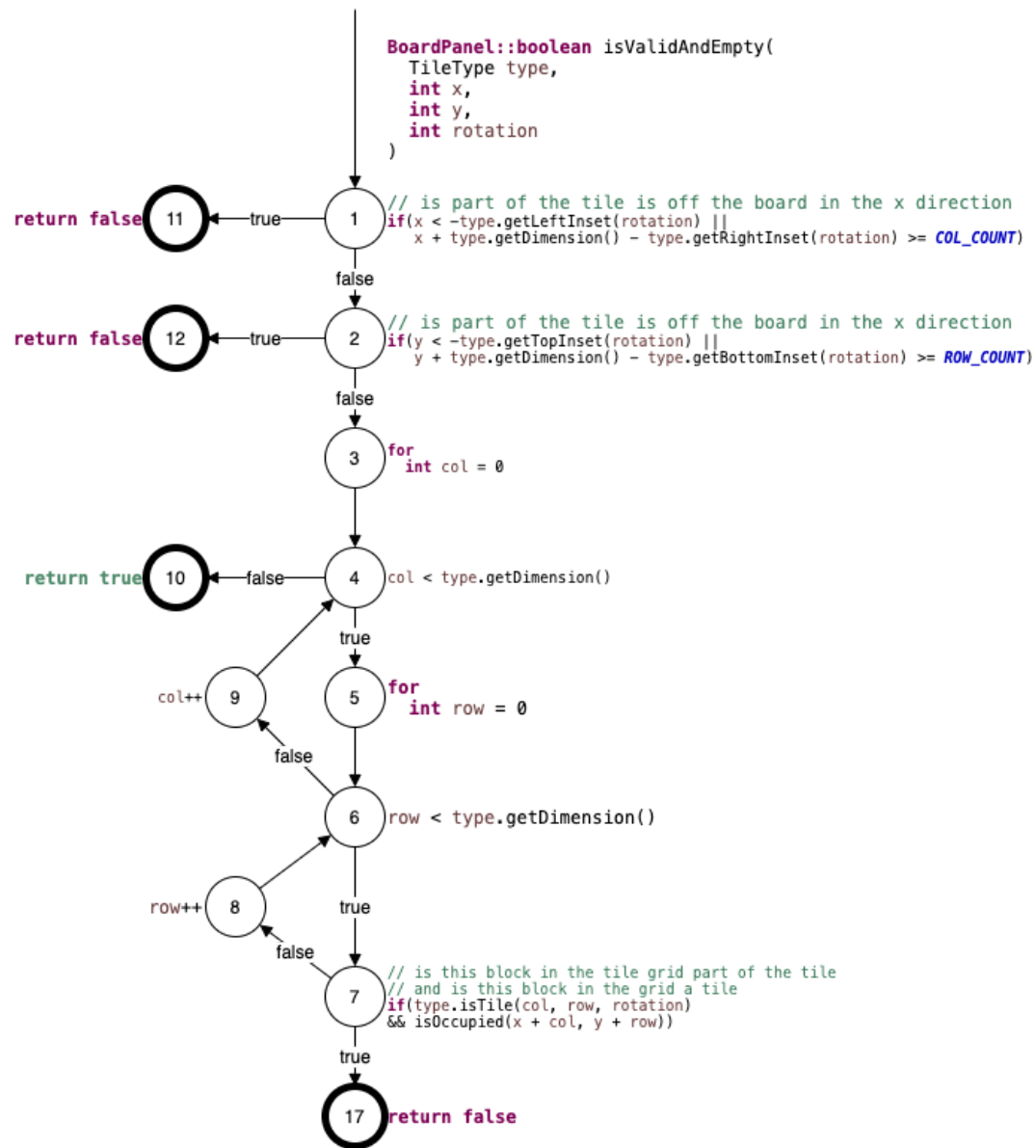


SE 461 Software Testing and Coverage
JUnit and Test Coverage Criteria
By: Lauren Gonzalez, Sirena Murphree, Paul Nguyen

