# **Testing**

## Black Box

(Requirement #: Explanation)

**1.1/1.2**: Player can choose to start a new game using the Start Button, or they can load a previously stored game using the Load Game button, which uses the saveData text file that stores saved games.

Input: Pressed “Start Game”, then selected “Regular”, “Player 1”, and then “Hard”.
Expected Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.
Actual Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.

Input: Pressed “Start Game”, then selected “Regular”, “Player 1”, and then “Medium”.
Expected Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.
Actual Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.

Input: Pressed “Start Game”, then selected “Regular”, “Player 1”, and then “Easy”.
Expected Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.
Actual Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.

Input: Pressed “Start Game”, then selected “Regular”, “Player 1”, and then “Very Easy”.
Expected Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.
Actual Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.

Input: Pressed “Start Game”, then selected “Regular”, “Player 2”, and then “Hard”.
Expected Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.
Actual Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.

Input: Pressed “Start Game”, then selected “Regular”, “Player 2”, and then “Medium”.
Expected Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.
Actual Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.

Input: Pressed “Start Game”, then selected “Regular”, “Player 2”, and then “Easy”.
Expected Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.
Actual Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.

Input: Pressed “Start Game”, then selected “Regular”, “Player 2”, and then “Very Easy”.
Expected Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.
Actual Output: A regular game set up appearing with 12 black piece and 12 red pieces properly placed.

Input: Pressed “Load Game”, then selected “Game” (previously saved game with only one black piece placed), then pressed “Load”, “Player 1”, “Very Easy”.
Expected Output: Game appears with only one black piece placed.
Actual Output: Game appears with only one black piece placed.

Input: Pressed “Load Game”, then selected “Game” (previously saved game with only one black piece placed), then pressed “Load”, “Player 2”, “Easy”.
Expected Output: Game appears with only one black piece placed.
Actual Output: Game appears with only one black piece placed.

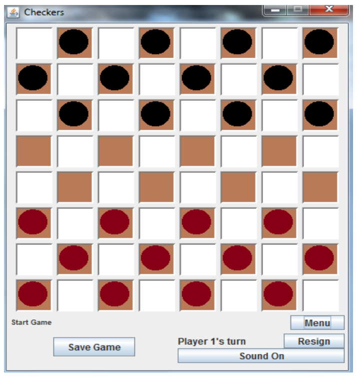
**1.3:** By selecting Two Players the user will deactivate the AI, allowing both moves to be made by human players.

Input: Selected “Two Players” check box, pressed “Start Game”, “Regular”.
Expected Output: Game appears with regular setup, after moving a red piece the user can move a black piece.
Actual Output:Game appears with regular setup, after moving a red piece the user can move a black piece.
Input: Selected “Two Players” check box, pressed “Start Game”, “Custom”.

Expected Output: Empty board appears, pieces are place-able, once setup is complete red pieces are moveable, then black piece can be moved by user.
Actual Output: Empty board appears, pieces are place-able, once setup is complete red pieces are moveable, then black piece can be moved by user.

**1.4/1.5**: **:** The aiMove method uses a combination of methods from the Checkers class to determine which move the computer will make, depending on the users requested AI difficulty.

(Starting with regular setup as player 1)
(square x,y represents square at row x and column y)

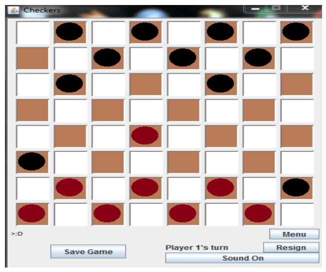


Input: Pressed the square 5,0, then pressed square 4,1.
Expected Output: Black piece is moved.
Actual Output: Black piece is moved.

Input: Pressed the square 5,2, then pressed square 4,3.
Expected Output: Black piece jumps red piece.
Actual Output: Black piece jumps red piece.

Input: Pressed the square 5,4, then pressed square 4,5.
Expected Output: Black piece is moved.
Actual Output: Black piece is moved.

Input: Pressed the square 4,6, then pressed square 3,4.
Expected Output: Black piece is moved.



Actual Output: Black piece is moved.

Input: Pressed the square 6,7 then pressed square 5,6.
Expected Output: Black piece is moved.
Actual Output: Black piece is moved.

