Test Plan Result Report

Team 6

Jeremy Pasimio | Maigan Davey | Seiichi Nagai

Revision K
11/27/2017

Revision History

DATE	REV	AUTHOR	DESCRIPTION
11/18/2017	A	Seiichi	System
11/18/2017	B	Jeremy	Acceptance
11/18/2017	C	Maigan	Integration
11/20/2017	D	Jeremy	Unit
11/21/2017	E	Seiichi	Integration
11/21/2017	F	Jeremy	Unit
11/25/2017	G	Seiichi	Added Pass/Fail/Description Column
11/262017	H	Seiichi	System, Integration, Acceptance testing Added Bug/Enhancements tables
11/27/2017	I	Maigan	Unit Testing - Checkers class tests & CheckerMove class tests 39 & 40
11/27/2017	J	Jeremy	Unit Testing Completed - CheckerMove class tests 1-38
11/27/2017	K	Maigan	Full list of bugs p.47
11/28/2017	L	Seiichi	Revise System Test Cases

Contents

UNIT TESTING

CheckerMove.java:

TEST CASE 1: getIndex(int x, int y), lowest x, lowest y
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	int[] result = new int[2];		

2	result = getIndex(1,1)	result[0] = 0;
		result[1] = 0;
			Pass

TEST CASE 2: getIndex(int x, int y), lowest x, highest y
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	int[] result = new int[2];		

2	result = getIndex(1,399)	result[0] = 0;
		result[1] = 7;
			Pass

TEST CASE 3: getIndex(int x, int y), highest x, highest y
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	int[] result = new int[2];		

2	result = getIndex(399,399)	result[0] = 7;
		result[1] = 7;
			Pass

TEST CASE 4: getIndex(int x, int y), highest x, lowest y
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	int[] result = new int[2];		

2	result = getIndex(399,1)	result[0] = 7;
		result[1] = 0;
			Pass

TEST CASE 5: getIndex(int x, int y), exceed max x, any y 0-399
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	int[] result = new int[2];		

2	result = getIndex(400,1)	result[0] = 0;
		result[1] = 0;
			Pass

TEST CASE 6: getIndex(int x, int y), any x 0-399, exceed max y
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	int[] result = new int[2];		

2	result = getIndex(1,400)	result[0] = 0;
		result[1] = 0;
			Pass

TEST CASE 7: noMovesLeft, Red to move, No moves
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Pass method an int[][] representing a board with no available moves and a Checkers.redNormal.	True	Pass

TEST CASE 8: noMovesLeft, Red to move, Capture available
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Pass method an int[][] representing a board with a capture move available for red and a Checkers.redKing.	False	Pass

TEST CASE 9: noMovesLeft, Red to move, Walk available
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Pass method an int[][] representing a board with a walk move available for red and a Checkers.redNormal.	False	Pass

TEST CASE 10: noMovesLeft, Yellow to move, No moves
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Pass method an int[][] representing a board with no available moves and a Checkers.yellowKing.	True	Pass

TEST CASE 11: noMovesLeft, Yellow to move, Capture available
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Pass method an int[][] representing a board with a capture move available for red and a Checkers.yellowNormal.	False	Pass

TEST CASE 12: noMovesLeft, Yellow to move, Walk available
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Pass method an int[][] representing a board with a walk move available for red and a Checkers.yellowKing.	False	Pass

TEST CASE 13: ApplyMove(int[][] board,int srtI,int srtJ,int endI,int endJ), move not legal
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	int result =ApplyMove(board,srtI,srtJ,endI,endJ) Where board is the current board state,srtI and srtJ are the starting coordinates of the move, and endI and endJ are the end coordinates of an illegal move.	result==illegalMove	Pass

TEST CASE 14: ApplyMove(int[][] board,int srtI,int srtJ,int endI,int endJ), move legal
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	int result =ApplyMove(board,srtI,srtJ,endI,endJ) Where board is the current board state,srtI and srtJ are the starting coordinates of the move, and endI and endJ are the end coordinates of a legal move.	result==legalMove	Pass

TEST CASE 15: isMoveLegal(int[][] board,int srtI,int srtJ,int endI,int endJ,int turn), move not legal
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	int result = isWalkLegal(board,srtI,srtJ,endI,endJ,turn) Where board is the current board state,srtI and srtJ are the starting coordinates of the move,endI and endJ are the end coordinates of an illegal move, and turn is the color of the piece trying to move.	result==illegalMove	Pass

TEST CASE 16: isMoveLegal(int[][] board,int srtI,int srtJ,int endI,int endJ,int turn), move legal
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	int result = isWalkLegal(board,srtI,srtJ,endI,endJ,turn) Where board is the current board state,srtI and srtJ are the starting coordinates of the move,endI and endJ are the end coordinates of a legal move, and turn is the color of the piece trying to move.	result==legalMove	Pass

TEST CASE 17: isWalkLegal(int[][] board,int srtI,int srtJ,int endI,int endJ), walk not legal
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	int result = isWalkLegal(board,srtI,srtJ,endI,endJ) Where board is the current board state,srtI and srtJ are the starting coordinates of the move,endI and endJ are the end coordinates of an illegal move.	result==illegalMove	Pass

TEST CASE 18: isWalkLegal(int[][] board,int srtI,int srtJ,int endI,int endJ), walk legal
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	int result = isWalkLegal(board,srtI,srtJ,endI,endJ) Where board is the current board state,srtI and srtJ are the starting coordinates of the move,endI and endJ are the end coordinates of a legal move.	result==legalMove	Pass

TEST CASE 19: isEmpty(int[][] board,int i, int j)
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	boolean result = isEmpty(board,i,j) Where board is the current board state and i and j are coordinates of an empty space.	result==true	Pass

