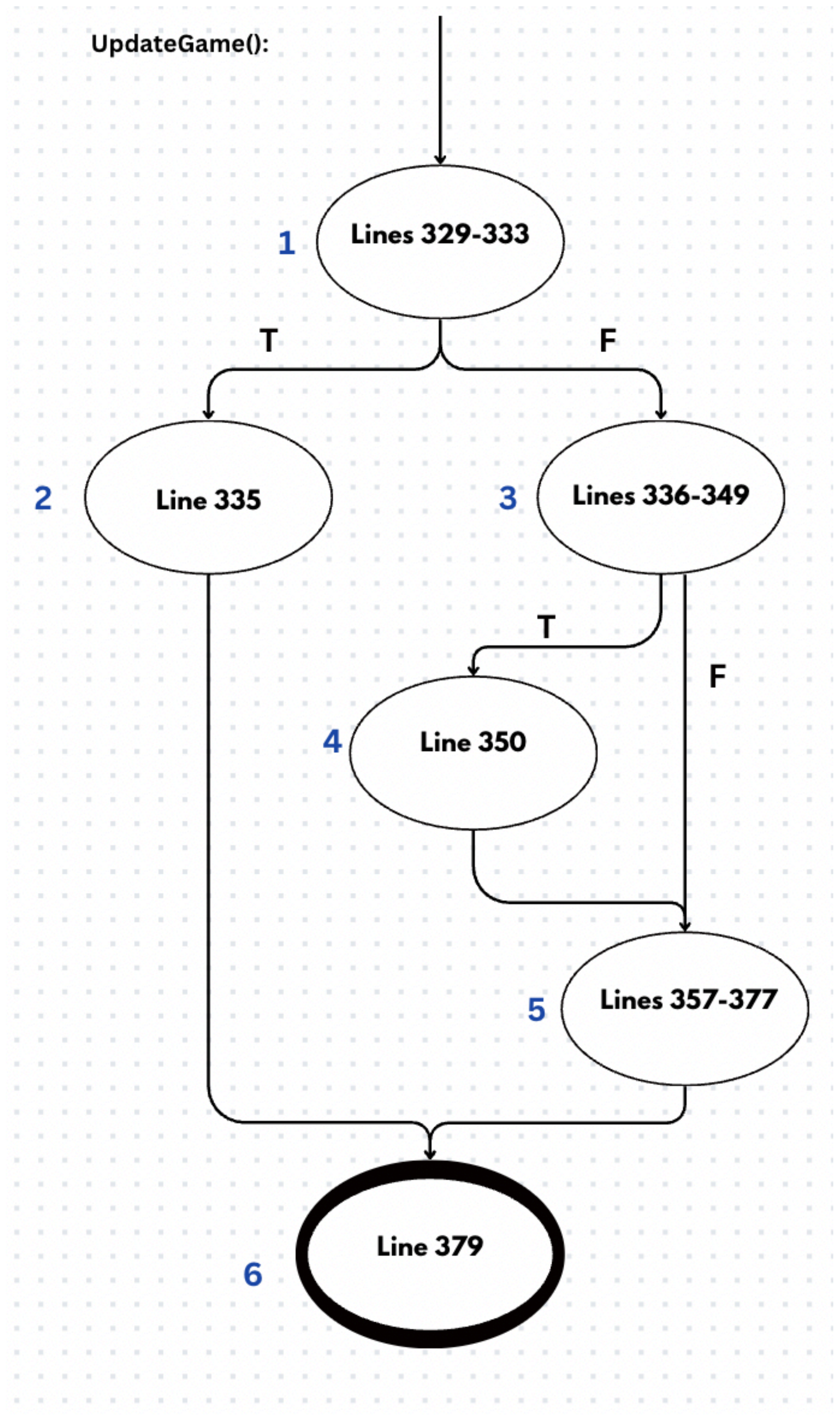
testTetris_P2()
Test Case: [1,17,18,19,23,25]
Input:
```
@Test
   void testTetris_P2() throws Exception {

       gameThread.start();

       Thread.sleep(500);
```

```
        //robot.keyPress(KeyEvent.VK_ENTER);
        //robot.keyRelease(KeyEvent.VK_ENTER);

       // Thread.sleep(200);
        //robot.keyPress(KeyEvent.VK_P);

        robot.keyPress(KeyEvent.VK_P);
        robot.keyRelease(KeyEvent.VK_P);

        Thread.sleep(200);
        //4

        assertFalse(tetris.isPaused());

    }
```

