International Institute of Information Technology, Bangalore
(IIIT Bangalore)

Software Testing CS 731

Project Report

Jayaa Shree Laxmi Kishoore
(MT2022053)

Shashidhar Basavaraj Nagaral

(MT2022108)

Dataflow Graph based Testing

Data-flow graph (DFG) is a graph that represents control flow of a function in which every node is labeled with definitions(def) and uses(use) of variables in that basic block.

In this test paths which are DU-paths are generated for each variable which which covers both a def and use of that variable.So basically in this method we use test paths of a program according to the locations of definitions and uses of variables in the program

It is concerned with:
- Statements where variables receive values,
- Statements where these values are used or referenced.

DEF(S) = {X | statement S contains the definition of X} USE(S) = {X | statement S contains the use of X}
Then, we write Test Cases for every unique DU Path.

Github Link

Jayaa Shree Laxmi :
https://github.com/jayashree-1998/Software-Testing-DFG

Shashidhar Nagaral:
https://github.com/ShashidharNagaral/Software-Testing-DFG

Tools Used for Testing

- **Data Flow Graph Coverage Web Application:**
(https://cs.gmu.edu:8443/offutt/coverage/DFGraphCoverage)
We used this web tool to generate All Def Coverage and All DU path Coverage for our Data Flow Graph of each function

- **JUnit:** (http://junit.org/junit5/)
It is a unit testing tool for java based applications, used for automating the execution of the Test Cases.

About our Project Code:

The codebase we have used in for a command-line chess application written in Java.
Chess game files:

1. Bishop.java
2. ChessBoard.java
3. ChessGame.java
4. ChessPiece.java
5. ChessUtil.java
6. COLOR.java
7. King.java
8. Knight.java
9. Main.java
10. Pawn.java
11. Queen.java
12. Rook.java

Description

The ChessTest.java contains the test cases for all the functions we have tested. It contains functions with nested ifs and conditional loops. We have tested the methods for the piece movement and also checked for the game conditions like CHECKMATE, STALEMATE.

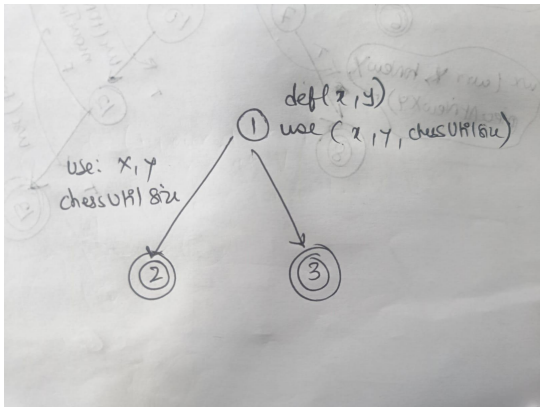
We have also tested the movement methods of the pieces like straight movement and diagonal movements.

We have mentioned **All DU Path Coverage** along with **All Def Coverage**. Since All-Def is subsumed by All-DU we have considered **All DU Path Coverage** to create test cases.

Functions:

1. ChessBoard.java – checkBounds()

Data Flow Graph:



Test Requirement:

DU Paths for all variables are:

Variable	DU Paths
x	[1,3] [1,2]
y	[1,2] [1,3]
ChessUtilSIZE	No path or No path needed

Test Paths:

All Def Coverage for all variables are:

Variable	All Def Coverage
x	[1,3]
y	[1,2]
ChessUtilSIZE	No path or No path needed

All DU Path Coverage for all variables are:

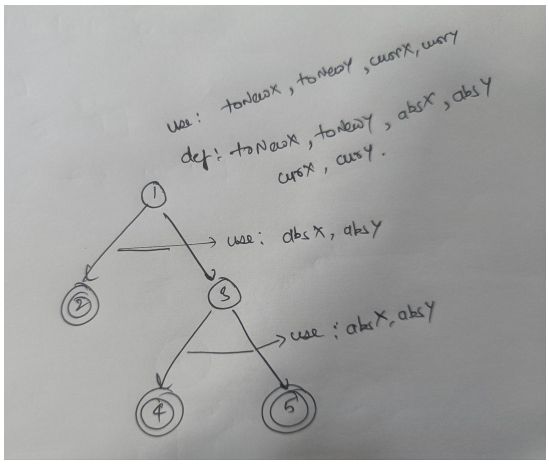
Variable	All DU Path Coverage
x	[1,3] [1,2]
y	[1,2] [1,3]
ChessUtilSIZE	No path or No path needed

JUnit Test Case:

```
@Test
public void boundCheckTest() {
    // Test Path: [1, 2] : within bound
    assertEquals( expected: true, chessBoard.boundCheck( x: 6,  y: 7));

    // Test Path: [1, 3] : out of bound
    assertEquals( expected: false, chessBoard.boundCheck( x: 8,  y: 7));
}
```

2. Knight.java: checkKnightMovement()

Data Flow Graph:



Test Requirement:

DU Paths for all variables are:

Variable	DU Paths
toNewX	No path or No path needed
toNewY	No path or No path needed
absX	[1,3] [1,2] [1,3,4] [1,3,5]
absY	[1,3] [1,2] [1,3,4] [1,3,5]
currX	No path or No path needed
currY	No path or No path needed

Test Paths:

All Def Coverage for all variables are:

Variable	All Def Coverage
toNewX	No path or No path needed
toNewY	No path or No path needed
absX	[1,3,4]
absY	[1,3,4]
currX	No path or No path needed
currY	No path or No path needed

All DU Path Coverage for all variables are:

