**TEST REPORT**

**Test for ValidateMoveService.py**

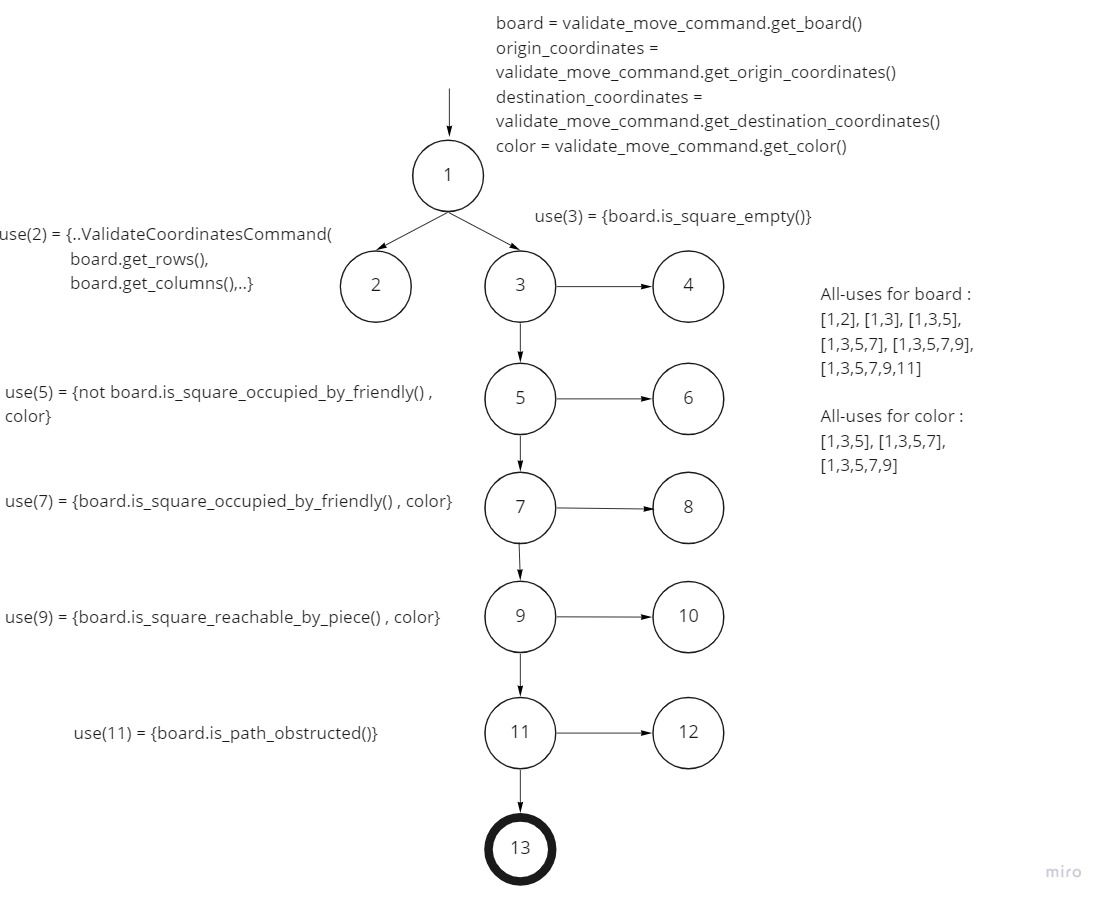
Class covered: ValidateMoveService

Coverage criteria used: All-uses

Coverage obtained: 100%

Testing steps:

Class consisted of mostly if statements with exceptions raised within each if statement. Because of the layout of the code in this class, all-use coverage criteria seemed reasonable to use. In order to cover all uses within the test file, a test was created for each instance where an exception could be raised. This includes exceptions OriginSquareEmptyException, OriginSquareContainsEnemyPieceException, DestinationSquareOccupiedException, InvalidMovementForPieceException, and PieceMovementPathObstructedException (these are all exceptions created by the author of this repo). Variables board, origin\_coordinates, destination\_coordinates, and color were all defined in node 1. Their uses varied slightly however, with color only being used within some if statements, and board, origin\_coordinates, and destination\_coordinates being used for every if statement.

Since all exceptions were covered and passed, 100% coverage was obtained.

