StrategoTests

January 5, 2017

Contents

1 BoardTest

```
class BoardTest is subclass of TestCase
        instance variables
                s : Stratego := new Stratego();
        operations

            -- Tests with valid Inputs

                private testValidCoords: () ==> ()
                testValidCoords() ==
                (
                        assertEqual(true,s.getBoard().validCoords(
                            mk_Board`Position(0,0)));

                        assertEqual(true,s.getBoard().validCoords(
                            mk_Board`Position(9,5)));

                        assertEqual(false,s.getBoard().validCoords(
                            mk_Board`Position(15,5)));
                );


                private testFreeSpace: () ==> ()
                testFreeSpace() ==
                (
                        assertEqual(true,s.getBoard().freeSpace(
                            mk_Board`Position(0,0)));

                        assertEqual(false,s.getBoard().freeSpace(
                            mk_Board`Position(2,5)));
```

```
        assertEqual(false,s.getBoard().freeSpace(
            mk_Board`Position(3,4)));

        assertEqual(true,s.getBoard().freeSpace(
            mk_Board`Position(9,9)));
);


private placePiece: () ==> ()
placePiece() ==
(
    s.placePiece(0,0,<CAPTAIN>,<BLUE>);

    -- six bombs, maximum
    s.placePiece(0,2,<BOMB>,<BLUE>);
    s.placePiece(1,2,<BOMB>,<BLUE>);
    s.placePiece(2,2,<BOMB>,<BLUE>);
    s.placePiece(3,2,<BOMB>,<BLUE>);
    s.placePiece(4,2,<BOMB>,<BLUE>);
    s.placePiece(1,3,<BOMB>,<BLUE>);

    -- Works because is the other player
    s.placePiece(5,9,<BOMB>,<RED>);


    s.placePiece(6,1,<SCOUT>,<BLUE>);
    s.placePiece(6,2,<SCOUT>,<BLUE>);
    s.placePiece(4,3,<SCOUT>,<BLUE>);
    s.placePiece(4,6,<SCOUT>,<RED>);


    s.placePiece(1,6,<MINER>,<RED>);


    s.placePiece(0,6,<SCOUT>,<RED>);
    s.placePiece(0,7,<SCOUT>,<RED>);

   --check freeSpace

        assertEqual(false,s.getBoard().freeSpace(
            mk_Board`Position(0,0)));
        assertEqual(false,s.getBoard().freeSpace(
            mk_Board`Position(0,2)));
);


private testHasPiece: () ==> ()
testHasPiece() ==
(
 assertEqual(true,s.getBoard().hasPiece(mk_Board`
     Position(0,0)));
 assertEqual(true,s.getBoard().hasPiece(mk_Board`
     Position(0,2)));

 --water
 assertEqual(false,s.getBoard().hasPiece(mk_Board`
     Position(3,4)));
```

```
);


  private possibleMove: () ==> ()
possibleMove() ==
(
  assertEqual(true,s.getBoard().movePossible(mk_Board`
      Position(0,0),mk_Board`Position(0,1),false));
  assertEqual(true,s.getBoard().movePossible(mk_Board`
      Position(1,3),mk_Board`Position(2,3),false));
  assertEqual(true,s.getBoard().movePossible(mk_Board`
      Position(1,3),mk_Board`Position(2,3),true));
  assertEqual(false,s.getBoard().movePossible(mk_Board`
      Position(50,0),mk_Board`Position(50,50),false));
  assertEqual(true,s.getBoard().movePossible(mk_Board`
      Position(1,0),mk_Board`Position(9,0),true));

  assertEqual(true,s.getBoard().movePossible(mk_Board`
      Position(1,0),mk_Board`Position(9,0),true));

    -- with range
    assertEqual(false,s.getBoard().movePossible(
        mk_Board`Position(0,6),mk_Board`Position(0,9),
        true));
    assertEqual(false,s.getBoard().movePossible(
        mk_Board`Position(0,6),mk_Board`Position(2,6),
        true));

);


private makeMove: () ==> ()
 makeMove() ==
(

  assertEqual(true,s.getBoard().makeMove(mk_Board`
      Position(0,0),mk_Board`Position(1,0),<BLUE>));
  assertEqual(false,s.getBoard().makeMove(mk_Board`
      Position(0,6),mk_Board`Position(0,7),<RED>));
  assertEqual(false,s.getBoard().makeMove(mk_Board`
      Position(6,1),mk_Board`Position(6,2),<BLUE>));


  --test scout range
  assertEqual(true,s.getBoard().makeMove(mk_Board`
      Position(4,6),mk_Board`Position(4,3),<RED>));
  assertEqual(false,s.getBoard().hasPiece(mk_Board`
      Position(4,6)));
  assertEqual(false,s.getBoard().hasPiece(mk_Board`
      Position(4,3)));

);


private testInteraction: () ==> ()
 testInteraction() ==
(
```

```
--test interaction
s.placePiece(9,6,<LIEUTENANT>,<RED>);
s.placePiece(9,3,<MAJOR>,<BLUE>);


assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(9,6),mk_Board`Position(9,5),<RED>));
assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(9,5),mk_Board`Position(9,4),<RED>));
assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(9,4),mk_Board`Position(9,3),<RED>));

assertEqual(<BLUE>, s.getBoard().getPiece(mk_Board`
    Position(9,3)).getColor());

--test another interaction

s.placePiece(8,6,<MAJOR>,<RED>);
s.placePiece(8,3,<LIEUTENANT>,<BLUE>);


assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(8,6),mk_Board`Position(8,5),<RED>));
assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(8,5),mk_Board`Position(8,4),<RED>));
assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(8,4),mk_Board`Position(8,3),<RED>));

assertEqual(<RED>, s.getBoard().getPiece(mk_Board`
    Position(8,3)).getColor());

--test miner


assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(1,6),mk_Board`Position(1,5),<RED>));
assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(1,5),mk_Board`Position(1,4),<RED>));
assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(1,4),mk_Board`Position(1,3),<RED>));

assertEqual(<RED>, s.getBoard().getPiece(mk_Board`
    Position(1,3)).getColor());

--test spy

s.placePiece(4,6,<SPY>,<RED>);
s.placePiece(4,3,<MARSHALL>,<BLUE>);

--impossible move

assertEqual(false,s.getBoard().makeMove(mk_Board`
    Position(4,6),mk_Board`Position(4,8),<RED>));

assertEqual(true,s.getBoard().makeMove(mk_Board`
    Position(4,6),mk_Board`Position(4,5),<RED>));
```

```
  assertEqual(true,s.getBoard().makeMove(mk_Board`
      Position(4,5),mk_Board`Position(4,4),<RED>));
  assertEqual(true,s.getBoard().makeMove(mk_Board`
      Position(4,4),mk_Board`Position(4,3),<RED>));

  assertEqual(<RED>, s.getBoard().getPiece(mk_Board`
      Position(4,3)).getColor());
);


private testIfGameOver : () ==> ()
testIfGameOver() == (
  --game not started
  assertEqual(nil,s.getBoard().checkGameOver());
  s.placePiece(3,7,<FLAG>,<RED>);
  --red won
  assertEqual(<RED>,s.getBoard().checkGameOver());
  s.placePiece(3,3,<FLAG>,<BLUE>);
  --during game
  assertEqual(nil,s.getBoard().checkGameOver());
  s.getBoard().getCell(mk_Board`Position(3,7)).
      removePiece();
  --blue won
  assertEqual(<BLUE>,s.getBoard().checkGameOver());
);


private StrategoTurns : () ==> ()
StrategoTurns() == (
 --Play returning false
 assertEqual(false,s.makePlay(6,1,6,2));

 --Blue Turn
 assertEqual(0,s.getTurn());
 --Blue piece
 assertEqual(true,s.makePlay(6,1,5,1));

 --Now red Turn
 assertEqual(1,s.getTurn());

 --Play returning false
 assertEqual(false,s.makePlay(1,3,3,3));

 --Blue Piece, must fails precondition
 --s.makePlay(5,1,6,1);
 --Red Piece, Play made
 assertEqual(true,s.makePlay(1,3,1,4));
 assertEqual(0,s.getTurn());
);


private repeatPlays : () ==> ()
repeatPlays() == (


  assertEqual(true,s.getBoard().makeMove(mk_Board`
      Position(4,3),mk_Board`Position(4,4),<RED>));
```

```
      assertEqual(true,s.getBoard().makeMove(mk_Board`
          Position(4,4),mk_Board`Position(4,3),<RED>));
      assertEqual(true,s.getBoard().makeMove(mk_Board`
          Position(4,3),mk_Board`Position(4,4),<RED>));

      --After 3 plays to the same position it returnes
          false
      assertEqual(false,s.getBoard().makeMove(mk_Board`
          Position(4,4),mk_Board`Position(4,3),<RED>));
    );



    -- Invalid Inputs, Pre Condition Failing, one at a time


    private placePieceOnWater : () ==> ()
    placePieceOnWater() == s.placePiece(4,5,<BOMB>,<BLUE>);


    private placePieceOutOfRegion : () ==> ()
    placePieceOutOfRegion() == s.placePiece(9,9,<BOMB>,<
        BLUE>);


    private excessiveTypePieces : () ==> ()
    excessiveTypePieces() == (
    -- six bombs, maximum
    s.placePiece(0,2,<BOMB>,<BLUE>);
    s.placePiece(1,2,<BOMB>,<BLUE>);
    s.placePiece(2,2,<BOMB>,<BLUE>);
    s.placePiece(3,2,<BOMB>,<BLUE>);
    s.placePiece(4,2,<BOMB>,<BLUE>);
  s.placePiece(1,3,<BOMB>,<BLUE>);

    s.placePiece(6,2,<BOMB>,<BLUE>);
    );


    public static main: () ==> ()
    main() ==
    (
            dcl test : BoardTest := new BoardTest();

            --Test Spaces
            test.testValidCoords();
            test.testFreeSpace();

            --Test Pieces
            test.placePiece();
            test.testHasPiece();

            --Tests Moves
            test.possibleMove();
            test.makeMove();
            test.testInteraction();
```

```
                        --Test Game
                        test.testIfGameOver();
                        test.StrategoTurns();
                        test.repeatPlays();
                );
end BoardTest
```

2 TestCase

```
class TestCase
/*
  Superclass for test classes, simpler but more practical than VDMUnit`
      TestCase.
  For proper use, you have to do: New -> Add VDM Library -> IO.
  JPF, FEUP, MFES, 2014/15.
*/

operations

 -- Simulates assertion checking by reducing it to pre-condition
     checking.
 -- If 'arg' does not hold, a pre-condition violation will be signaled.

 protected assertTrue: bool ==> ()
 assertTrue(arg) ==
  return
 pre arg;

 -- Simulates assertion checking by reducing it to post-condition
     checking.
 -- If values are not equal, prints a message in the console and
     generates
 -- a post-conditions violation.

 protected assertEqual: ? * ? ==> ()
 assertEqual(expected, actual) ==
  if expected <> actual then (
     IO`print("Actual value (");
     IO`print(actual);
     IO`print(") different from expected (");
     IO`print(expected);
     IO`println(")\n")
  )
 post expected = actual

end TestCase
```