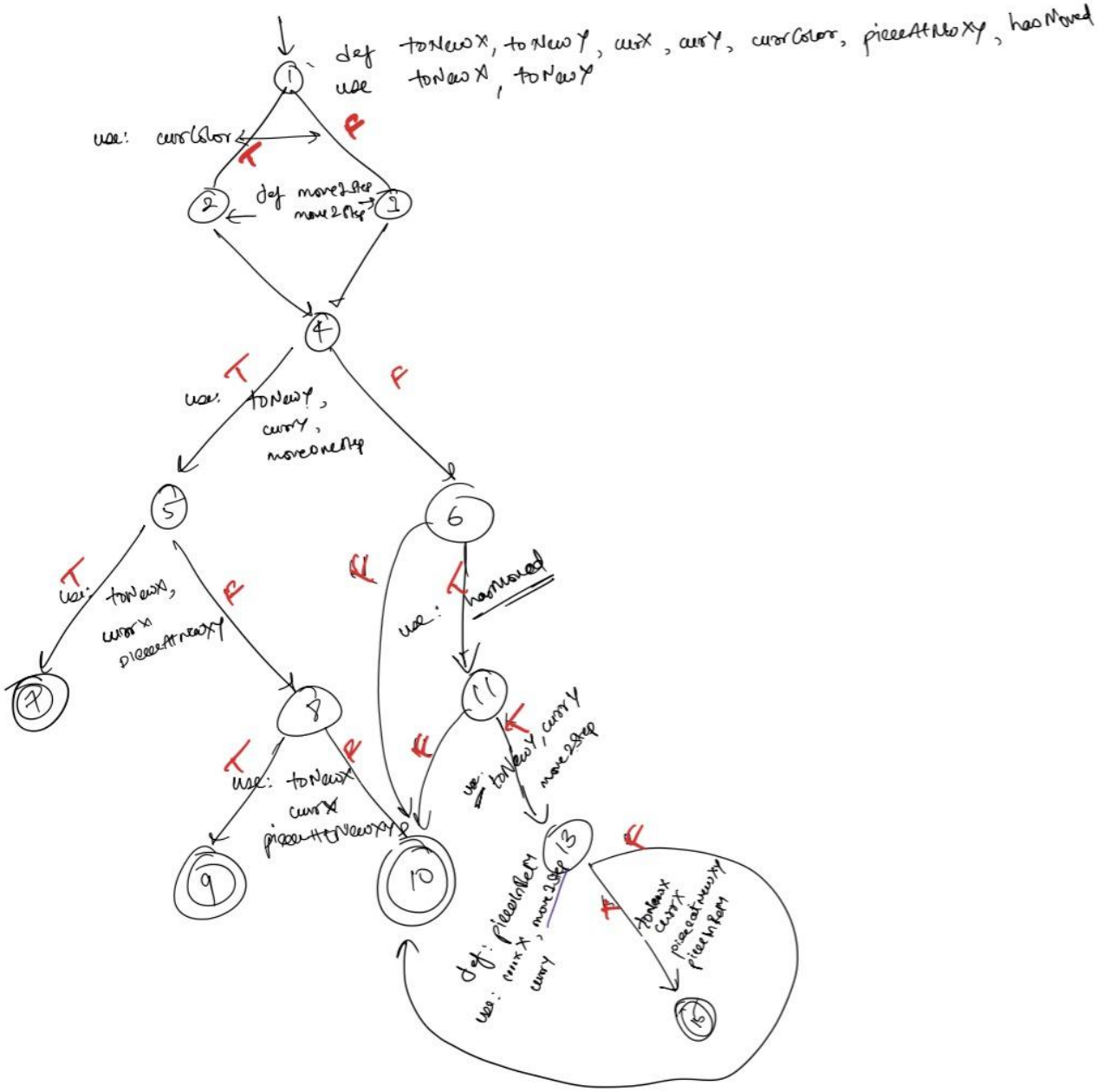
Variable	All DU Path Coverage
toNewX	No path or No path needed
toNewY	No path or No path needed
absX	[1,3,4] [1,2] [1,3,5]
absY	[1,3,4] [1,2] [1,3,5]
currX	No path or No path needed
currY	No path or No path needed

JUnit Test Case:

```
@Test
public void checkKnightMovementTest() {
    Knight knight = chessGame.addKnight( x: 4,  y: 4, COLOR.WHITE);

    // Test Path: [1, 2] : knight is moving 2 step in x direction and 1 step in y direction
    assertEquals( expected: true, knight.checkKnightMovement( toNewX: 6,  toNewY: 3));
    assertEquals( expected: true, knight.checkKnightMovement( toNewX: 6,  toNewY: 5));
    assertEquals( expected: true, knight.checkKnightMovement( toNewX: 2,  toNewY: 3));
    assertEquals( expected: true, knight.checkKnightMovement( toNewX: 2,  toNewY: 5));

    // Test Path: [1, 3, 4] : knight is moving 2 step in y direction and 1 step in x direction
    assertEquals( expected: true, knight.checkKnightMovement( toNewX: 3,  toNewY: 2));
    assertEquals( expected: true, knight.checkKnightMovement( toNewX: 5,  toNewY: 2));
    assertEquals( expected: true, knight.checkKnightMovement( toNewX: 3,  toNewY: 6));
    assertEquals( expected: true, knight.checkKnightMovement( toNewX: 3,  toNewY: 6));

    // Test Path: [1, 3, 5] : invalid move for a knight piece
    assertEquals( expected: false, knight.checkKnightMovement( toNewX: 5,  toNewY: 5));
}
```

3. Pawn.java : checkPawnMovement()

Data Flow Graph:



Test Requirements:

DU Paths for all variables are:

Variable	DU Paths
toNewX	[1,3,4,5,7] [1,3,4,5,8] [1,2,4,5,7] [1,2,4,5,8] [1,3,4,5,8,10] [1,3,4,5,8,9] [1,2,4,5,8,10] [1,2,4,5,8,9] [1,3,4,6,11,13,10] [1,3,4,6,11,13,15] [1,2,4,6,11,13,10] [1,2,4,6,11,13,15]
toNewY	[1,2,4,6] [1,2,4,5] [1,3,4,6] [1,3,4,5] [1,2,4,6,11,13] [1,2,4,6,11,10] [1,3,4,6,11,13] [1,3,4,6,11,10]
currX	[1,3,4,6] [1,3,4,5] [1,2,4,6] [1,2,4,5] [1,3,4,5,8] [1,3,4,5,7] [1,2,4,5,8] [1,2,4,5,7] [1,3,4,6,11,13] [1,3,4,5,8,9] [1,3,4,5,8,10] [1,2,4,6,11,13] [1,2,4,5,8,9] [1,2,4,5,8,10] [1,3,4,6,11,13,15] [1,3,4,6,11,13,10] [1,2,4,6,11,13,15] [1,2,4,6,11,13,10]
currY	[1,3,4,6] [1,3,4,5] [1,2,4,6] [1,2,4,5] [1,3,4,6,11,13] [1,3,4,6,11,10] [1,2,4,6,11,13] [1,2,4,6,11,10]
currColor	[1,3] [1,2]
pieceAtNewXY	[1,3,4,5,8] [1,3,4,5,7] [1,2,4,5,8] [1,2,4,5,7] [1,3,4,5,8,9] [1,3,4,5,8,10] [1,2,4,5,8,9] [1,2,4,5,8,10] [1,3,4,6,11,13,15] [1,3,4,6,11,13,10] [1,2,4,6,11,13,15] [1,2,4,6,11,13,10]
moveOneStep	[2,4,6] [2,4,5] [2,4,6,11,13] [3,4,6] [3,4,5] [3,4,6,11,13]
moveTwoStep	[2,4,6,11,13] [2,4,6,11,10] [3,4,6,11,13] [3,4,6,11,10]
hasMoved	[1,3,4,6,11] [1,3,4,6,10] [1,2,4,6,11] [1,2,4,6,10]
pieceInBetween	[13,15] [13,10]

Test Paths:

All Def Coverage for all variables are:

Variable	All Def Coverage
toNewX	[1,3,4,5,7]
toNewY	[1,2,4,6,10]
currX	[1,3,4,6,10]
currY	[1,3,4,6,10]
currColor	[1,3,4,6,10]
pieceAtNewXY	[1,3,4,5,8,9]
moveOneStep	[1,2,4,6,10] [1,3,4,6,10]
moveTwoStep	[1,2,4,6,11,13,15] [1,3,4,6,11,13,15]
hasMoved	[1,3,4,6,11,10]
pieceInBetween	[1,3,4,6,11,13,15]

All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
toNewX	[1,3,4,5,7] [1,3,4,5,8,9] [1,2,4,5,7] [1,2,4,5,8,9] [1,3,4,5,8,10] [1,2,4,5,8,10] [1,3,4,6,11,13,10] [1,3,4,6,11,13,15] [1,2,4,6,11,13,10] [1,2,4,6,11,13,15]
toNewY	[1,2,4,6,10] [1,2,4,5,7] [1,3,4,6,10] [1,3,4,5,7] [1,2,4,6,11,13,15] [1,2,4,6,11,10] [1,3,4,6,11,13,15] [1,3,4,6,11,10]
currX	[1,3,4,6,10] [1,3,4,5,7] [1,2,4,6,10] [1,2,4,5,7] [1,3,4,5,8,9] [1,2,4,5,8,9] [1,3,4,6,11,13,15] [1,3,4,5,8,10] [1,2,4,6,11,13,15] [1,2,4,5,8,10] [1,3,4,6,11,13,10] [1,2,4,6,11,13,10]
currY	[1,3,4,6,10] [1,3,4,5,7] [1,2,4,6,10] [1,2,4,5,7] [1,3,4,6,11,13,15] [1,3,4,6,11,10] [1,2,4,6,11,13,15] [1,2,4,6,11,10]
currColor	[1,3,4,6,10] [1,2,4,6,10]
pieceAtNewXY	[1,3,4,5,8,9] [1,3,4,5,7] [1,2,4,5,8,9] [1,2,4,5,7] [1,3,4,5,8,10] [1,2,4,5,8,10] [1,3,4,6,11,13,15] [1,3,4,6,11,13,10] [1,2,4,6,11,13,15] [1,2,4,6,11,13,10]
moveOneStep	[1,2,4,6,10] [1,2,4,5,7] [1,2,4,6,11,13,15] [1,3,4,6,10] [1,3,4,5,7] [1,3,4,6,11,13,15]
moveTwoStep	[1,2,4,6,11,13,15] [1,2,4,6,11,10] [1,3,4,6,11,13,15] [1,3,4,6,11,10]
hasMoved	[1,3,4,6,11,10] [1,3,4,6,10] [1,2,4,6,11,10] [1,2,4,6,10]
pieceInBetween	[1,3,4,6,11,13,15] [1,3,4,6,11,13,10]

JUnit Test Case:

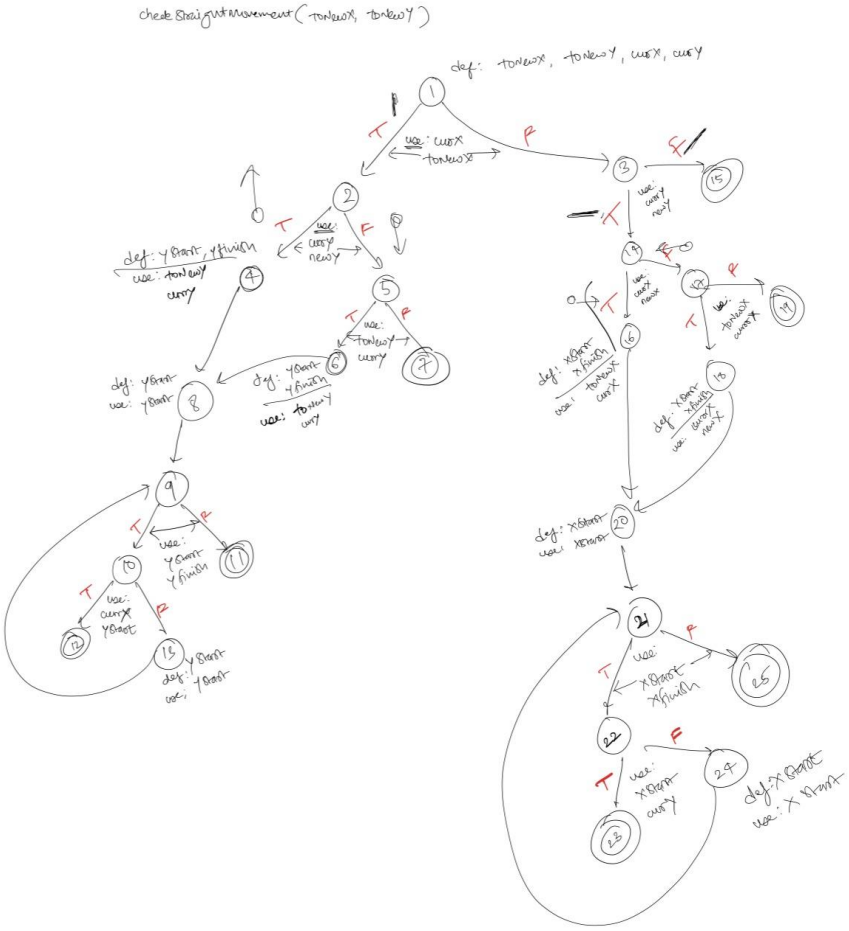
```
@Test
public void checkPawnMovementTest() {
    Pawn whitePawn1 = chessGame.addPawn( x: 0,  y: 6, COLOR.WHITE);
    Pawn blackPawn1 = chessGame.addPawn( x: 0,  y: 1, COLOR.BLACK);

    // Test Path: [1, 2, 4, 5, 7] : this is for black pawn moving 1 step forward and no piece at target location
    assertEquals( expected: true, blackPawn1.checkPawnMovement( toNewX: 0,  toNewY: 2));

    // Test Path: [1, 3, 4, 5, 7] : this is for white pawn moving 1 step forward and no piece at target location
    assertEquals( expected: true, whitePawn1.checkPawnMovement( toNewX: 0,  toNewY: 5));

    // Test Path: [1, 2, 4, 6, 10] : this is the black pawn with hasMoved == true, and it is trying to move 2 step forward
    blackPawn1.moveTo( x: 0,  y: 2); // moved to 0,2
    assertEquals( expected: false, blackPawn1.checkPawnMovement( toNewX: 0,  toNewY: 4)); // check movement from 0,2 to 0,4

    // Test Path: [1, 3, 4, 6, 10] : this is the white pawn with hasMoved == true, and it is trying to move 2 step forward
    whitePawn1.moveTo( x: 0,  y: 5); // moved to 0,5
    assertEquals( expected: false, whitePawn1.checkPawnMovement( toNewX: 0,  toNewY: 3)); // check movement from 0,5 to 0,3

    // Test Path: [1, 2, 4, 5, 8, 9] // if there is any opponent's piece diagonal to black pawn
    Queen whiteQueen = chessGame.addQueen( x: 1,  y: 3, COLOR.WHITE);
    assertEquals( expected: true, blackPawn1.checkPawnMovement( toNewX: 1,  toNewY: 3));
    chessBoard.capturePiece(whiteQueen);

    // Test Path: [1, 3, 4, 5, 8, 9] // if there is any opponent's piece diagonal to white pawn
    Queen blackQueen = chessGame.addQueen( x: 1,  y: 4, COLOR.BLACK);
    assertEquals( expected: true, whitePawn1.checkPawnMovement( toNewX: 1,  toNewY: 4));
    chessBoard.capturePiece(blackQueen);
```

```
    // Test Path: [1, 2, 4, 5, 8, 10]
    // black pawn wants to move diagonal but there is no piece in that location
    assertEquals( expected: false, blackPawn1.checkPawnMovement( toNewX: 1,  toNewY: 3));
    // black pawn moves more than 1 step in horizontal direction
    assertEquals( expected: false, blackPawn1.checkPawnMovement( toNewX: 2,  toNewY: 3));


    // Test Path: [1, 3, 4, 5, 8, 10]
    // white pawn wants to move diagonal but there is no piece in that location
    assertEquals( expected: false, whitePawn1.checkPawnMovement( toNewX: 1,  toNewY: 4));
    // white pawn moves more than 1 step in horizontal direction
    assertEquals( expected: false, whitePawn1.checkPawnMovement( toNewX: 2,  toNewY: 4));


    // Test Path: [1, 2, 4, 6, 11, 10] // black pawn wants to move its first move with more than two steps
    Pawn blackPawn2 = chessGame.addPawn( x: 5,  y: 1, COLOR.BLACK);
    assertEquals( expected: false, blackPawn2.checkPawnMovement( toNewX: 5,  toNewY: 4));

    // Test Path: [1, 3, 4, 6, 11, 10] // white pawn wants to move its first move with more than two steps
    Pawn whitePawn2 = chessGame.addPawn( x: 7,  y: 6, COLOR.WHITE);
    assertEquals( expected: false, whitePawn2.checkPawnMovement( toNewX: 7,  toNewY: 3));

    // Test Path: [1, 2, 4, 6, 11, 13, 10]
    // black pawn moves diagonal 2 steps
    assertEquals( expected: false, blackPawn2.checkPawnMovement( toNewX: 4,  toNewY: 3));
    // black pawn moves 2 steps with an opponent piece in between
    Rook whiteRook = chessGame.addRook( x: 5,  y: 2, COLOR.WHITE);
    assertEquals( expected: false, blackPawn2.checkPawnMovement( toNewX: 5,  toNewY: 3));
    // black pawn moves 2 steps with opponent piece in that location
    whiteRook.moveTo( x: 5,  y: 3);
    assertEquals( expected: false, blackPawn2.checkPawnMovement( toNewX: 5,  toNewY: 3));
```

```
        // Test Path: [1, 3, 4, 6, 11, 13, 10]
        // white pawn moves diagonal 2 steps
        assertEquals( expected: false, whitePawn2.checkPawnMovement( toNewX: 6,  toNewY: 4));
        // white pawn moves 2 steps with an opponent piece in between
        Rook blackRook = chessGame.addRook( x: 7,  y: 5, COLOR.BLACK);
        assertEquals( expected: false, whitePawn2.checkPawnMovement( toNewX: 7,  toNewY: 4));
        // white pawn moves 2 steps with opponent piece in that location
        whiteRook.moveTo( x: 7,  y: 4);
        assertEquals( expected: false, whitePawn2.checkPawnMovement( toNewX: 7,  toNewY: 4));
        chessBoard.capturePiece(blackRook);
        chessBoard.capturePiece(whiteRook);

        // Test Path: [1, 2, 4, 6, 11, 13, 15] : black pawn moves two-step forward
        assertEquals( expected: true, blackPawn2.checkPawnMovement( toNewX: 5,  toNewY: 3));

        // Test Path: [1, 3, 4, 6, 11, 13, 15]: white pawn moves two-step forward
        assertEquals( expected: true, whitePawn2.checkPawnMovement( toNewX: 7,  toNewY: 4));
    }
```

4. ChessPiece.java: checkStraightMovement()

Data Flow Graph:



Test Requirement:

DU Paths for all variables are:

Variable	DU Paths
toNewX	[1,3] [1,2] [1,3,14,17] [1,3,14,16] [1,3,14,17,19] [1,3,14,17,18]
toNewY	[1,3,14] [1,3,15] [1,2,5] [1,2,4] [1,2,5,6] [1,2,5,7]
currX	[1,3] [1,2] [1,3,14,17] [1,3,14,16] [1,3,14,17,18] [1,3,14,17,19] [1,2,4,8,9,10,13] [1,2,4,8,9,10,12] [1,2,5,6,8,9,10,13] [1,2,5,6,8,9,10,12]
currY	[1,3,14] [1,3,15] [1,2,5] [1,2,4] [1,2,5,6] [1,2,5,7] [1,3,14,16,20,21,22,24] [1,3,14,16,20,21,22,23] [1,3,14,17,18,20,21,22,24] [1,3,14,17,18,20,21,22,23]
yStart	[4,8] [8,9,10] [8,9,11] [8,9,10,13] [8,9,10,12] [6,8] [13,9,10] [13,9,11] [13,9,10,13] [13,9,10,12]
yFinish	[4,8,9,10] [4,8,9,11] [6,8,9,10] [6,8,9,11]
xStart	[16,20] [18,20] [20,21,22] [20,21,25] [20,21,22,24] [20,21,22,23] [24,21,22] [24,21,25] [24,21,22,24] [24,21,22,23]
xFinish	[16,20,21,22] [16,20,21,25] [18,20,21,22] [18,20,21,25]

Test Paths:

All Def Coverage for all variables are:

Variable	All Def Coverage
toNewX	[1,3,15]
toNewY	[1,3,14,17,19]
currX	[1,3,15]
currY	[1,3,14,17,19]
yStart	[1,2,4,8,9,11] [1,2,4,8,9,10,12] [1,2,5,6,8,9,11] [1,2,4,8,9,10,13,9,10,12]
yFinish	[1,2,4,8,9,10,12] [1,2,5,6,8,9,10,12]
xStart	[1,3,14,16,20,21,25] [1,3,14,17,18,20,21,25] [1,3,14,16,20,21,22,23] [1,3,14,16,20,21,22,24,21,22,23]
xFinish	[1,3,14,16,20,21,22,23] [1,3,14,17,18,20,21,22,23]

