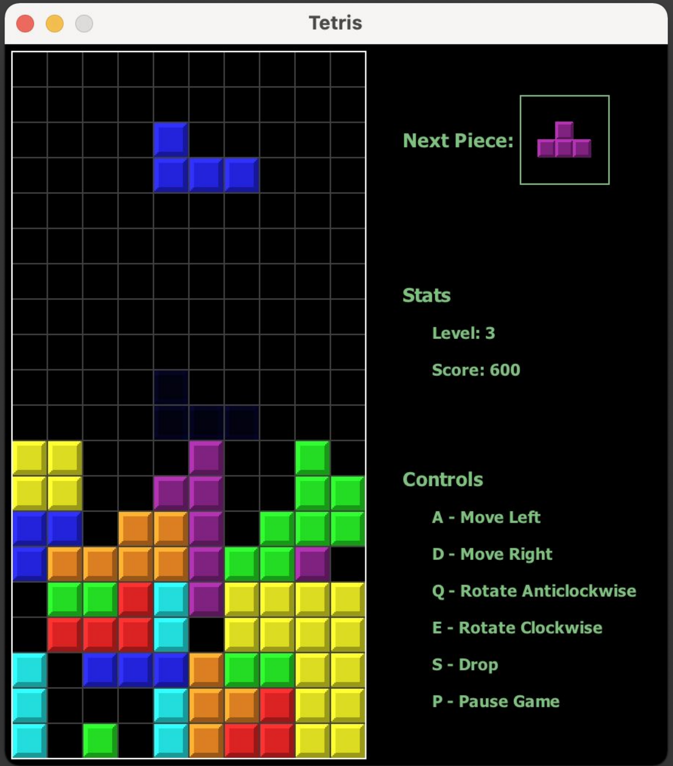
Tetris JUnit Testing

Lauren Gonzalez | Sirena Murphree | Paul Nguyen

Testing Design

- Graph Based Coverage
  - Prime Path Coverage
- Input Space Partitioning
  - All Combination Coverage ACoC
- Test Results
  - Bugs
- JaCoCo
  - Code Coverage



Tetris

Next Piece:

Stats

Level: 3

Score: 600

Controls

A - Move Left

D - Move Right

Q - Rotate Anticlockwise

E - Rotate Clockwise

S - Drop

P - Pause Game

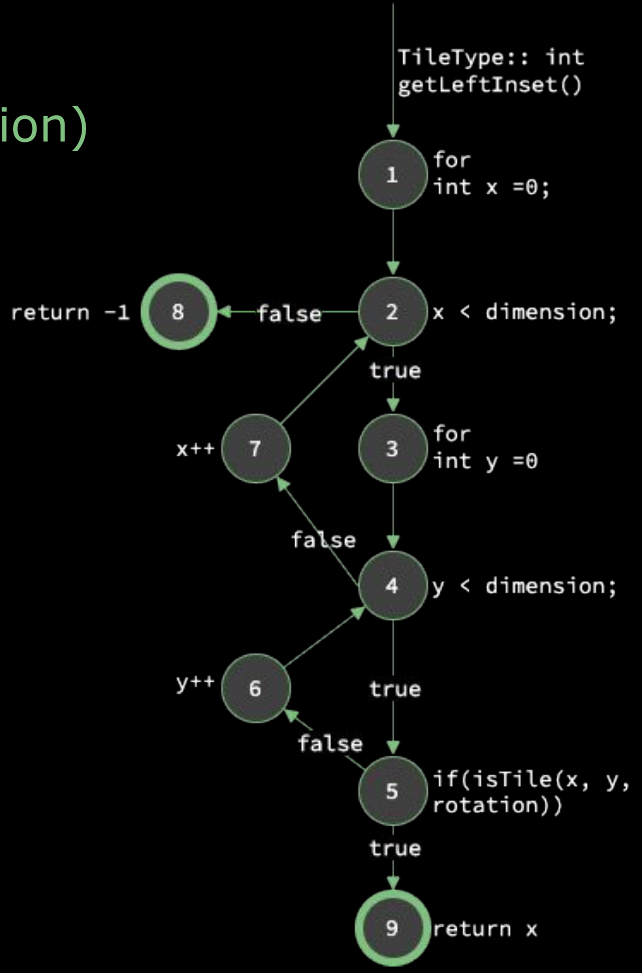
TileType::int getLeftInset(int rotation)