Test Design



BoardPanel::Boolean isValidAndEmpty(tt, x, y, r)

Complete Prime Paths

1. 1, 11
2. 1, 2, 12
3. 1, 2, 3, 4, 10
4. 1, 2, 3, 4, 5, 6, 7, 17

Incomplete Prime Paths:

5. 1, 2, 3, 4, 5, 6, 9
6. 1, 2, 3, 4, 5, 6, 7, 8
7. 4, 5, 6, 9, 4
8. 5, 6, 9, 4, 5
9. 5, 6, 9, 4, 10
10. 6, 7, 8, 6
11. 6, 9, 4, 5, 6
12. 7, 8, 6, 7
13. 7, 8, 6, 9, 4, 5
14. 7, 8, 6, 9, 4, 10
15. 8, 6, 7, 8
16. 8, 6, 7, 17
17. 9, 4, 5, 6, 9
18. 9, 4, 5, 6, 7, 8
19. 9, 4, 5, 6, 7, 17

Path: 5, 6, 9

Not Feasible:

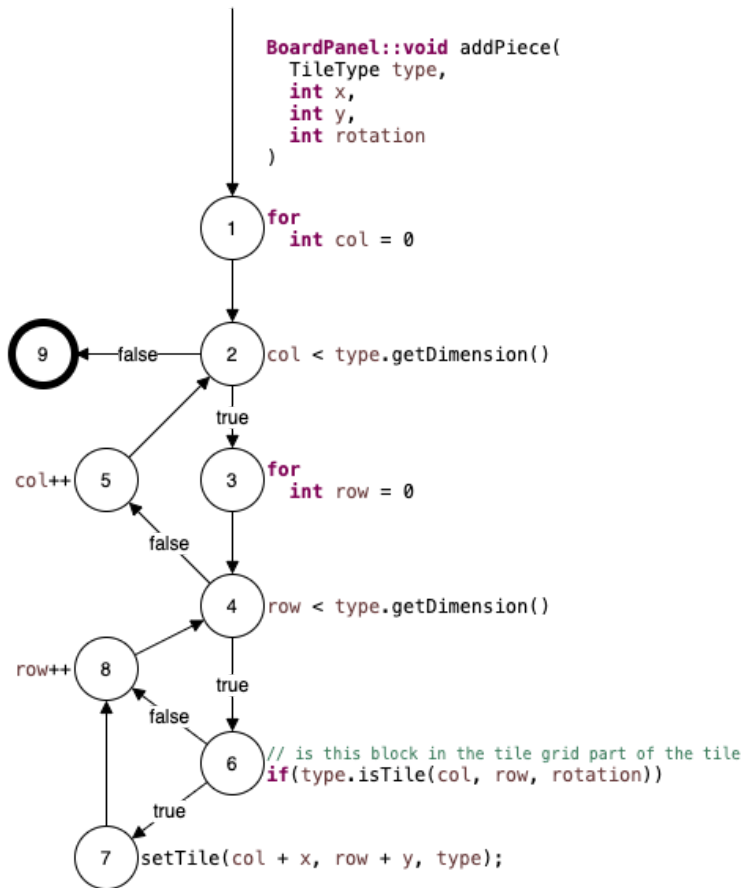
type.getDimensions() will return a value >0; this loop will always run at least 1 time.

Path: 3, 4, 10

Not Feasible:

type.getDimensions() will return a value >0; this loop will always run at least 1 time.