**1.6:** The Save button can be used to save games, which uses the saveMenu class and saveData text file.
(refer to pages 7 and 10 for more details)

Input: Pressed save game, entered name “Game Save”, pressed “OK”
Expected Output: Save window disappears, saveData.txt has game written within
Actual Output: Save window disappears, saveData.txt has game written within

**1.7:** The switchPlayer button has been reused during game play as the Resign button, since switchPlayer is only a function needed during custom setup, which is a time when Resign is not needed.

Input: Pressed “Switch Player”
Expected Output: Player changed from 1 to 2.
Actual Output: Player changed from 1 to 2.

Input: Pressed “Switch Player”
Expected Output: Player changed from 2 to 1.
Actual Output: Player changed from 2 to 1.

Input: Pressed “Resign”
Expected Output: Message displays “Player 1 wins!”
Actual Output: Message displays “Player 1 wins!”

Input: Pressed “Resign”
Expected Output: Message displays “Player 2 wins!”
Actual Output: Message displays “Player 2 wins!”

**1.8:** The SquareClicked method notifies the user whenever they click an illegal square, when they click an empty square before selecting a piece, or when they select a piece that is not their own. This is done by updating the global string “message” which is displayed on the GUI gameboard after the actionPerformed method is complete (updating is the last thing the method does).

(Player 1s turn with no pieces selected)

Input: Pressed a white square.
Expected Output: Message displays “You clicked an illegal square”.
Actual Output: Message displays “You clicked an illegal square”.

Input: Pressed a brown square.
Expected Output: Message displays “Please select a square first”.
Actual Output: Message displays “Please select a square first.”

Input: Pressed a black piece.
Expected Output: Message displays “Player 1 cannot select Player 2's pieces.”
Actual Output: Message displays “Player 1 cannot select Player 2's pieces.”

(Player 2s turn with no pieces selected)

Input: Pressed a black piece.
Expected Output: Message displays “Player 2 cannot select Player 1's pieces.”
Actual Output: Message displays “Player 2 cannot select Player 1's pieces.”

**White Box**

**removeLegalsFor(int row, int col)**:void: removes legal moves for the given piece.

Input: (2,1)
Expected Output: Piece at row 2, column 1 has no legal moves.
Actual Output: Piece at row 2, column 1 has no legal moves.

Input: (2,3)
Expected Output: Piece at row 2, column 3 has no legal moves.
Actual Output: Piece at row 2, column 3 has no legal moves.

Input: (2,5)
Expected Output: Piece at row 2, column 5 has no legal moves.
Actual Output: Piece at row 2, column 5 has no legal moves.

Input: (2,7)
Expected Output: Piece at row 2, column 7 has no legal moves.
Actual Output: Piece at row 2, column 7 has no legal moves.

**setLegalJumpsFor(int[][] board, int player, int row, int col)**:void: sets legal jumps for the given piece.

Input: (2,1)
Expected Output: Piece at row 2, column 1 has legal moves.
Actual Output: Piece at row 2, column 1 has legal moves.

Input: (2,3)
Expected Output: Piece at row 2, column 3 has  legal moves.
Actual Output: Piece at row 2, column 3 has legal moves.

Input: (2,5)
Expected Output: Piece at row 2, column 5 has legal moves.
Actual Output: Piece at row 2, column 5 has legal moves.

Input: (2,7)
Expected Output: Piece at row 2, column 7 has legal moves.
Actual Output: Piece at row 2, column 7 has legal moves.

**removeLegalJumpsFor(int,board[][], int row, int col)**:void: removes legal jumps for the given piece.

Input: (2,1)
Expected Output: Piece at row 2, column 1 has no legal jumps.
Actual Output: Piece at row 2, column 1 has no legal jumps.

Input: (2,3)
Expected Output: Piece at row 2, column 3 has no legal jumps.
Actual Output: Piece at row 2, column 3 has no legal jumps.

Input: (2,5)
Expected Output: Piece at row 2, column 5 has no legal jumps.
Actual Output: Piece at row 2, column 5 has no legal jumps.

Input: (2,7)
Expected Output: Piece at row 2, column 7 has no legal jumps.
Actual Output: Piece at row 2, column 7 has no legal jumps.