Output:
Expects false because we expect the game to be paused.
Expected:False
Actual:False
Test Path: This covers path [1,17,18,19,23,25] and is part of Prime Path Coverage for Tetris()

2. UpdateGame:

CFG:



TR EC Sets: {[1,2], [1,3], [2,6], [3,4], [3,5], [4,5], [5,6]}
Test Sets For Edge Coverage: {[1,3,5,6], [1,3,4,5,6], [1,2,6]}

updateGame_NoScore()

Test Case: [1,3], [3,5], [5,6]
Input:

```
@Test
	void updateGame_NoScore() throws Exception {

		gameThread.start();

		Thread.sleep(1000);

		robot.keyPress(KeyEvent.VK_ENTER);
		robot.keyRelease(KeyEvent.VK_ENTER);

		Thread.sleep(1000);

		robot.keyPress(KeyEvent.VK_S);
		Thread.sleep(1000);
		robot.keyRelease(KeyEvent.VK_S);

		Thread.sleep(1000);
		assertEquals(0, tetris.getScore());

	}
```

Output:

Expected Output: True
Actual Output: False

The expected output is supposed to be true because the score should be zero according to the rules printed in the commented area on lines 343 to 347. The score should be zero because a line is never cleared in this case. The score is increased due to a bug in the code somewhere where checking if lines have been cleared is done.

Test Path: Supposed to cover path [1,3,5,6] and is part of Edge Coverage for updateGame(). Instead due to a bug in the code it covers [1,3,4,5,6].

updateGame_Score()

Test Case: [1,3], [3,4], [4,5], [5,6]
Input:

```
@Test
	void updateGame_Score() throws Exception{
		gameThread.start();

		Thread.sleep(1000);
		robot.keyPress(KeyEvent.VK_ENTER);
		robot.keyRelease(KeyEvent.VK_ENTER);

		Thread.sleep(3000);

		for(int i = 0; i < 10; i++) {
			tetris.board.setTile(i, 21, TileType.TypeI);
		}

		Thread.sleep(500);
		robot.keyPress(KeyEvent.VK_S);
		Thread.sleep(1000);
		robot.keyRelease(KeyEvent.VK_S);
		Thread.sleep(200);

		assertEquals(tetris.getScore(), 100);
	}
```

Output:

	Expected Output: True
	Actual Output: False

The expected output is supposed to be true because if a line is cleared the score should be increased by 100 according to the rules printed in the commented area on lines 343 to 347. The result being false means there is a bug in this method when the score is calculated after a line has been cleared.

Test Path: Covers path [1,3,4,5,6] and is part of Edge Coverage for updateGame()

updateGame_CurrentRow()

Test Case: [1,2], [2,6]

Input:

```
	@Test
	void updateGame_currentRow() throws Exception {

		gameThread.start();

		Thread.sleep(200);

		robot.keyPress(KeyEvent.VK_ENTER);
		robot.keyRelease(KeyEvent.VK_ENTER);

		int oldRow = tetris.getPieceRow();

		Thread.sleep(1500);
```

```
                    assertTrue(tetris.getPieceRow() > oldRow);
          }
```
Output:

Expected Output: True

Actual Output: True

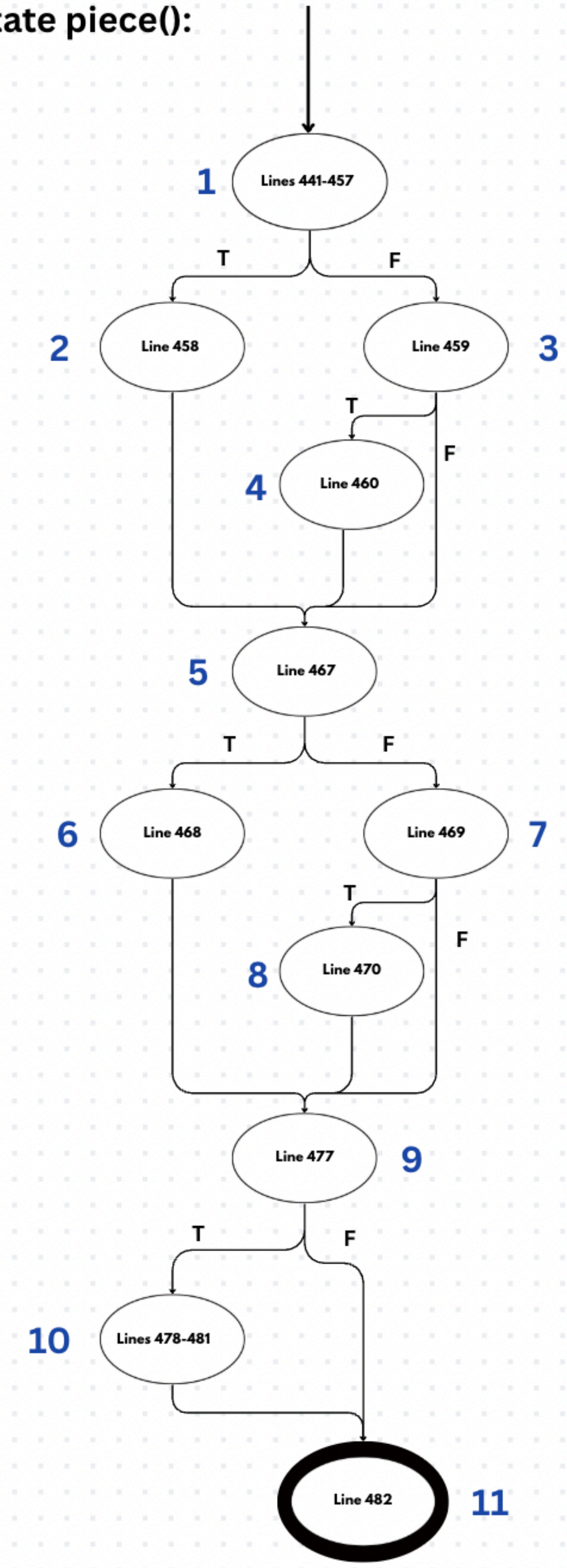
The expected output is supposed to be true because updateGame() should increment the row several times over the span of 1500 milliseconds. This test passed.

Test Path: Covers path [1,2,6] and is part of Edge Coverage for updateGame()

3. RotatePiece

CFG:

Rotate piece():



TR EC Sets: {[1,2], [1.3], [2,5], [3,4], [3,5], [4,5], [5,6], [5,7], [6,9], [7,8], [7,9], [8,9], [9,10], [9,11], [10,11]}

EC Test Paths: {[1,2,5,7,9,10,11], [1,3,4,5,7,9,10,11], [1,3,5,6,9,10,11], [1,3,5,7,8,9,10,11], [1,3,5,7,9,11]}

rotatePiece_blockedLeft()