All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
toNewX	[1,3,15] [1,2,5,7] [1,3,14,17,19] [1,3,14,16,20,21,25] [1,3,14,17,18,20,21,25]
toNewY	[1,3,14,17,19] [1,3,15] [1,2,5,7] [1,2,4,8,9,11] [1,2,5,6,8,9,11]
currX	[1,3,15] [1,2,5,7] [1,3,14,17,19] [1,3,14,16,20,21,25] [1,3,14,17,18,20,21,25] [1,2,4,8,9,10,13,9,11] [1,2,4,8,9,10,12] [1,2,5,6,8,9,10,13,9,11] [1,2,5,6,8,9,10,12]
currY	[1,3,14,17,19] [1,3,15] [1,2,5,7] [1,2,4,8,9,11] [1,2,5,6,8,9,11] [1,3,14,16,20,21,22,24,21,25] [1,3,14,16,20,21,22,23] [1,3,14,17,18,20,21,22,24,21,25] [1,3,14,17,18,20,21,22,23]
yStart	[1,2,4,8,9,11] [1,2,4,8,9,10,12] [1,2,4,8,9,10,13,9,11] [1,2,5,6,8,9,11] [1,2,4,8,9,10,13,9,10,12] [1,2,4,8,9,10,13,9,10,13,9,11]
yFinish	[1,2,4,8,9,10,12] [1,2,4,8,9,11] [1,2,5,6,8,9,10,12] [1,2,5,6,8,9,11]
xStart	[1,3,14,16,20,21,25] [1,3,14,17,18,20,21,25] [1,3,14,16,20,21,22,23] [1,3,14,16,20,21,22,24,21,25] [1,3,14,16,20,21,22,24,21,22,23] [1,3,14,16,20,21,22,24,21,22,24,21,25]
xFinish	[1,3,14,16,20,21,22,23] [1,3,14,16,20,21,25] [1,3,14,17,18,20,21,22,23] [1,3,14,17,18,20,21,25]

JUnit Test Case:

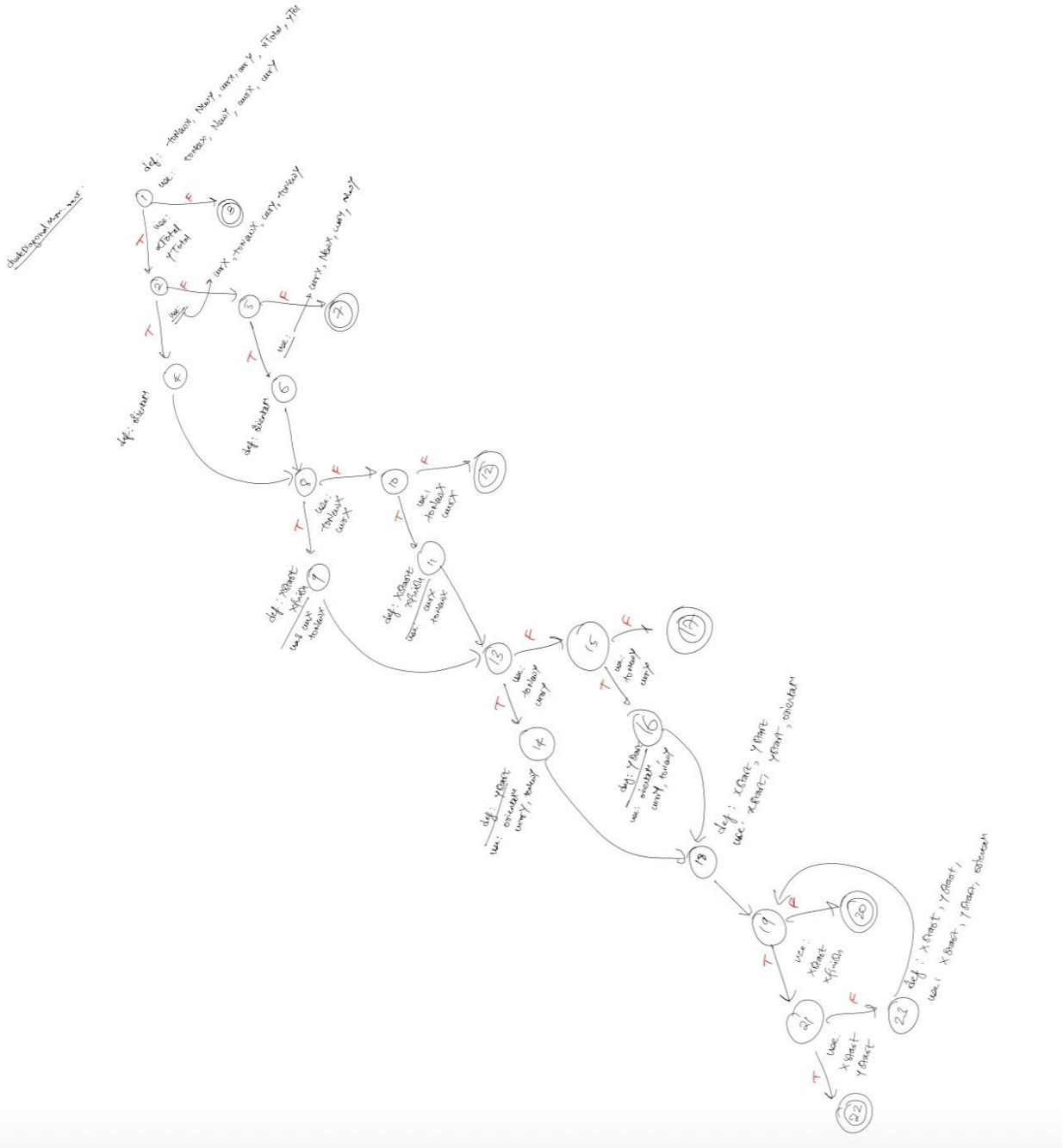
```
@Test
public void checkStraightMovementTest() {
    Rook blackRook = chessGame.addRook( x: 6, y: 2,COLOR.BLACK);
    Rook whiteRook = chessGame.addRook( x: 2, y: 6,COLOR.WHITE);

    // Test Path: [1, 3, 15] : Rook does not move straight
    assertEquals( expected: false, blackRook.checkStraightMovement( toNewX: 7, toNewY: 4));

    // Test Path: [1, 2, 5, 7] : Rook moves to same location it is already at
    assertEquals( expected: false, blackRook.checkStraightMovement( toNewX: 6, toNewY: 2));

    // Test Path: [1,2,4,8,9,11] : Rook moves one position vertically straight above
    assertEquals( expected: true, whiteRook.checkStraightMovement( toNewX: 2, toNewY: 5));

    // Test Path: [1,2,5,6,8,9,11] : Rook moves one position vertically straight down
    assertEquals( expected: true, blackRook.checkStraightMovement( toNewX: 6, toNewY: 6));

    // Test Path: [1,2,4,8,9,10,12] : There is opponent piece right below the rook
    Queen whiteQueen = chessGame.addQueen( x: 6, y: 3,COLOR.WHITE);
    assertEquals( expected: false, blackRook.checkStraightMovement( toNewX: 6, toNewY: 6));
    chessBoard.capturePiece(whiteQueen);

    // Test Path: [1,2,5,6,8,9,10,12] : There is opponent piece right above the rook
    Queen blackQueen = chessGame.addQueen( x: 2, y: 5,COLOR.BLACK);
    assertEquals( expected: false, whiteRook.checkStraightMovement( toNewX: 2, toNewY: 4));
    chessBoard.capturePiece(blackQueen);

    // Test Path: [1,2,5,6,8,9,10,13,9,11] : black Rook moves two positions straight down
    assertEquals( expected: true, blackRook.checkStraightMovement( toNewX: 6,  toNewY: 4));

    // Test Path: [1,2,4,8,9,10,13,9,11]  : white Rook moves two positions straight up
    assertEquals( expected: true, whiteRook.checkStraightMovement( toNewX: 2,  toNewY: 4));
```

```
        // Test Path: [1,2,4,8,9,10,13,9,10,12] : white rook moves up more than two position but finds opponents piece in its way
        blackQueen = chessGame.addQueen( x: 2, y: 4,COLOR.BLACK);
        assertEquals( expected: false, whiteRook.checkStraightMovement( toNewX: 2,  toNewY: 3));
        chessBoard.capturePiece(blackQueen);

        // Test Path: [1,2,4,8,9,10,13,9,10,13,9,11] : whiteRook moves more than two position straight up
        assertEquals( expected: true, whiteRook.checkStraightMovement( toNewX: 2,  toNewY: 3));

        // moving the white and black to test for horizontal moves
        blackRook.moveTo( x: 4,  y: 2);
        whiteRook.moveTo( x: 4,  y: 5);

        // Test Path: [1,3,14,16,20,21,25] : Rook moves one position horizontally straight right
        assertEquals( expected: true, blackRook.checkStraightMovement( toNewX: 5,  toNewY: 2));

        // Test Path: [1,3,14,17,18,20,21,25] : Rook moves one position horizontally straight left
        assertEquals( expected: true, blackRook.checkStraightMovement( toNewX: 3,  toNewY: 2));

        // Test Path: [1,3,14,16,20,21,22,23] : there is opponent piece just right side
        whiteQueen = chessGame.addQueen( x: 5,  y: 2, COLOR.WHITE);
        assertEquals( expected: false, blackRook.checkStraightMovement( toNewX: 6,  toNewY: 2));

        // Test Path: 1,3,14,17,18,20,21,22,23] : there is opponent piece just left side
        whiteQueen.moveTo( x: 3,  y: 2);
        assertEquals( expected: false, blackRook.checkStraightMovement( toNewX: 2,  toNewY: 2));

        // Test Path: [1,3,14,16,20,21,22,24,21,25]: black rook moves right more than two position but finds opponents piece in its way
        whiteQueen.moveTo( x: 6,  y: 2);
        assertEquals( expected: false, blackRook.checkStraightMovement( toNewX: 7,  toNewY: 2));
```

```
        // Test Path: [1,3,14,17,18,20,21,22,24,21,25]: black rook moves left more than two position but finds opponents piece in its way
        whiteQueen.moveTo( x: 2,  y: 2);
        assertEquals( expected: false, blackRook.checkStraightMovement( toNewX: 1,  toNewY: 2));
        chessBoard.capturePiece(whiteQueen);

        // Test Path: [1,3,14,16,20,21,22,24,21,22,24,21,25] : whiteRook moves more than two position straight up
        assertEquals( expected: true, blackRook.checkStraightMovement( toNewX: 0,  toNewY: 2));
    }
```