Prime Path Coverage:

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 Results

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



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

Prime Path Coverage

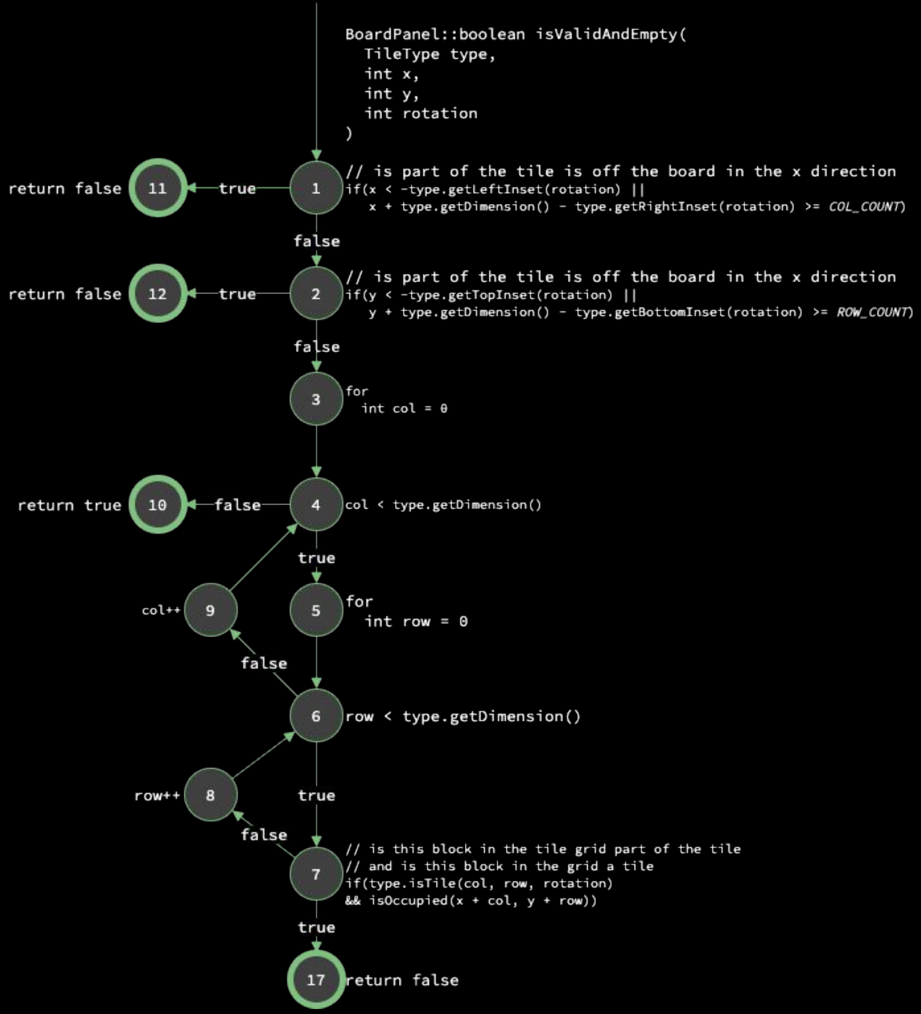
Complete Prime Paths

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

Not Feasible: 3, 4, 10 and 5, 6, 9



```
BoardPanel::boolean isValidAndEmpty(
  TileType type,
  int x,
  int y,
  int rotation
)
```

```
// is part of the tile is off the board in the x direction
if(x < -type.getLeftInset(rotation) ||
   x + type.getDimension() - type.getRightInset(rotation) >= COL_COUNT)
```

```
// is part of the tile is off the board in the x direction
if(y < -type.getTopInset(rotation) ||
   y + type.getDimension() - type.getBottomInset(rotation) >= ROW_COUNT)
```

```
for
  int col = 0
```

```
col < type.getDimension()
```

```
for
  int row = 0
```

```
row < type.getDimension()
```

```
// is this block in the tile grid part of the tile
// and is this block in the grid a tile
if(type.isTile(col, row, rotation)
&& isOccupied(x + col, y + row))
```

BoardPanel:: Boolean isValidAndEmpty(TileType, x, y, r) (continued)