TileType	x	y	r	Internal State	Output	Logic	Path Coverage {path covered} Test Path
TypeI	10	10	0	Not relevant	false	Off the board in the x direction	{1} 1, 11
TypeI	2	21	1	Not relevant	false	Off the board in the y direction	{2} 1, 2, 12
Type0	5	5	0	No tiles at [5][5], [5][6] [6][5], [6][6]	true	[][] [][] No block overlap	{6, 10, 11, 12, 13, 14, 15, 18} 1, 2, 3, 4, 5, 6, 7, 8, 6, 7, 8, 6, 9, 4, 5, 6, 7, 8, 6, 7, 8, 6, 9, 4, 10
Type0	5	5	0	Tile at position [5][5]	false	Top left block overlaps with another block	{4} 1, 2, 3, 4, 5, 6, 7, 17
TypeS	3	5	0	Tile at position [5][5]	false	[][][x] Top right block overlaps with another block	{6, 10, 12, 16} 1, 2, 3, 4, 5, 6, 7, 8, 6, 7, 8, 6, 7, 17
TaypL	5	4	2	Tile at position [5][5]	false	[][][] [x] Block on [x][y+n] overlaps with another block	{6, 12, 15, 19} 1, 2, 3, 4, 5, 6, 7, 8, 6, 7, 8, 6, 7, 8, 6, 9, 4, 5, 6, 7, 17



```
BoardPanel::void addPiece (tt, x, y, r)
```

Complete Prime Paths

1. 1, 2, 9

Incomplete Prime Paths:

2. 1, 2, 3, 4, 5

3. 1, 2, 3, 4, 6, 8

4. 1, 2, 3, 4, 6, 7, 8

5. 2, 3, 4, 5, 2

6. 3, 4, 5, 2, 3

7. 3, 4, 5, 2, 9

8. 4, 6, 8, 4

9. 4, 5, 2, 3, 4

10. 4, 6, 7, 8, 4

11. 5, 2, 3, 4, 5

12. 5, 2, 3, 4, 6, 7, 8

13. 6, 7, 8, 4, 6

14. 6, 8, 4, 6

15. 6, 7, 8, 4, 5, 2, 3

16. 6, 7, 8, 4, 5, 2, 9

17. 6, 8, 4, 5, 2, 3

18. 6, 8, 4, 5, 2, 9

19. 7, 8, 4, 6, 7

20. 8, 4, 6, 8

21. 8, 4, 6, 7, 8

Path: 1, 2, 9

Path: 3, 4, 5

Not Feasible:

type.getDimensions() will return a value >0; this loop will always run at least 1 time.

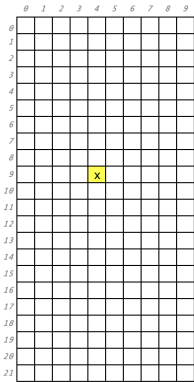
TileType	x	y	r	Logic	Path Coverage {Prime Paths covered} Test Path
Type0	3	5	0	[x][x] [x][x]	{4, 9, 10, 12, 13, 15, 16, 19, 21} 1, 2, 3, 4, 6, 7, 8, 4, 6, 7, 8, 4, 5, 2, 3, 4, 6, 7, 8, 4, 6, 7, 8, 4, 5, 2, 9
TypeI	0	8	2	[][][][] [][][][] [x][x][x][x] [][][][]	{3, 8, 9, 10, 13, 14, 17, 18, 20, 21} 1, 2, 3, 4, 6, 8, 4, 6, 8, 4, 6, 7, 8, 4, 6, 8, 4, 5, 2, 3, 4, 6, 8, 4, 6, 8, 4, 6, 7, 8, 4, 6, 8, 4, 5, 2, 3, 4, 6, 8, 4, 6, 8, 4, 6, 7, 8, 4, 6, 8, 4, 5, 2, 3, 4, 6, 8, 4, 6, 8, 4, 6, 7, 8, 4, 6, 8, 4, 5, 2, 9

BoardPanel::isOccupied

Characteristic 1: X [x, x']
Partitions: X is in a column that contains a tile
X is in a column that does not contain a tile

Characteristic 2: Y [y, y']
Partitions: Y is in a row that contains a tile
Y is in a row that does not contain a tile

$TR(ACoC) = \{(x', y'), (x', y), (x, y'), (x, y)\}$

Internal State	X	Y	Expected Output
	2	15	false
	2	9	fasle
	4	15	false
	4	9	true

BoardPanel::Boolean checkLine(line)

