Randoop Report:

Randoop generates two kinds of unit tests by default : Error-revealing tests and Regression tests. Error-revealing tests are tests that fail when executed, indicating a potential error. Regression tests are tests that pass when executed. Whenever changes are made to the source code, running regression tests can identify errors introduced in the latest changes to the source code.

When randoop was run for generating tests for our source code, the following output was generated:

```
policy = sun.security.provider.PolicyFile@448139f0
Ignoring abstract class ChessPiece specified via --classlist or --testclass.
PUBLIC MEMBERS=34
Explorer = randoop.sequence.ForwardGenerator@4d76f3f8

Progress update: test inputs generated=0, failing inputs=0     (Thu Apr 07 12:58:39 MDT 2016)
Progress update: test inputs generated=316, failing inputs=0     (Thu Apr 07 12:58:44 MDT 2016)
Progress update: test inputs generated=467, failing inputs=0     (Thu Apr 07 12:58:49 MDT 2016)
Progress update: test inputs generated=626, failing inputs=0     (Thu Apr 07 12:58:54 MDT 2016)
Progress update: test inputs generated=753, failing inputs=0     (Thu Apr 07 12:58:59 MDT 2016)
Progress update: test inputs generated=878, failing inputs=0     (Thu Apr 07 12:59:04 MDT 2016)
Progress update: test inputs generated=1038, failing inputs=0     (Thu Apr 07 12:59:09 MDT 2016)
Progress update: test inputs generated=1162, failing inputs=0     (Thu Apr 07 12:59:14 MDT 2016)
Progress update: test inputs generated=1295, failing inputs=0     (Thu Apr 07 12:59:19 MDT 2016)
Progress update: test inputs generated=1421, failing inputs=0     (Thu Apr 07 12:59:24 MDT 2016)
Progress update: test inputs generated=1523, failing inputs=0     (Thu Apr 07 12:59:29 MDT 2016)
Progress update: test inputs generated=1653, failing inputs=0     (Thu Apr 07 12:59:34 MDT 2016)
Progress update: test inputs generated=1762, failing inputs=0     (Thu Apr 07 12:59:39 MDT 2016)
Normal method executions:806217
Exceptional method executions:244

Average method execution time (normal termination):     0.0736
Average method execution time (exceptional termination): 0.118

No error-revealing tests to output



Regression test output:
Writing 909 junit tests

Created file: C:\Users\skshimon\Documents\Randoop\Chess_Output\RegressionTest0.java
Created file: C:\Users\skshimon\Documents\Randoop\Chess_Output\RegressionTest1.java
Created file: C:\Users\skshimon\Documents\Randoop\Chess_Output\RegressionTest.java
```

Our run did not generate any error-revealing tests, and generated a regression test suite of 909 test cases, which document the current behavior of the classes under test.

One interesting thing about the generated test inputs is that for multiple test inputs – the test being performed is the same. For example, let's say testcase 1 initializes two chessboards 'x' and 'y' with pattern 'p' and 'q' respectively. For testing, it just tests chessboard 'x' for pattern 'p'. testcase 2 initializes three chessboards 'x' , 'y' and 'z' with pattern 'p' , 'q' and 'r' respectively. For testing, it just tests chessboard 'x' for pattern 'p'. In effect, both test cases are the same and can be reduced to one single test case.
This type of test input is generated as Randoop builds test inputs incrementally by incorporating feedback obtained from executing previous test inputs. Testcase 2 was build upon testcase 1 by adding a initialization statement for chessboard z with pattern 'r' , which was not used in testing later.

Artificial Fault Seeding:
Fault 1:

Bishop.java, line 33.
Original line: `for(int OffSet=1;OffSet<=7;OffSet++){`
Modified line: `for(int OffSet=1;OffSet<7;OffSet++){`

The effect of this line is that if the bishop is at one end of the chessboard, and if there is a possible move to the other end of the chess board, the possible move to the other end will not be counted as a valid possible move for the bishop.

When we ran our regression test, none of the test cases were able to detect this fault.