\* since board, origin\_coordinates, and destination\_coordinates are all used at the same time within this class, only board is tracked for clarity, color is tracked separately

**Test for GoldGeneral.py**

Class covered: GoldGeneral(Piece)

Coverage criteria used: Base choice

Coverage obtained: 100%

Testing steps:

Since this class was about the Gold General, I thought base choice coverage would be fitting because I can go over characteristics of the game piece. Below is the table of the base choice blocks and characteristics associated with it. Originally the strikethrough cells were going to be implemented, but couldn’t get characteristics from both blocks asserted within the same function to setup proper coverage. Description and Representation characteristics covered def\_\_init\_\_() function within GoldGeneral class through asserts. Movement characteristic covered can\_reach() function, usage of assertTrue and assertFalse satisfied if else conditions throughout the class. Black and white piece move sets were also account for to further satisfy remaining if else conditions.

|  |  |  |
| --- | --- | --- |
| Characteristics (X) | Block 1 (X1) | Block 2 (X2) |
| Description (A) | “Gold General” (A1) | “Silver General” (A2) |
| Representation (B) | ‘G’ (B1) | ‘S’ (B2) |
| Movement/Move set (C) | One space in any direction except back-diagonal; up, down, left, right, up right, up left (C1) | One space diagonally or forward (C2) |
| ~~Promotion (D)~~ | ~~Does not promote (D1)~~ | ~~Promotes to Promoted Silver (D2)~~ |
| ~~Piece Count (E)~~ | ~~2~~ | ~~2~~ |

**Test for SilverGeneral.py**

Class covered: SilverGeneral(Piece)

Coverage criteria used: Base choice

Coverage obtained: 100%

Testing steps:

Just like the Gold General, since this class was about the Silver General, I thought base choice coverage would be fitting because I can go over characteristics of the game piece. Below is the table of the base choice blocks and characteristics associated with it. Description and Representation characteristics covered def\_\_init\_\_() function within SilverGeneral class through asserts. Movement characteristic covered can\_reach() function, usage of assertTrue and assertFalse satisfied if else conditions throughout the class. Black and white piece moves sets were also account for to further satisfy remaining if else conditions.

|  |  |  |
| --- | --- | --- |
| Characteristics (X) | Block 1 (X1) | Block 2 (X2) |
| Description (A) | “Gold General” (A1) | “Silver General” (A2) |
| Representation (B) | ‘G’ (B1) | ‘S’ (B2) |
| Movement/Move set (C) | One space diagonally or forward; up, up right, up left, down left, down right (C1) | One space in any direction except back-diagonal; up, down, left, right, up right, up left (C2) |

**Test for Knight.py**

Class covered: Knight(Piece)

Coverage criteria used: Base choice

Coverage obtained: 100%

Testing steps:

Another Shogi piece; base choice is used as coverage for characteristics of the game piece. Below is the table of the base choice blocks and characteristics associated with it. Description and Representation characteristics covered def\_\_init\_\_() function within Knight class through asserts. Movement characteristic covered can\_reach() function, usage of assertTrue and assertFalse satisfied if else conditions throughout the class. Black and white piece moves sets were also account for to further satisfy remaining if else conditions.

|  |  |  |
| --- | --- | --- |
| Characteristics (X) | Block 1 (X1) | Block 2 (X2) |
| Description (A) | “Knight” (A1) | “Lance” (A2) |
| Representation (B) | ‘N’ (B1) | ‘L’ (B2) |
| Movement/Move set (C) | Only forward 2 spaces and then move left or right 1 space; L-shape right, L-shape left (C1) | As many spaces as desired, only forward; up\* (C2) |

**Test for Pawn.py**

Class covered: Pawn(Piece)

Coverage criteria used: Each choice

Coverage obtained: 100%

Testing steps:

This time each choice was used for coverage criteria. Since this type of coverage is used instead of base choice, there will be a third block for more combinations. Number of tests is only at least largest number of blocks among all characteristics (3). A separate list of asserts was created for white and black piece move sets for more coverage, as well as true/false asserts to cover if/else conditions within Pawn.py. Although coverage was 100%, the test criteria itself isn’t very strong.

|  |  |  |  |
| --- | --- | --- | --- |
| Characteristics (X) | Block 1 (X1) | Block 2 (X2) | Block 3 (X3) |
| Description (A) | “Pawn” (A1) | “King” (A2) | “Bishop” (A3) |
| Representation (B) | ‘P’ (B1) | ‘K’ (B2) | ‘B’ (B3) |
| Movement/Move set (C) | Only forward 1 space; up (C1) | One space in any direction; up, down, left, right, up right, up left, down right, down left (C2) | Diagonal as many spaces as desired, forward\* and backward\* (C3) |

**Summary**

Initially, deciding on which coverage criteria to use was difficult due to lack of understanding of the source code. Being able to decipher someone else’s code will always be time consuming it seems, but a good skill to pick up. This was also my first experience doing tests for Python files which was surprisingly very similar in syntax compared to Java’s JUnit testing. In addition, my choice of coverage criteria was too simple, causing test creation to be very tedious and prone to human error.