Characteristic 1: Expected Return

1 - row full

0 - At least 1 column is empty

Characteristic 2: Line Placement

f - first row

m - middle row

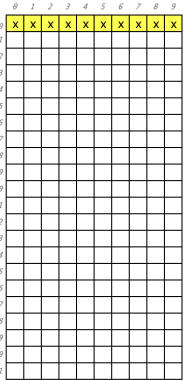
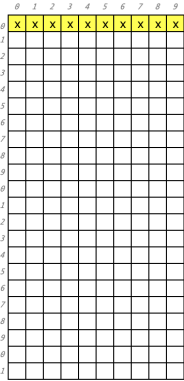
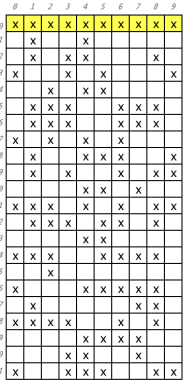
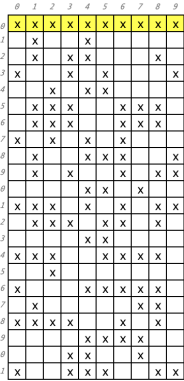
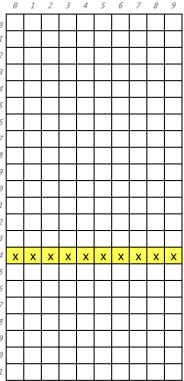
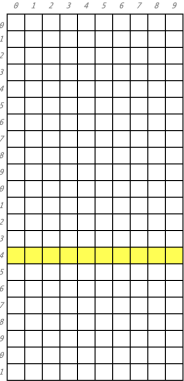
l - last row

Characteristic 3: Board State

e - Board is empty

a - Board has tiles

TR(ACoC) = {(1, f, e), (1, f, a), (1, m, e), (1, m, a), (1, l, e), (1, l, a), (0, f, e), (0, f, a), (0, m, e), (0, m, a), (0, l, e), (0, l, a)}

Internal State	Line	Expected Output	Expected Resulting State	Test Set
	0	True, No Change		1, f, e
	0	True, No Change		1, f, a
	14	True, Change		1, m, e

	14	True, Change		1, m, a
	21	True, Change		1, l, e
	21	True, Change		1. l, a
	0	False, No Change		0, f, e

	<p>0</p> <p>False, No Change</p>		<p>0, f, a</p>
	<p>14</p> <p>False, No Change</p>		<p>0, m, e</p>
	<p>14</p> <p>False, No Change</p>		<p>0, m, a</p>
	<p>21</p> <p>False, No Change</p>		<p>0, l, e</p>

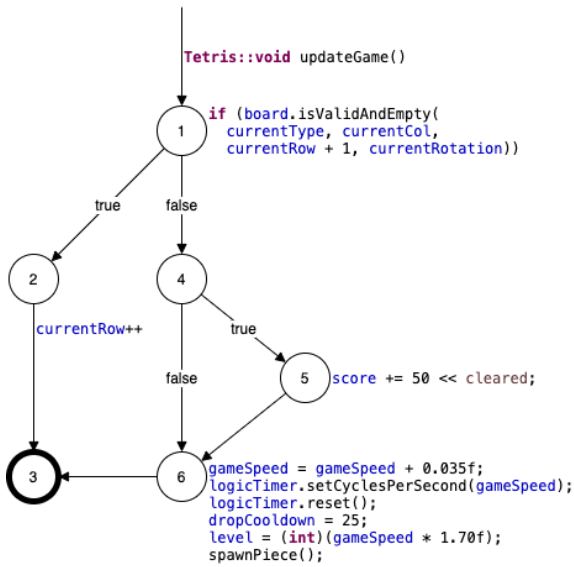
	0	1	2	3	4	5	6	7	8	9
0			x		x	x				x
1		x				x				
2		x		x	x					x
3	x			x			x			x
4		x		x	x					
5		x	x	x				x	x	x
6		x	x	x				x	x	x
7	x		x		x					
8	x			x	x	x				x
9		x		x			x	x	x	
10					x	x				
11	x	x	x		x		x		x	x
12		x	x	x		x	x			x
13					x					
14	x	x	x				x	x	x	x
15				x						
16	x				x	x	x	x	x	
17		x						x	x	
18	x	x	x	x				x		x
19					x	x	x	x		
20					x	x			x	
21	x	x			x	x	x	x	x	x