Fault 2:

Rook.java, line 33.
Original line: `for(int OffSet=1;OffSet<=7;OffSet++){`
Modified line: `for(int OffSet=1;OffSet<7;OffSet++){`

The effect of this line is that if the rook is at one end of the chessboard, and if there is a possible move on the other end of the chess board, the possible move to the other end will not be counted as a valid possible move for the rook.

When we ran our regression test, none of our test cases were able to detect this fault.

Fault 3:

Pawn.java , line 35.
Original line : `if(this.getRow()==1){`
Modified line : `if(this.getRow()==0){`

If the White pawn is at Row 1, it will be it's opening move and 2 movements are allowed. The effect of the modified line is, for White pawn, 2 movements as a valid opening movement will be valid only if the White pawn is at Row 0 (which is not possible, since when the chessboard is initialized, White pawns are initialized at Row 1).

When we ran our regression test, none of our test cases were able to detect this fault.

Fault 4:

Pawn.java, line 77.
Original line: `if(this.getRow()==6){`
Modified line: `if(this.getRow()==7){`

If the Black pawn is at Row 1, it will be it's opening move and 2 movements are allowed. The effect of the modified line is, for Black pawn, 2 movements as a valid opening movement will be valid only if the Black pawn is at Row 7 (which is not possible, since when the chessboard is initialized , Black pawns are initialized at Row 6).

When we ran our regression test, none of our test cases were able to detect this fault.

Fault 5:

Pawn.java, line 26.
Original line: `int nextRow = this.getRow()+1;`
Modified line: `int nextRow = this.getRow()-1;`

According to the original specification, white pawns can only move up and Black pawns can only go down. The effect of this modified line is that White pawns will be able to move only downwards.

When we ran our regression test, 75 test cases showed errors and were able to detect this fault. We observed that because of the artificial fault introduced, a call to **legalMoves()** method of any instance variable of class **Pawn** resulted in a unhandled nullpointer exception.

Fault 6:

Pawn.java, line 68.
Original line: `int nextRow = this.getRow()-1;`
Modified line: `int nextRow = this.getRow()+1;`

According to the original specification, Black pawns can only move down and White pawns can only go up. The effect of this modified line is that Black pawns will be able to move only upwards.

When we ran our regression test, none of the generated test cases were able to detect this fault.

Fault 7:

Rook.java, line 28.
Original line: `int[] verDirection = { 0,0,-1,+1 };`
Modified line: `int[] verDirection = { 0,0,0,+1 };`

According to the specification, Rooks are able to move only vertically (up or down) , and horizontally (left and right). Because of the modification, rooks will be able to move vertically in only upward direction.

When we ran our regression test, none of the generated test cases were able to detect this fault.

Fault 8:

Rook.java, line 27.
Original line: `int[] horDirection = { -1,+1,0,0 };`
Modified line: `int[] horDirection = { 0,+1,0,0 };`

According to the specification, Rooks are able to move only vertically (up or down) , and horizontally (left and right). Because of the modification, rooks will not be able to move to left in horizontal direction.

When we ran our regression test, none of the generated test cases were able to detect this fault.

Fault 9:

Queen.java, line 15.
Original line: `return "\u2655";`
Modified line: `return "\u2656";`

The original line returns the Unicode character of White Queen when called, and the modified line returns Unicode character of white chess rook.

When we ran our regression test, we found 59 failures that were able to detect this fault. All of these test cases failed at the point where the generated chess board was being compared against a chessboard with correct Unicode characters for the chess pieces.

Fault 10:

Chesspiece.java, line 73.
Original line: `row = digit -'0' -1;`
Modified line: `row = digit -'0' +1;`

The original line takes the position of a chess piece in string (for example “a3”) and converts it to appropriate row and column position of the 8x8 2D chessboard array. The string representation of a position is converted to a 0^{th} index based array position in the original line, and the modified line will result in a corresponding row position in the array which will result in array out of bounds exception.

When we ran the regression tests, 10 test cases failed. These test cases all involve either getRow() or getPosition() method at the point of failure – where a call to these methods is made to check against the correct expected row position, or position represented in string.