**anyLegalJumps(int[][] board, int player)**:boolean: returns true if the given player can make a legal jump anywhere on the board, otherwise returns false.

Input: (regular setup board, 1)
Expected Output: true
Actual Output: true

Input: (regular setup board, 2)
Expected Output: true
Actual Output: true

Input: (custom setup board 1 piece for each player, 1)
Expected Output: true
Actual Output: true

Input: (custom setup board 1 piece for each player, 2)
Expected Output: true
Actual Output: true

**canJump(int[][] board, int player, int row, int col)**:boolean: returns true if the given piece can jump, otherwise returns false.

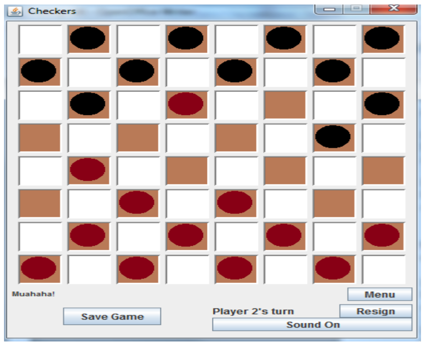
Input: (regular setup board, 1, 1, 0)
Expected Output: false
Actual Output: false

Input: (regular setup board, 1, 0, 1)
Expected Output: false
Actual Output: false

Input: (regular setup board, 1, 0,5)
Expected Output: false
Actual Output: false

Input: (Game 1, 2, 1, 2)
Expected Output: true
Actual Output: true

Input: (Game 1, 2, 1, 4)
Expected Output: true
Actual Output: true



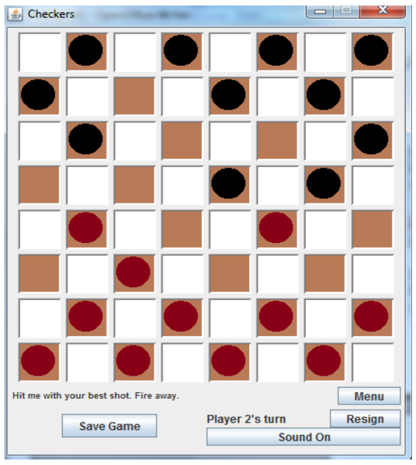
Muahaha!
Player 2's turn

*Game 1*

**jumpMove(int[][] board, int row, int col)**:void: handles the case where the selected piece is to make a jump.

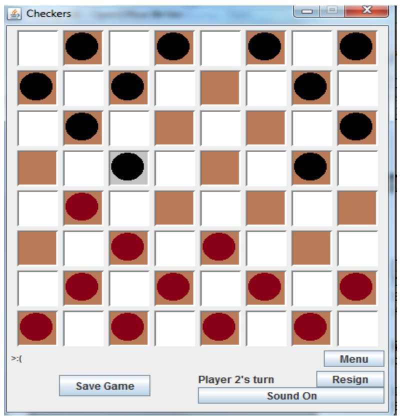
Input: (Game 1, 2, 1, 2)
Expected Output: Game 2
Actual Output: Game 2

Input: (Game 1, 2, 1, 4)
Expected Output: Game 3
Actual Output: Game 3



Hit me with your best shot. Fire away.