Test Case: [1,2], [2,5], [5,7], [7,9], [9,10], [10,11]
Input:
```
@Test
        void rotatePiece_blockedLeft() throws Exception {
                gameThread.start();

                Thread.sleep(200);


                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(500);

                tetris.currentType = TileType.TypeI;

                Thread.sleep(200);

                robot.keyPress(KeyEvent.VK_Q);
                robot.keyRelease(KeyEvent.VK_Q);

                Thread.sleep(200);

                robot.keyPress(KeyEvent.VK_A);
                robot.keyRelease(KeyEvent.VK_A);

                Thread.sleep(200);

                robot.keyPress(KeyEvent.VK_A);
                robot.keyRelease(KeyEvent.VK_A);

                Thread.sleep(200);
```

```
            robot.keyPress(KeyEvent.VK_A);
            robot.keyRelease(KeyEvent.VK_A);

            Thread.sleep(200);

            robot.keyPress(KeyEvent.VK_A);
            robot.keyRelease(KeyEvent.VK_A);

            Thread.sleep(200);

            robot.keyPress(KeyEvent.VK_A);
            robot.keyRelease(KeyEvent.VK_A);

            int oldCol = tetris.getPieceCol();

            Thread.sleep(200);

            robot.keyPress(KeyEvent.VK_Q);
            robot.keyRelease(KeyEvent.VK_Q);


            Thread.sleep(200);

            robot.keyPress(KeyEvent.VK_Q);
            robot.keyRelease(KeyEvent.VK_Q);

            Thread.sleep(200);

            robot.keyPress(KeyEvent.VK_Q);
            robot.keyRelease(KeyEvent.VK_Q);

            assertTrue(oldCol != tetris.getPieceRow());
      }
```
Output:

Expected Output: True

Actual Output: True

Row is expected to be different because the row should be moved by passing the if statement on line 457 through the code executed on line 458. This test passed.

Test Path: Covers path [1,2,5,7,9,10,11] and is part of Edge Coverage for rotatePiece()

rotatePiece_blockedRight()

Test Case: [1,3], [3,4], [4,5], [5,7], [7,9] [9,10], [10,11]

Input:

```
@Test
	void rotatePiece_blockedRight() throws Exception {
		gameThread.start();

		Thread.sleep(200);


		robot.keyPress(KeyEvent.VK_ENTER);
		robot.keyRelease(KeyEvent.VK_ENTER);

		Thread.sleep(500);

		tetris.currentType = TileType.TypeI;

		Thread.sleep(200);

		robot.keyPress(KeyEvent.VK_Q);
		robot.keyRelease(KeyEvent.VK_Q);

		Thread.sleep(200);

		robot.keyPress(KeyEvent.VK_D);
		robot.keyRelease(KeyEvent.VK_D);

		Thread.sleep(200);

		robot.keyPress(KeyEvent.VK_D);
		robot.keyRelease(KeyEvent.VK_D);

		Thread.sleep(200);

		robot.keyPress(KeyEvent.VK_D);
		robot.keyRelease(KeyEvent.VK_D);

		Thread.sleep(200);

		robot.keyPress(KeyEvent.VK_D);
		robot.keyRelease(KeyEvent.VK_D);

		Thread.sleep(200);

		robot.keyPress(KeyEvent.VK_D);
		robot.keyRelease(KeyEvent.VK_D);
```

```
                Thread.sleep(200);

                int oldCol = tetris.getPieceCol();

                robot.keyPress(KeyEvent.VK_E);
                robot.keyRelease(KeyEvent.VK_E);

                Thread.sleep(200);

                robot.keyPress(KeyEvent.VK_E);
                robot.keyRelease(KeyEvent.VK_E);

                Thread.sleep(200);

                robot.keyPress(KeyEvent.VK_E);
                robot.keyRelease(KeyEvent.VK_E);

                assertTrue(oldCol != tetris.getPieceRow());
        }
```
Output:

Expected Output = True

Actual Output = True

Row is expected to be different because the row should be moved by passing the if statement on line 459 through the code executed on line 460. This test passed.