21

False, No Change

	0	1	2	3	4	5	6	7	8	9
0		x		x	x					x
1		x			x					
2		x		x	x					x
3	x			x		x				x
4		x		x	x					
5		x	x	x				x	x	x
6		x	x	x				x	x	x
7	x		x		x					
8	x			x	x	x				x
9		x		x			x	x	x	
10					x	x				
11	x	x	x		x		x		x	x
12		x	x	x		x	x			x
13					x	x				
14	x	x	x				x	x	x	x
15				x						
16	x				x	x	x	x	x	
17		x						x	x	
18	x	x	x	x				x		x
19					x	x	x	x		
20					x	x			x	
21	x	x			x	x	x	x	x	x

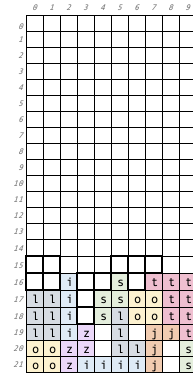
0, 1, a



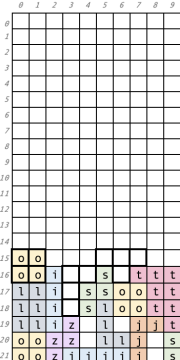
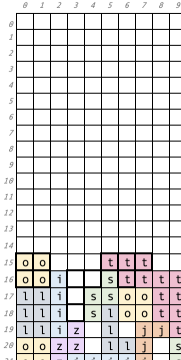
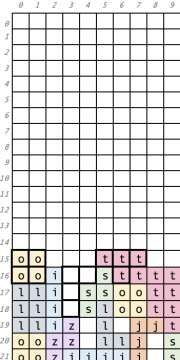
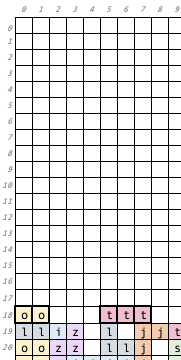
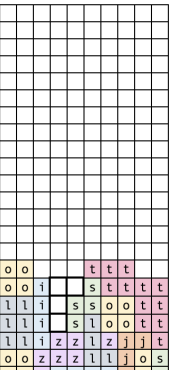
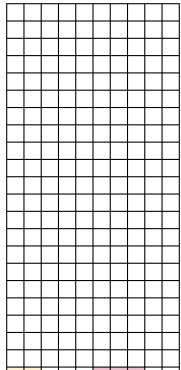
Tetris::update()