*Game 2*



>:(

*Game 3*

**ratio(int[][] board, int player)**:double:  method returns the ratio of player pieces to enemy pieces.

Input: (Game 2, 1)
Expected Output: 1
Actual Output: 1

Input: (Game 2, 2)
Expected Output: 1
Actual Output: 1

Input: (Game 1, 1)
Expected Output: 1.09
Actual Output: 1.09

Input: (Game 1, 2)
Expected Output: 0.92
Actual Output: 0.92

**getNumPieces(int[][] board, int player)**:int: method returns the number of pieces on the board belonging to player.

Input: (Game 2, 1)
Expected Output: 11
Actual Output: 11

Input: (Game 2, 2)
Expected Output: 11
Actual Output: 11

Input: (Game 3, 1)
Expected Output: 11
Actual Output: 11

Input: (Game 3, 2)
Expected Output: 11
Actual Output: 11

Input: (Game 1, 1)
Expected Output: 12
Actual Output: 12

**sumAllPieces(int[][] board)**:int:  method returns the total number of pieces on the board.

Input: (Game 1)
Expected Output: 23
Actual Output: 23

Input: (Game 2)
Expected Output: 22
Actual Output: 22

Input: (Game 3)
Expected Output: 22
Actual Output: 22

**copy(int[][] move)**:int[][]: makes a copy of the given 2D array.

Input: {{4,2},{2,3}}
Expected Output:{{4,2},{2,3}}
Actual Output:{{4,2},{2,3}}

Input: {{5,3},{4,1}}
Expected Output:{{5,3},{4,1}}
Actual Output:{{5,3},{4,1}}

Input: {{3,6},{1,1}}
Expected Output:{{3,6},{1,1}}
Actual Output:{{3,6},{1,1}}

Input: {{7,3},{5,7}}
Expected Output:{{7,3},{5,7}}
Actual Output:{{7,3},{5,7}}

Input: {{8,6},{0,3}}
Expected Output:{{8,6},{0,3}}
Actual Output:{{8,6},{0,3}}

Input: {{3,3},{1,9}}
Expected Output:{{3,3},{1,9}}
Actual Output:{{3,3},{1,9}}

Input: {{0,0},{4,2}}
Expected Output:{{0,0},{4,2}}
Actual Output:{{0,0},{4,2}}

Input: {{1,5},{2,8}}
Expected Output:{{1,5},{2,8}}
Actual Output:{{1,5},{2,8}}

**isPiece(int[][] board, int player, int row, int col)**:boolean: returns true if the given square is a piece, otherwise returns false.

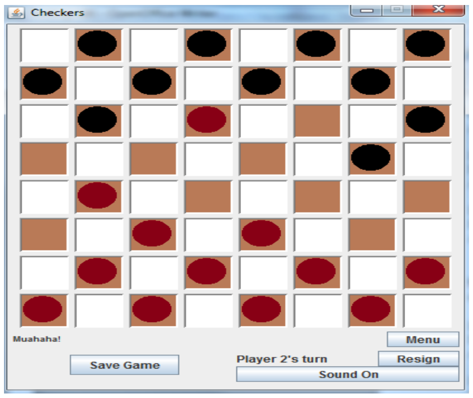
Input: (Game 1, 1, 7, 0)
Expected Output: true
Actual Output: true

Input: (Game 1, 1, 5, 0)
Expected Output: false
Actual Output: false

Input: (Game 1, 1, 7, 2)
Expected Output: true
Actual Output: true

Input: (Game 1, 1, 1, 4)
Expected Output: false
Actual Output: false

Input: (Game 1, 1, 7, 6)
Expected Output: true
Actual Output: true



*Game 1*

**areEqual(int[][] board1, int[][] board2):**boolean: returns true if the two arrays are equal, otherwise returns false.

Input: (Game 1, Game 2)
Expected Output: false
Actual Output: false

Input: (Game 2, Game 2)
Expected Output: true
Actual Output: true

Input: (Game 3, Game 2)
Expected Output: false
Actual Output: false

Input: (Game 1, Game 3)
Expected Output: false
Actual Output: false

Input: (Game 1, Game 1)
Expected Output: true
Actual Output: true

**isLegal(int i)**:boolean: method returns true if the given number correlates to some legal move value and false otherwise.

Input: 11
Expected Output: true
Actual Output: true

Input:12
Expected Output: true
Actual Output:

Input:13
Expected Output: true
Actual Output: true

Input: 14
Expected Output: true
Actual Output: true

```
Input:31
Expected Output: true
Actual Output: true

Input:32
Expected Output: true
Actual Output: true

Input:33
Expected Output: true
Actual Output: true

Input:34
Expected Output: true
Actual Output: true

Input:1
Expected Output: false
Actual Output: false

Input:2
Expected Output: false
Actual Output: false

Input:3
Expected Output: false
Actual Output: false

Input:4
Expected Output: false
Actual Output: false

Input:5
Expected Output: false
Actual Output: false

Input:6
Expected Output: false
Actual Output: false

Input:7
Expected Output: false
Actual Output: false

Input:8
Expected Output: false
Actual Output: false

Input:9
Expected Output: false
Actual Output: false

Input:10
```

```
Expected Output: false
Actual Output: false

Input:11
Expected Output: false
Actual Output: false

Input:12
Expected Output: false
Actual Output: false
```