Test Path: Covers path [1,3,4,5,7,9,10,11] and is part of Edge Coverage for rotatePiece()

rotatePiece_blockedBottom()

Test Case: [1,3], [3,5], [5,6], [6,9], [9,10], [10,11]

Input:
```
@Test
        void rotatePiece_blockedBottom() throws Exception {
                gameThread.start();

                Thread.sleep(200);


                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(500);
                tetris.currentType = TileType.TypeI;

                robot.keyPress(KeyEvent.VK_S);
                Thread.sleep(600);
                robot.keyRelease(KeyEvent.VK_S);

                Thread.sleep(500);

                int oldRow = tetris.getPieceRow();

                robot.keyPress(KeyEvent.VK_Q);
                robot.keyRelease(KeyEvent.VK_Q);

                Thread.sleep(200);

                robot.keyPress(KeyEvent.VK_Q);
                robot.keyRelease(KeyEvent.VK_Q);

                Thread.sleep(200);

                robot.keyPress(KeyEvent.VK_Q);
                robot.keyRelease(KeyEvent.VK_Q);

                assertTrue(oldRow != tetris.getPieceRow());
        }
```
Output:

Expected Output = True

Actual Output = True

Row is expected to be different because the row should be moved by passing the if statement on line 469 through the code executed on line 470. This test passed.

Test Path: Covers path [1,3,5,6,9,10,11] and is part of Edge Coverage for rotatePiece()

rotatePiece_blockedTop()

Test Case: [1,3], [3,5], [5,7], [7,8], [9,10], [10,11]

Input:
```
@Test
    void rotatePiece_blockedTop() throws Exception {
        gameThread.start();

        Thread.sleep(200);
```

```
                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(200);

                tetris.currentType = TileType.TypeI;
                tetris.currentRow = -1;
                int oldRow = tetris.getPieceRow();
                robot.keyPress(KeyEvent.VK_Q);
                robot.keyRelease(KeyEvent.VK_Q);

                Thread.sleep(100);

                assertTrue(oldRow != tetris.getPieceRow());
        }
```

Output:

Expected Output = True

Actual Output = True

Row is expected to be different because the row should be moved by passing the if statement on line 467 through the code executed on line 468. This test passed.

Test Path: Covers path [1,3,5,7,8,9,10,11] and is part of Edge Coverage for rotatePiece()

rotatePiece_notValidandEmpty()

Test Case: [1,3], [3,5], [5,7], [7,8], [8,9], [9,10], [10,11]

Input:

```
@Test
        void rotatePiece_notValidandEmpty() throws Exception{
                gameThread.start();
                Thread.sleep(200);
                robot.keyPress(KeyEvent.VK_ENTER);
                robot.keyRelease(KeyEvent.VK_ENTER);

                Thread.sleep(100);

                for(int j = 0; j < 22; j++) {
                        for(int i = 0; i < 10; i++) {
                                tetris.board.setTile(i, j, TileType.TypeI);
                        }
                }

                Thread.sleep(100);

                robot.keyPress(KeyEvent.VK_Q);
                robot.keyRelease(KeyEvent.VK_Q);

                Thread.sleep(200);

                assertFalse(tetris.board.isValidAndEmpty(tetris.currentType, tetris.getPieceCol(),
tetris.getPieceRow(), tetris.getPieceRotation()));

        }
```
Output:

Expected Output: False

Actual Output: False

This case should be true due to the isValidAndEmpty command being necessary to get false of the if statement on line 477 being the bridge to the edge leading from node 9 to node 11. This test passed.

Test Path: Covers path [1,3,5,7,8,9,10,11] and is part of Edge Coverage for rotatePiece()

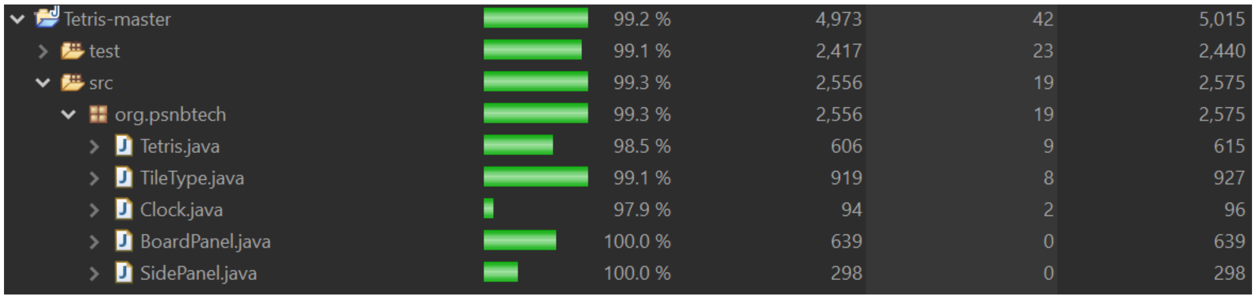
Problems Found:

Incorrect Score Bug: The score adjustment which takes place in updateGame() in the Tetris class on line 350 has "<< cleared" after "score += 50". This bug can be fixed by simply deleting the "<< cleared" leaving it just as "score += 50". This bug was discovered when testing updateGame() in the updateGame_Score test case which covers the path [1,3,4,5,6] in the CFG for that class. Instead of the score being 100 when a line was cleared the score was instead several million points which was a clear sign something was wrong in the code.

Check Line Bug: The line checking function in the BoardPanel class on line 225 has an issue where when it checks if a line is not occupied on line 231 it will return true instead of false. This bug can be fixed by changing true to false on line 232. This bug was discovered when testing updateGame() in the updateGame_NoScore test case which was supposed to cover the path [1,3,5,6] but instead covered the path [1,3,4,5,6] due to this bug. When a piece was dropped the score went up which was a clear sign something was wrong so for the test case a piece is dropped and the score is checked to see if it is still zero.

JaCoCo Score:

Instruction Counters Coverage:



Tetris-master	🟩	99.2 %	4,973	42	5,015
test	🟩	99.1 %	2,417	23	2,440
src	🟩	99.3 %	2,556	19	2,575
org.psnbtech	🟩	99.3 %	2,556	19	2,575
Tetris.java	🟩	98.5 %	606	9	615
TileType.java	🟩	99.1 %	919	8	927
Clock.java	🟩	97.9 %	94	2	96
BoardPanel.java	🟩	100.0 %	639	0	639
SidePanel.java	🟩	100.0 %	298	0	298

The JaCoCo coverage score is 99.3%. The score started around 80-90% when JaCoCo was first installed. It was very helpful how JaCoCo highlights missed lines of code in red. Without that feature this assignment would have been much more difficult. This score was improved by changing coverage from node coverage to edge coverage to prime path coverage depending on which line of code needed to be reached.

Reflection and Lessons Learned:

When we initially played the game we noticed that the score seemed to be incorrect and the coverage criteria hindered us because we couldn't directly test the score. We needed to consider the path we were testing under which coverage criteria and how the code was calculating the score. Considering how we could test the score helped influence the methods we wanted to test and under which criteria. There are many private methods that are also called in other larger methods. We found that testing functions in the Tetris class allowed us to test many supporting methods in other classes. We ran into some problems when it came to determining what coverage criteria would be the best for the methods we tested. When considering prime path coverage there were sometimes many unfeasible paths so it made more sense to use node or edge coverage. Additionally, we struggled in determining which asserts made the most sense (especially when the methods were of the void type) and verified the path we were testing.Choosing assert methods could sometimes be difficult due to multiple conditions in the if statements. We tried to avoid changing private methods or members to public as much as possible but in order to achieve our coverage criterias it was sometimes necessary.

However, although we ran into many problems we learned many lessons. It could sometimes be frustrating to test the code but our coverage criterias and CFGs helped us create a guideline on how to test the program. Initially when we viewed the code it looked very intimidating especially since it was a GUI based program. After we better understood the code and created our CFGs we were able to break up the tests into smaller steps. We also felt more confident in the test methods we created when we saw our Jacoco scores and it helped motivate us to achieve higher coverage scores. Additionally, we learned how to use java robot which came with its own trials but it overall was a very useful tool in testing this program. We also learned how to use different assert methods and often discussed which method would best suit each testing method. Overall, using JUnit and the coverage criterias we learned in class came with their own learning curves but were helpful in ensuring that we covered as much code as possible in the program.