5. ChessPiece.java - checkDiagionalMovement()

Data Flow Graph:



Test Requirement:

DU Paths for all variables are:

Variable	DU Paths
toNewX	[1,2,5] [1,2,4] [1,2,5,7] [1,2,5,6] [1,2,4,8,10] [1,2,4,8,9] [1,2,5,6,8,9] [1,2,5,6,8,10] [1,2,4,8,10,12] [1,2,4,8,10,11] [1,2,5,6,8,10,11] [1,2,5,6,8,10,12]
toNewY	[1,2,5] [1,2,4] [1,2,5,6] [1,2,5,7] [1,2,4,8,9,13,15] [1,2,4,8,9,13,14] [1,2,5,6,8,9,13,15] [1,2,5,6,8,9,13,14] [1,2,4,8,10,11,13,15] [1,2,4,8,10,11,13,14] [1,2,4,8,9,13,15,16] [1,2,4,8,9,13,15,17] [1,2,5,6,8,10,11,13,15] [1,2,5,6,8,10,11,13,14] [1,2,5,6,8,9,13,15,16] [1,2,5,6,8,9,13,15,17] [1,2,4,8,10,11,13,15,16] [1,2,4,8,10,11,13,15,17] [1,2,5,6,8,10,11,13,15,16] [1,2,5,6,8,10,11,13,15,17]
currX	[1,2,5] [1,2,4] [1,2,5,6] [1,2,5,7] [1,2,4,8,10] [1,2,4,8,9] [1,2,5,6,8,10] [1,2,5,6,8,9] [1,2,4,8,10,11] [1,2,4,8,10,12] [1,2,5,6,8,10,11] [1,2,5,6,8,10,12]
currY	[1,2,5] [1,2,4] [1,2,5,6] [1,2,5,7] [1,2,4,8,9,13,15] [1,2,4,8,9,13,14] [1,2,5,6,8,9,13,15] [1,2,5,6,8,9,13,14] [1,2,4,8,10,11,13,15] [1,2,4,8,10,11,13,14] [1,2,4,8,9,13,15,16] [1,2,4,8,9,13,15,17] [1,2,5,6,8,10,11,13,15] [1,2,5,6,8,10,11,13,14] [1,2,5,6,8,9,13,15,16] [1,2,5,6,8,9,13,15,17] [1,2,4,8,10,11,13,15,16] [1,2,4,8,10,11,13,15,17] [1,2,5,6,8,10,11,13,15,16] [1,2,5,6,8,10,11,13,15,17]
xTotal	[1,2] [1,3]
yTotal	[1,2] [1,3]