TileType	x	y	r	Internal State	Expected Output	Prime Paths Covered Test Path
TypeI	10	10	0	Not relevant	false	{1} 1, 11
TypeI	2	21	1	Not relevant	false	{2} 1, 2, 12
TypeO	5	5	0	No tiles at [5][5], [5][6] [6][5], [6][6]	true	{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
TypeO	5	5	0	Tile at position [5][5]	false	{4} 1, 2, 3, 4, 5, 6, 7, 17
TypeS	3	5	0	Tile at position [5][5]	false	{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	{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

Bug

```
/**
 * Checks the board to see if any lines have been cleared, and
 * removes them from the game.
 * @return The number of lines that were cleared.
 */
public int checkLines() {
        int completedLines = 0;
        /*
         * Here we loop through every line and check it to see if
         * it's been cleared or not. If it has, we increment the
         * number of completed lines and check the next row.
         * The checkLine function handles clearing the line and
         * shifting the rest of the board down for us.
         */
        for(int row = 0; row < ROW_COUNT; row++) {
                if(checkLine(row)) {
                        completedLines++;
                }
        }
        return completedLines;
}
```

```
/**
 * Checks whether or not {@code row} is full.
 * @param line The row to check.
 * @return Whether or not this row is full.
 */
public boolean checkLine(int line) {
        /*
         * Iterate through every column in this row. If any of them are
         * empty, then the row is not full.
         */
        for(int col = 0; col < COL_COUNT; col++) {
                if(!isOccupied(col, line)) {
                        return true; // << BUG!! should return FALSE
                }
        }

        /*
         * Since the line is filled, we need to 'remove' it from the game.
         * To do this, we simply shift every row above it down by one.
         */
        for(int row = line - 1; row >= 0; row--) {
                for(int col = 0; col < COL_COUNT; col++) {
                        setTile(col, row + 1, getTile(col, row));
                }
        }
        return true;
}
```

Bug BoardPanel :: Boolean checkLine(int)

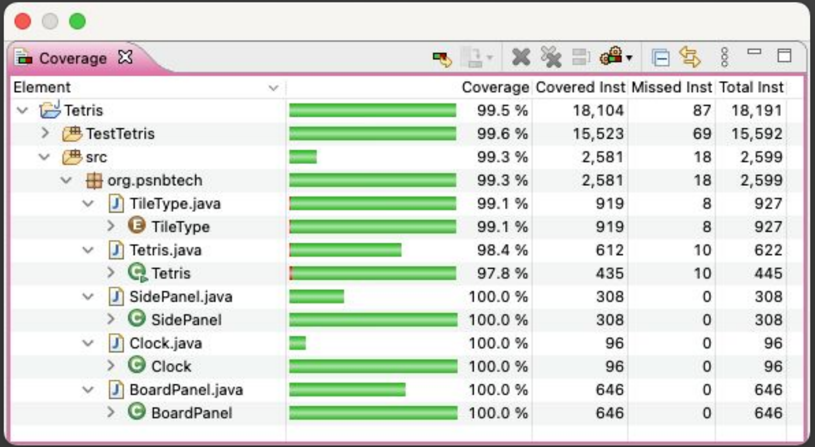
Input Space Partitioning & ACoC

1. Expected Return
   - 1 – Row full
   - 0 – At least 1 column is empty
2. Line Placement
   - f – first row
   - m – middle row
   - l – last row
3. Board State
   - e – Board is empty
   - a – Board has tiles

Combination	Return Value	Internal State
(1, f, e)	true	No Change
(1, f, a)	true	No Change
(1, m, e)	true	Change
(1, m, a)	true	Change
(1, l, e)	true	Change
(1, l, a)	true	Change
(0, f, e)	false	No Change
(0, f, a)	false	No Change
(0, m, e)	false	No Change
(0, m, a)	false	No Change
(0, l, e)	false	No Change
(0, l, a)	false	No Change

JaCoCo

99.3%



Element	Coverage	Covered Inst	Missed Inst	Total Inst
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
Tetris.java	98.4 %	612	10	622
Tetris	97.8 %	435	10	445
SidePanel.java	100.0 %	308	0	308
SidePanel	100.0 %	308	0	308
Clock.java	100.0 %	96	0	96
Clock	100.0 %	96	0	96
BoardPanel.java	100.0 %	646	0	646
BoardPanel	100.0 %	646	0	646