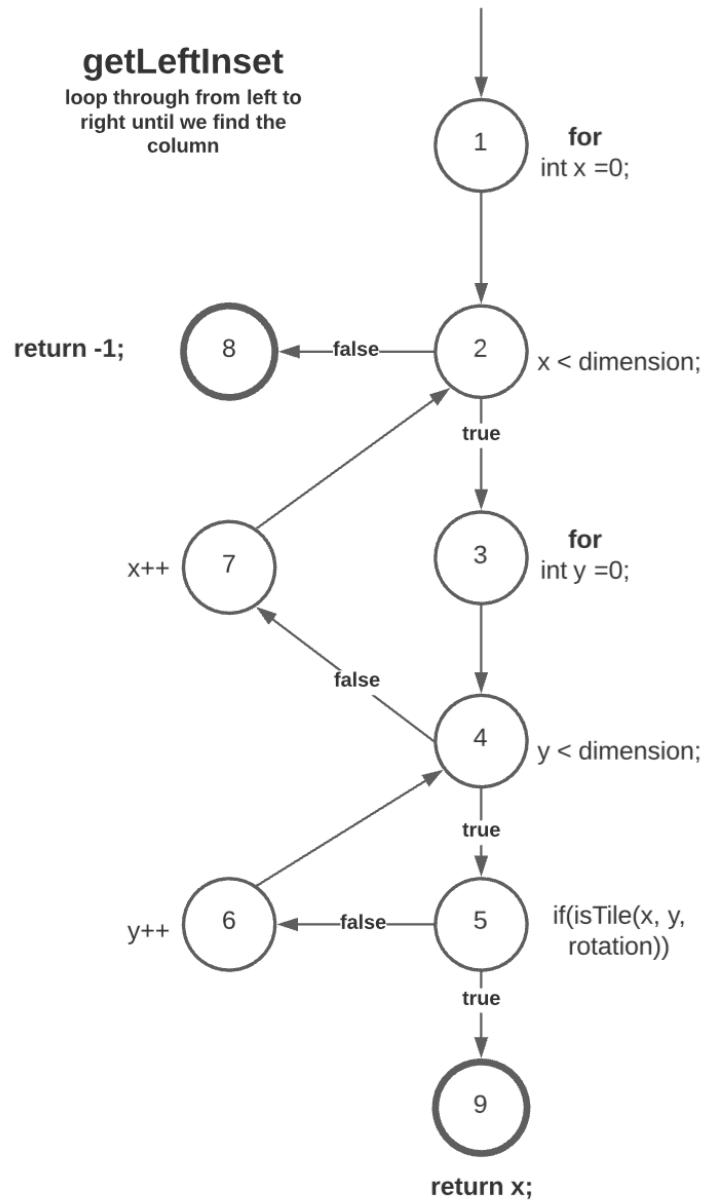
Prime Paths:

- 1, 2, 3
- 1, 4, 5, 6, 3
- 1, 4, 6, 3



Internal State board	Row Column Rotation TileType	Expected Result currentRow gameSpeed score dropCooldown level	Internal State Board	Path
	R: <15 C: Any Rot: Any T: 0	gameSpeed - no change Score - no change dropCooldown - no change Level - no change		1, 2, 3
	R: 15 C: 0 Rot: 2 T: 0	gameSpeed - +0.035 Score - no change dropCooldown - 25 Level - gs*1.70		1, 4, 6, 3

	<p>R: 14 C: 5 Rot: 2 T: T</p>	<p>gameSpeed - +0.035 Score - no change dropCooldown - 25 Level - gs*1.70</p>		<p>1, 4, 6, 3</p>
	<p>R: 16 C: 2 Rot: 1 T: J</p>	<p>gameSpeed - +0.035 Score - +400 dropCooldown - 25 Level - gs*1.70</p>		<p>1, 4, 5, 6, 3</p>
	<p>R: 16 C: 2 Rot: 1 T: J</p>	<p>Score change by +50*2^clear</p>		<p>1, 4, 5, 6, 3</p>



TR's: {1,2,3,4,5,6,7,8,9}

Prime paths:

1. 4, 5, 6, 4
2. 2, 3, 4, 7, 2
3. 3, 4, 5, 6, 4
4. 4, 5, 6, 4, 7
5. 1, 2, 3, 4, 7, 2
6. 2, 3, 4, 5, 6, 4
7. 3, 4, 5, 6, 4, 7
8. 4, 5, 6, 4, 7, 2
9. 6, 4, 7, 2, 3, 4
10. 1, 2, 3, 4, 5, 6, 4
11. 2, 3, 4, 5, 6, 4, 7
12. 1, 2, 3, 4, 5, 6, 4, 7, 2

Test Cases:

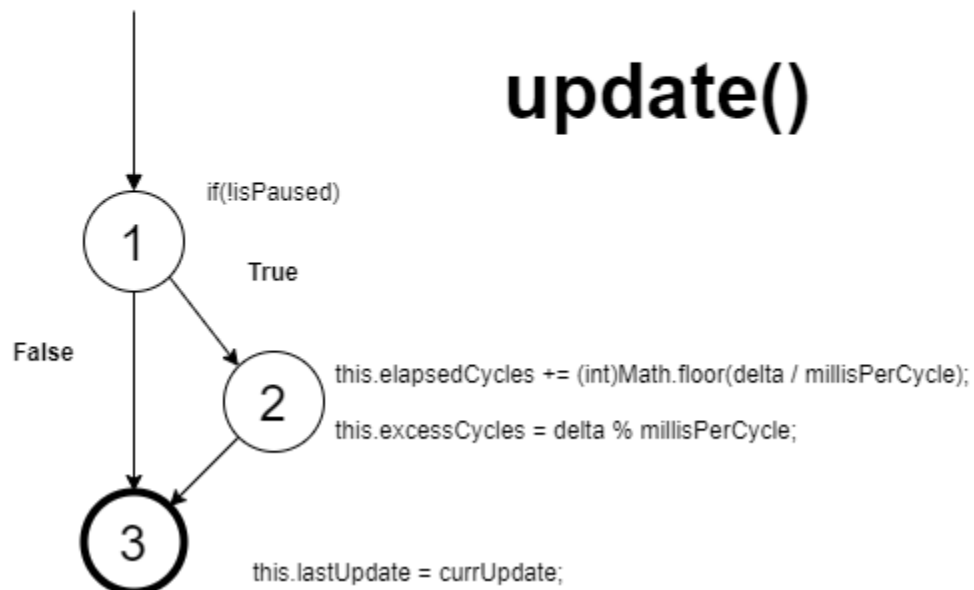
For tile type I,

1. Input: 0, Expected: 0
2. Input: 1, Expected: 2
3. Input: 2, Expected: 0
4. Input: 3, Expected: 3

The inputs can only test result values of 0-3 because the dimension cannot exceed those values.

```
long currUpdate = getCurrentTime();
```

```
float delta = (float)(currUpdate - lastUpdate) + excessCycles;
```



Prime path coverage

1. [1, 2, 3]
2. [1, 3]

Testing cases:

Input: 0; expected: 0

Test Results and Analysis:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Tetris	99.5 %	18,104	87	18,191
TestTetris	99.6 %	15,523	69	15,592
src	99.3 %	2,581	18	2,599
org.psnbtech	99.3 %	2,581	18	2,599
TileType.java	99.1 %	919	8	927
TileType	99.1 %	919	8	927
isTile(int, int, int)	100.0 %	12	0	12
getTopInset(int)	92.3 %	24	2	26
getSpawnRow()	100.0 %	3	0	3
getSpawnColumn()	100.0 %	3	0	3
getRows()	100.0 %	3	0	3
getRightInset(int)	93.3 %	28	2	30
getLightColor()	100.0 %	3	0	3
getLeftInset(int)	92.3 %	24	2	26
getDimension()	100.0 %	3	0	3
getDarkColor()	100.0 %	3	0	3
getCols()	100.0 %	3	0	3
getBottomInset(int)	93.3 %	28	2	30
getBaseColor()	100.0 %	3	0	3
Tetris.java	98.4 %	612	10	622
Tetris	97.8 %	435	10	445
updateGame()	100.0 %	73	0	73
startGame()	95.5 %	64	3	67
spawnPiece()	100.0 %	45	0	45
rotatePiece(int)	100.0 %	121	0	121
resetGame()	100.0 %	37	0	37
renderGame()	100.0 %	7	0	7
isPaused()	100.0 %	3	0	3
isNewGame()	100.0 %	3	0	3
isGameOver()	100.0 %	3	0	3
getScore()	100.0 %	3	0	3
getPieceType()	100.0 %	3	0	3
getPieceRow()	100.0 %	3	0	3
getPieceRotation()	100.0 %	3	0	3
getPieceCol()	100.0 %	3	0	3
getNextPieceType()	100.0 %	3	0	3
getLevel()	100.0 %	3	0	3
Tetris()	100.0 %	54	0	54
new KeyAdapter() { ... }	100.0 %	177	0	177
keyReleased(KeyEvent)	100.0 %	15	0	15
keyPressed(KeyEvent)	100.0 %	156	0	156
main(String[])	0.0 %	0	7	7
SidePanel.java	100.0 %	308	0	308
SidePanel	100.0 %	308	0	308
paintComponent(Graphics)	100.0 %	192	0	192
drawTile(TileType, int, int, Graphics)	100.0 %	78	0	78
SidePanel(Tetris)	100.0 %	16	0	16
Clock.java	100.0 %	96	0	96
Clock	100.0 %	96	0	96
update()	100.0 %	36	0	36
setPaused(boolean)	100.0 %	4	0	4
setCyclesPerSecond(float)	100.0 %	8	0	8
reset()	100.0 %	13	0	13
peekElapsedCycle()	100.0 %	7	0	7
isPaused()	100.0 %	3	0	3
hasElapsedCycle()	100.0 %	13	0	13
Clock(float)	100.0 %	8	0	8
getCurrentTime()	100.0 %	4	0	4
BoardPanel.java	100.0 %	646	0	646
BoardPanel	100.0 %	646	0	646
setTile(int, int, TileType)	100.0 %	8	0	8
paintComponent(Graphics)	100.0 %	315	0	315
isValidAndEmpty(TileType, int, int, int)	100.0 %	71	0	71
isOccupied(int, int)	100.0 %	11	0	11
getTile(int, int)	100.0 %	7	0	7
drawTile(TileType, int, int, Graphics)	100.0 %	12	0	12
drawTile(Color, Color, Color, int, int, Graphics)	100.0 %	75	0	75
clear()	100.0 %	22	0	22
checkLines()	100.0 %	16	0	16
checkLine(int)	100.0 %	41	0	41
addPiece(TileType, int, int, int)	100.0 %	32	0	32
BoardPanel(Tetris)	100.0 %	21	0	21

JaCoCo overall coverage score of 99.3%

The bug found in the code was in `BoardPanel::Boolean checkLine()`. This error was determined while testing `BoardPanel::int checkLines`, and confirmed while creating test cases for the `checkLine` function.

The more difficult test revolved around testing AWT elements. Test score was improved by using a Java Robot to simulate player interaction and verify that the internal state updated appropriately during the normal course of game play.

To test the visual parts of the game we require users to look at a frame and enter basic confirmation messages to verify that what they are seeing is representative of what the internal state reflects. `Paint Tile` is a function that runs as part of the paint component so we opted not to test it on its own but instead evaluate the paint component functions as a whole.

Reflection and Lessons Learned:

Coverage criteria helps by determining what are useful test cases to run and how to logic out some expected results so that assertions made can be examined.

Lauren: The JUnit test results were helpful to see possible faults in the code. Within the most of the classes were testing getters because they were simple test cases that although seem unnecessary to test are actually helpful to see if the code altogether can pass needed values. What worked was viewing the code as a whole and see where there are commonalities. For example, in the tile type class the nested for loops were similar within their graphs so it was helpful to reuse one graph and rename labels to test. I think testing was very helpful to thoroughly understand your own code and figure out where the errors are happening if you do not understand how or where they are.

Sirena: One of the more difficult parts of creating tests was for interdependent class objects that are tightly coupled. Running some tests on classes that are not fully instantiated caused null errors to be thrown by classes that the class under test is dependent on, highlighting the importance of having a complete understanding of a program before testing it. One particularly pesky error in BoardPanel; it would try to render in cases where Tetris::currentTile was not set, throwing errors that were unrelated to the current method under evaluation. A temporary fix was to put the whole area that throws the error in a "if not null" so as to not cause distraction. When trying to run a test suite the Java Robot and the Board and Side Panel test caused conflicting errors that are not a problem when tested independently.

Paul: Testing multiple variables to determine if the same outcome occurs or not. Working on multiple testing designs gave a base on where to start. Learning multiple path coverages for different test cases.