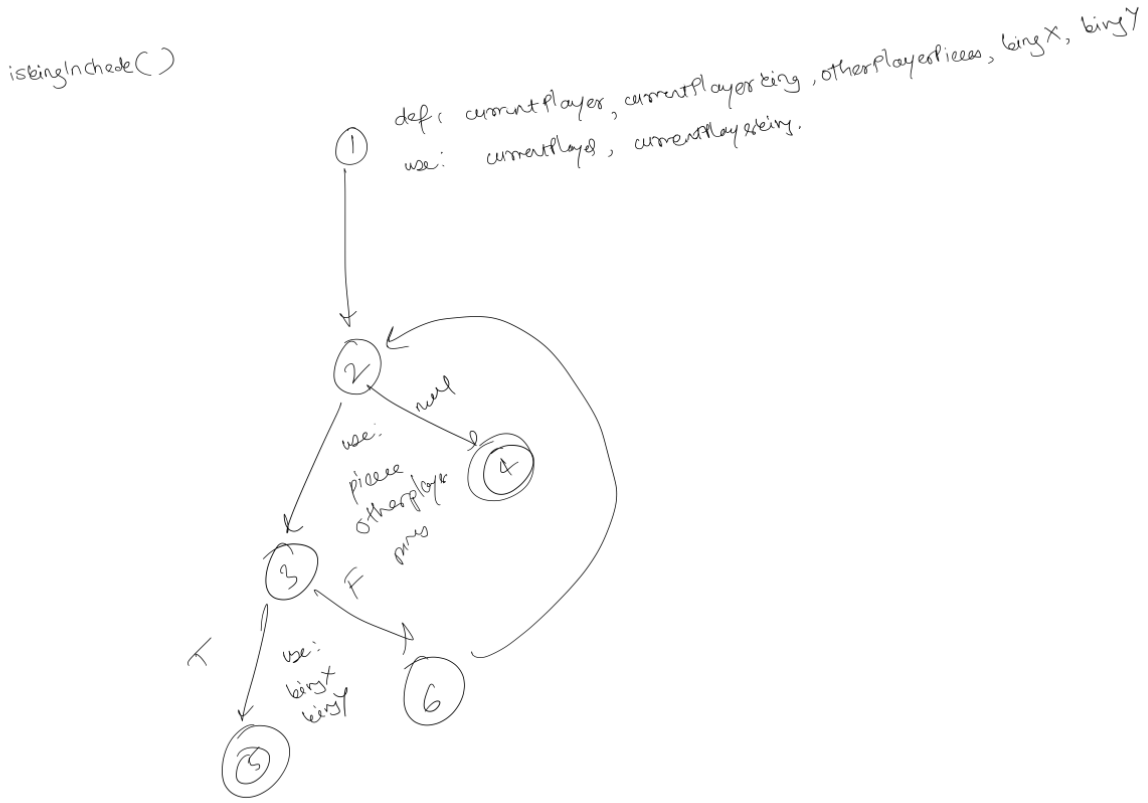
orientation	[4,8,9,13,14] [4,8,10,11,13,14] [4,8,9,13,15,16] [4,8,9,13,14,18] [4,8,10,11,13,15,16] [4,8,10,11,13,14,18] [4,8,9,13,15,16,18] [4,8,10,11,13,15,16,18] [4,8,9,13,14,18,19,21,23] [4,8,10,11,13,14,18,19,21,23] [4,8,9,13,15,16,18,19,21,23] [4,8,10,11,13,15,16,18,19,21,23] [6,8,9,13,14] [6,8,10,11,13,14] [6,8,9,13,15,16] [6,8,9,13,14,18] [6,8,10,11,13,15,16] [6,8,10,11,13,14,18] [6,8,9,13,15,16,18] [6,8,10,11,13,15,16,18] [6,8,9,13,14,18,19,21,23] [6,8,10,11,13,14,18,19,21,23] [6,8,9,13,15,16,18,19,21,23] [6,8,10,11,13,15,16,18,19,21,23]
xStart	[9,13,14,18] [9,13,15,16,18] [11,13,14,18] [11,13,15,16,18] [18,19,21] [18,19,20] [18,19,21,23] [18,19,21,22] [23,19,21] [23,19,20] [23,19,21,23] [23,19,21,22]
xFinish	[9,13,14,18,19,21] [9,13,14,18,19,20] [9,13,15,16,18,19,21] [9,13,15,16,18,19,20] [11,13,14,18,19,21] [11,13,14,18,19,20] [11,13,15,16,18,19,21] [11,13,15,16,18,19,20]
yStart	[14,18] [16,18] [18,19,21,23] [18,19,21,22] [23,19,21,23] [23,19,21,22]

Test Paths Along With JUnit Test Case:

```
    @Test
    public void checkDiagonalMovementTest() {
        Queen blackQueen = chessGame.addQueen( x: 4,  y: 4, COLOR.BLACK);

        // TestPath: [1, 3]: queen moves non-diagonal
        assertEquals( expected: false, blackQueen.checkDiagonalMovement( toNewX: 4,  toNewY: 6));

        // Test Path: [1, 2, 4, 8, 9, 13, 14, 18, 19, 21, 23, 19, 20] : queen moves diagonally from 4, 4 to 1, 1, with no opponent's piece on the path
        assertEquals( expected: true, blackQueen.checkDiagonalMovement( toNewX: 1,  toNewY: 1));

        // Test Path: [1, 2, 4, 8, 9, 13, 14, 18, 19, 21, 23, 19, 21, 22] : queen moves diagonally from 4, 4 to 1, 1, with opponent's piece on the path
        Pawn whitePawn = chessGame.addPawn( x: 2,  y: 2, COLOR.WHITE);
        assertEquals( expected: false, blackQueen.checkDiagonalMovement( toNewX: 1,  toNewY: 1));

        // Test Path: [1, 2, 4, 8, 9, 13, 14, 18, 19, 21, 22] : queen moves diagonally from 4, 4 to 1, 1, with opponent's piece is at 3, 3
        whitePawn.moveTo( x: 3,  y: 3);
        assertEquals( expected: false, blackQueen.checkDiagonalMovement( toNewX: 1,  toNewY: 1));
    }
}
```

6. ChessGame.java - isKingInCheck()

Data Flow Graph:



Test Requirement:

DU Paths for all variables are:

Variable	DU Paths
currentPlayer	No path or No path needed
currentPlayerKing	No path or No path needed
otherPlayerPieces	[1,2,3] [1,2,4]
kingX	[1,2,3,6] [1,2,3,5]
kingY	[1,2,3,6] [1,2,3,5]

Test Paths:

All Def Coverage for all variables are:

Variable	All Def Coverage
currentPlayer	No path or No path needed
currentPlayerKing	No path or No path needed
otherPlayerPieces	[1,2,3,5]
kingX	[1,2,3,6,2,4]
kingY	[1,2,3,6,2,4]

All DU Path Coverage for all variables are:

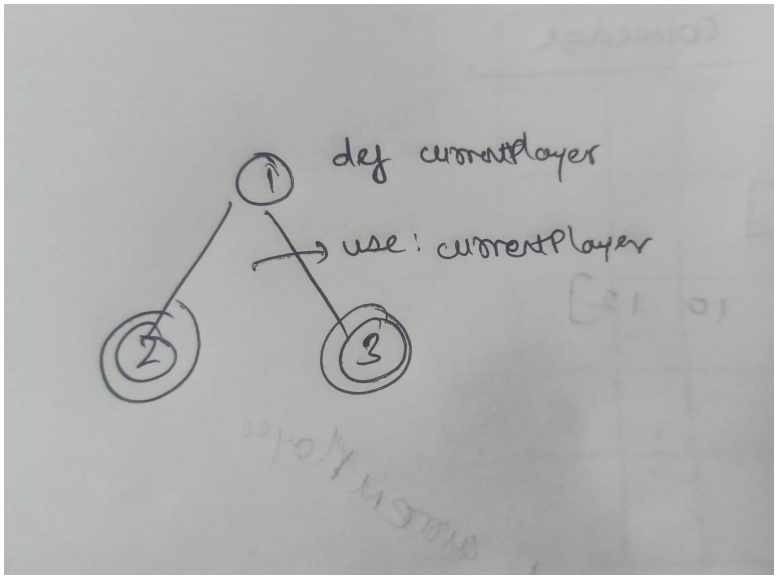
Variable	All DU Path Coverage
currentPlayer	No path or No path needed
currentPlayerKing	No path or No path needed
otherPlayerPieces	[1,2,3,5] [1,2,4]
kingX	[1,2,3,6,2,4] [1,2,3,5]
kingY	[1,2,3,6,2,4] [1,2,3,5]

JUnit Test Case:

```
no usages  new *
@Test
public void isKingInCheckTest() {
    King blacKing = chessGame.addKing( x: 3,  y: 2, COLOR.BLACK);

    // TestPath: [1, 2, 4]: no opponent's piece to give on the king
    assertEquals( expected: false, chessGame.isKingInCheck(COLOR.BLACK));


    // TestPath: [1, 2, 3, 5]: white queen gives check to the black king
    Queen whiteQueen = chessGame.addQueen( x: 5,  y: 4, COLOR.WHITE);
    assertEquals( expected: true, chessGame.isKingInCheck(COLOR.BLACK));
    chessBoard.capturePiece(whiteQueen);

    // TestPath: [1, 2, 3, 6, 2, 4]: checking for check with more than two opponent piece, no check
    Pawn whitePawn1 = chessGame.addPawn( x: 6,  y: 6, COLOR.WHITE);
    Pawn whitePawn2 = chessGame.addPawn( x: 5,  y: 6, COLOR.WHITE);
    assertEquals( expected: false, chessGame.isKingInCheck(COLOR.BLACK));
}
```

7. ChessGame.java - isCheckMate()

Data Flow Graph:



Test Requirement:

DU Paths for all variables are:

Variable	DU Paths
currentPlayer	[1,3] [1,2]

Test Paths:

All Def Coverage for all variables are:

Variable	All Def Coverage
currentPlayer	[1,3]

All DU Path Coverage for all variables are:

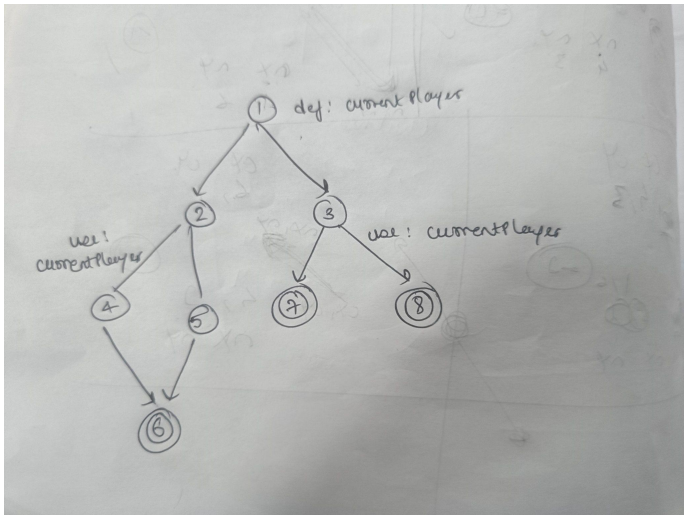
Variable	All DU Path Coverage
currentPlayer	[1,3] [1,2]

JUnit Test Case:

```
no usages  new *
@Test
public void isCheckMateTest() {
    King blackKing = chessGame.addKing( x: 7,  y: 7, COLOR.BLACK);
    Queen whiteQueen = chessGame.addQueen( x: 5,  y: 7, COLOR.WHITE);
    Queen whiteRook = chessGame.addQueen( x: 7,  y: 5, COLOR.WHITE);

    // TestPath: [1, 2]: checkmate
    assertEquals( expected: true, chessGame.isCheckMate(COLOR.BLACK));

    // TestPath: [1, 3]: no checkmate
    chessBoard.capturePiece(whiteRook); // removing whiteRook to remove the checkmate on the king
    assertEquals( expected: false, chessGame.isCheckMate(COLOR.BLACK));
}
```

8. ChessGame.java - isGameOver()

Data Flow Graph:



Test Requirements:

DU Paths for all variables are:

Variable	DU Paths
currentPlayer	[1,3] [1,2] [1,3,8] [1,3,7] [1,2,5] [1,2,4]

Test Paths:

All Def Coverage for all variables are:

Variable	All Def Coverage
currentPlayer	[1,3,7]

All DU Path Coverage for all variables are:

Variable	All DU Path Coverage
currentPlayer	[1,3,7] [1,2,4,6] [1,3,8] [1,2,5,6]

JUnit Test Case:

```
no usages  new *
@Test
public void isGameOverTest() {
    // TestPath: [1 3 8] : no checkmate, no game over
    chessGame.setCurrentPlayer(COLOR.BLACK);
    King blackKing = chessGame.addKing( x: 7,  y: 7, COLOR.BLACK);
    Queen whiteQueen = chessGame.addQueen( x: 5,  y: 7, COLOR.WHITE);
    assertEquals( expected: false, chessGame.isGameOver());

    // TestPath: [1 3 7] : stalemate
    Rook whiteRook = chessGame.addRook( x: 6,  y: 6, COLOR.WHITE);
    assertEquals( expected: true, chessGame.isGameOver());
}
```

Contributions:

Jayaa Shree Laxmi Kishoore (MT2022053)
a. Drew DFG for checkStraightMovement, checkDiagonalMovement, isGameOver, checkPawnMovement.
b. Designed JUnit tests for the above functions.
c. Cross-verified other functions

Shashidhar Basavaraj Nagaral (MT2022108)
a. Drew CFG for boundCheck, checkKnightMovement, isKingInCheck, isCheckMate,
b. Designed JUnit tests for the above functions.
c. Cross-verified other functions