Jami Biddle

HW3 Testing Document

It is not required to test the inherited, non-overridden tests, therefore I did not.

**Part 1: ChessPiece**

1. Getter methods can only really be tested with one thing, so it will simply make sure it returns what it should from the constructor. These getters include:
   1. ChessGame.Side getSide()
   2. String getLabel()
   3. Object getIcon()
   4. ChessBoard getChessBoard()
   5. int getColumn()
   6. int getRow()
2. Set Location:
   1. Change row & col to be different from the constructor.
3. isLegalMove(int toRow, int toColumn)
   1. should be tested for one with a legal non capture move, one with a legal capture move, one with both, and one with neither, for false.

#### **Part 2: The individual chess pieces.**

**Individual Piece Tests:**

1. **KnightPiece**:
   1. isLegalNonCapture move should be tested for:
      1. a non legal move (against the rules of the piece)
      2. A capture move (returns false)
      3. A legal move to an empty space
         1. One for all 8 possible spots; maybe unnecessary, but I want to test them.
   2. Is LegalCaptureMove
      1. a non legal move (against the rules of the piece)
      2. A capture move (returns true)
      3. A legal move to an empty space, so it returns false
      4. A ‘legal’ move that had the same side piece, so it should be false.
   3. isLegalMove
      1. A legal non-capture move
      2. A legal capture move
      3. An illegal move
   4. Make Move
      1. Make a legal move
      2. Make an illegal move
   5. MoveDone
      1. GetTurn() should now give back the side of the pice on which it was called
      2. GetFirstMove() should now equal true
2. **RookPiece:**
   1. isLegalNonCapture move should be tested for:
      1. a non legal move (against the rules of the piece)
      2. A capture move (returns false)
      3. The following, for the loop that checks to make sure there are no spaces in between:
         1. (none) A legal move to an empty space
         2. (one) A ‘legal’ move, but it has a piece in between
         3. (many) A ‘legal’ move, but is has a few pieces in between
         4. (first) A ‘legal’ move, but there is a piece in the way at the first space in between
         5. (Middle) A ‘legal’ move, but there is a piece in the way at the first space in between
         6. (Last) A ‘legal’ move, but there is a piece between at the space right before the intended space.
      4. The following legal moves
         1. Upwards
         2. Downwards
         3. To the left
         4. To the right
   2. Is LegalCaptureMove
      1. a non legal move (against the rules of the piece)
      2. A legal capture move (returns true)
         1. Upwards
         2. Downwards
         3. To the left
         4. To the right
      3. A legal move to an empty space, so it returns false
      4. A ‘legal’ move that had the same side piece, so it should be false.
      5. The following, for the loop that checks to make sure there are no spaces in between:
         1. (none) A legal move to an empty space
         2. (one) A ‘legal’ move, but it has a piece in between
         3. (many) A ‘legal’ move, but is has a few pieces in between
         4. (first) A ‘legal’ move, but there is a piece in the way at the first space in between
         5. (Middle) A ‘legal’ move, but there is a piece in the way at the first space in between
         6. (Last) A ‘legal’ move, but there is a piece between at the space right before the intended space.
   3. isLegalMove
      1. A legal non-capture move
      2. A legal capture move
      3. An illegal move
   4. Make Move
      1. Make a legal move
      2. Make an illegal move
   5. MoveDone
      1. GetTurn() should now give back the side of the pice on which it was called
      2. GetFirstMove() should now equal true
3. **BishopPiece**:
   1. isLegalNonCapture move should be tested for:
      1. a non legal move (non-diagonal)
      2. A capture move (returns false)
      3. The following, for the loop that checks to make sure there are no spaces in between:
         1. (None) A legal move to an empty space
         2. (One) A ‘legal’ move, but it has a piece in between
         3. (Many) A ‘legal’ move, but is has a few pieces in between
         4. (First) A ‘legal’ move, but there is a piece in the way at the first space in between
         5. (Middle) A ‘legal’ move, but there is a piece in the way at the first space in between
         6. (Last) A ‘legal’ move, but there is a piece between at the space right before the intended space.
      4. The following legal moves
         1. One to the top right
         2. One to the bottom right
         3. One to the bottom left
         4. One to the top left
   2. Is LegalCaptureMove
      1. a non legal move (non-diagonal)
      2. A capture move (returns true)
         1. One to the top right
         2. One to the bottom right
         3. One to the bottom left
         4. One to the top left
      3. A legal move to an empty space, so it returns false
      4. A ‘legal’ move that had the same side piece, so it should be false.
      5. The following, for the loop that checks to make sure there are no spaces in between:
         1. (none) A legal move to an empty space
         2. (one) A ‘legal’ move, but it has a piece in between
         3. (many) A ‘legal’ move, but is has a few pieces in between
         4. (first) A ‘legal’ move, but there is a piece in the way at the first space in between
         5. (Middle) A ‘legal’ move, but there is a piece in the way at the first space in between
         6. (Last) A ‘legal’ move, but there is a piece between at the space right before the intended space.
   3. isLegalMove
      1. A legal non-capture move
      2. A legal capture move
      3. An illegal move
   4. Make Move
      1. Make a legal move
      2. Make an illegal move
   5. MoveDone
      1. GetTurn() should now give back the side of the pice on which it was called
      2. GetFirstMove() should now equal true
4. **QueenPiece:**
   1. isLegalNonCapture move should be tested for:
      1. a non legal move (i.e. a knight move, so not directly diagonal or straight)
      2. A capture move (returns false)
      3. The following, for the loop that checks to make sure there are no spaces in between:
         1. (None) A legal move to an empty space
         2. (One) A ‘legal’ move, but it has a piece in between
         3. (Many) A ‘legal’ move, but is has a few pieces in between
         4. (First) A ‘legal’ move, but there is a piece in the way at the first space in between
         5. (Middle) A ‘legal’ move, but there is a piece in the way at the first space in between
         6. (Last) A ‘legal’ move, but there is a piece between at the space right before the intended space.
      4. The following legal moves
         1. One to the top right
         2. One to the bottom right
         3. One to the bottom left
         4. One to the top left
         5. Upwards
         6. Downwards
         7. To the left
         8. To the right
   2. Is LegalCaptureMove
      1. a non legal move (i.e. a knight move)
      2. A capture move (returns true)
         1. One to the top right
         2. One to the bottom right
         3. One to the bottom left
         4. One to the top left
         5. Upwards
         6. Downwards
         7. To the left
         8. To the right
      3. A legal move to an empty space, so it returns false
      4. A ‘legal’ move that had the same side piece, so it should be false.
      5. The following, for the loop that checks to make sure there are no spaces in between:
         1. (none) A legal move to an empty space
         2. (one) A ‘legal’ move, but it has a piece in between
         3. (many) A ‘legal’ move, but is has a few pieces in between
         4. (first) A ‘legal’ move, but there is a piece in the way at the first space in between
         5. (Middle) A ‘legal’ move, but there is a piece in the way at the first space in between
         6. (Last) A ‘legal’ move, but there is a piece between at the space right before the intended space.
   3. isLegalMove
      1. A legal non-capture move
      2. A legal capture move
      3. An illegal move
   4. Make Move
      1. Make a legal move
      2. Make an illegal move
   5. MoveDone
      1. GetTurn() should now give back the side of the pice on which it was called
      2. GetFirstMove() should now equal true
5. **KingPiece:** 
   1. isLegalNonCapture move should be tested for:
      1. a non legal move (any move more than one space away (except castling, see the last section of KingPiece))
      2. A capture move (returns false)
      3. The following legal moves
         1. One to the top right
         2. One to the bottom right
         3. One to the bottom left
         4. One to the top left
         5. Upwards
         6. Downwards
         7. To the left
         8. To the right
   2. Is LegalCaptureMove
      1. a non legal move (i.e. a knight move)
      2. A capture move (returns true)
         1. One to the top right
         2. One to the bottom right
         3. One to the bottom left
         4. One to the top left
         5. One Upwards
         6. One Downwards
         7. One To the left
         8. One To the right
      3. A legal move to an empty space, so it returns false
      4. A ‘legal’ move that had the same side piece, so it should be false.
   3. isLegalMove
      1. A legal non-capture move
      2. A legal capture move
      3. An illegal move
   4. Make Move
      1. Make a legal move
      2. Make an illegal move
   5. MoveDone
      1. GetTurn() should now give back the side of the pice on which it was called
      2. GetFirstMove() should now equal true
   6. Castling
      1. I did this visually. I initialized
6. **PawnPiece**
   1. isLegalNonCapture move should be tested for:
      1. a non legal move (any move more than one space away (except castling, see the last section of KingPiece))
      2. A capture move (returns false)
      3. The following
         1. If North:
            1. First move down by two
            2. move down by one
            3. Non-first move by two (false)
            4. Diagonal move left (false)
            5. Diagonal move right (false)
         2. If South:
            1. First move up by two
            2. move up by one
            3. Non-first move by two (false)
            4. Diagonal move left (false)
            5. Diagonal move right (false)
         3. If East:
            1. First move left by two
            2. move left by one
            3. Non-first move by two (false)
            4. Diagonal move up (false)
            5. Diagonal move down (false)
         4. If west:
            1. First move right by two
            2. move right by one
            3. Non-first move by two (false)
            4. Diagonal move up(false)
            5. Diagonal move down(false)
   2. Is LegalCaptureMove
      1. a non legal move (i.e. a knight move)
      2. A capture move (returns true) for:
         1. If North:
            1. Diagonal move left
            2. Diagonal move right
         2. If South:
            1. Diagonal move left
            2. Diagonal move right
         3. If East:
            1. Diagonal move up
            2. Diagonal move down
         4. If west:
            1. Diagonal move up
            2. Diagonal move down
      3. A legal move to an empty space, so it returns false
      4. A ‘legal’ move that had the same side piece, so it should be false.
   3. isLegalMove
      1. A legal non-capture move
      2. A legal capture move
      3. An illegal move
   4. Make Move
      1. Make a legal move
      2. Make an illegal move
   5. MoveDone
      1. GetTurn() should now give back the side of the pice on which it was called
      2. GetFirstMove() should now equal true

**Part 3: European Chess**

1. **EuropeanChess**
   1. GetTurn()
      1. Set using SetTurn(), then get
   2. SetTurn()
      1. See the getter
   3. Main method
      1. If it is correct, the pieces i input should all pop up on the board, working.
         1. It does (It’s a miracle or something)