TEST CASE 20: isEmpty(int[][] board,int i, int j)
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	boolean result = isEmpty(board,i,j) Where board is the current board state and i and j are coordinates of an occupied space (piece color and rank does not matter).	result==false	Pass

TEST CASE 21: colour(int piece), yellow piece
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	piece=yellowNormal ||yellowKing int result = colour(piece)	result==yellowNormal	Pass

TEST CASE 22: colour(int piece), red piece
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description

1	piece=redNormal ||redKing int result = colour(piece)	result==redNormal	Pass

TEST CASE 23:canCapture(int[][] board,int toMove), red to move, no capture available
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Pass method board state with no capture available and either redNormal or redKing	False	Pass

TEST CASE 24:canCapture(int[][] board,int toMove), red to move, capture available
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Pass method board state with at least one capture available and either redNormal or redKing	True	Pass

TEST CASE 25:canCapture(int[][] board,int toMove), yellow to move, no capture available
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**

1	Pass method board state with no capture available and either yellowNormal or yellowKing	False	Pass

TEST CASE 26:canCapture(int[][] board,int toMove), yellow to move, capture available
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Pass method board state with at least one capture available and either yellowNormal or yellowKing	True	Pass

TEST CASE 27:canCapture(int[][] board,int i, int j), no capture available
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Pass method board state with no capture available and coordinates (i,j) of a piece on the board.	False	Pass

TEST CASE 28:canCapture(int[][] board,int i, int j), capture available, coordinates of piece with no capture available
Method of Testing: Unit

Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Pass method board state with a capture available and coordinates (i,j) of a piece on the board that cannot capture.	False	Pass

TEST CASE 29:canCapture(int[][] board, int i, int j), capture available, coordinates of capturing piece
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Pass method board state with a capture available and coordinates (i,j) of a the piece that can capture.	True	Pass

TEST CASE 30:canWalk(int[][] board, int i, int j), non-capturing move available, coordinates of movable piece
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Pass method board state with a non-capturing move available and coordinates (i,j) of a the piece that can move.	True	Pass

TEST CASE 31:canWalk(int[][] board, int i, int j), non-capturing move available, coordinates of non-movable piece
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Pass method board state with a non-capturing move available and coordinates (i,j) of a the piece that cannot move.	False	Pass

TEST CASE 32: inRange(int i, int j)
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	boolean result = inRange(0,0)	result==True	Pass

TEST CASE 33: inRange(int i, int j)
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	boolean result = inRange(7,7)	result==True	Pass

TEST CASE 34: inRange(int i, int j)
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	boolean result = inRange(-1,7)	result==False	Pass

TEST CASE 35: inRange(int i, int j)
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	boolean result = inRange(8,7)	result==False	Pass

TEST CASE 36: inRange(int i, int j)
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	boolean result = inRange(7,-1)	result==False	Pass

TEST CASE 37: inRange(int i, int j)
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	boolean result = inRange(7,8)	result==False	Pass

TEST CASE 38: generateMoves(int[][] board,int turn)
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Initialize Vector moves for moves that are expected to be generated.		
2	Vector moves2 = generateMoves(board,turn) where board is the state of the board and turn is the color of the piece trying to move.		
3	Compare moves with moves2	moves==moves2	Pass

TEST CASE 39: moveComputer(int[][] board,int[] move)
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Initialize int[][] for initial board state and int[][] for expected final board state.		Pass
2	Pass method initial board state and int[] containing move details		Pass

3	Compare new board state with expected final board state.	New board state==expected final board state.	Pass

TEST CASE 40: forceCaptures(int[][] board, int[] move, Vector moves_list,int inc), capture available

Method of Testing: Unit

Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Pass method initial board state, list of available moves		Pass
2	Capture is available	No non-capture moves can be made	Pass

<u>Checkers.java:</u>

TEST CASE 1: undo(), single click
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Initialize board and preBoard3 to different board states		Pass
2	Run undo()		Pass
3	Compare board and preBoard3	board==preBoard3	Pass

TEST CASE 2: undo(), max undo clicks
Method of Testing: Unit
Precondition: none

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Initialize board, preBoard1, preBoard2, and preBoard3 to different board states		Pass
2	Run undo()		Pass
3	Run undo()		Fail: Does not allow multiple undos on the same play/in a row
4	Run undo()		Fail: Does not allow multiple undos on the same play/in a row
5	Compare board and preBoard3	board==preBoard1	Pass

TEST CASE 3: isPossibleSquare(int i,int j), 'odd' square
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	boolean result = isPossibleSquare(0,1)	result = true	Pass assertEquals(isPossibleSquare(0,1), true)
2	boolean result = isPossibleSquare(1,0)	result = true	Pass assertEquals(isPossibleSquare(1,0), true)
3	boolean result = isPossibleSquare(6,7)	result = true	Pass assertEquals(isPossibleSquare(6,7), true)
4	boolean result = isPossibleSquare(7,6)	result = true	Pass assertEquals(isPossibleSquare(7,6), true)

TEST CASE 4: isPossibleSquare(int i,int j), 'even' square
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	boolean result = isPossibleSquare(0,0)	result = false	Pass assertEquals(isPossibleSquare(0, 0), false)
2	boolean result = isPossibleSquare(1,1)	result = false	Pass assertEquals(isPossibleSquare(1, 1), false)
3	boolean result = isPossibleSquare(6,6)	result = false	Pass assertEquals(isPossibleSquare(6, 6), false)
4	boolean result = isPossibleSquare(7,7)	result = false	Pass assertEquals(isPossibleSquare(7, 7), false)

TEST CASE 5: showStatus(), run through all status messages
Method of Testing: Unit
Precondition: none

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	toMove=yellowNormal		Pass

2	Run showStatus()	msg.getText().equals(“Yellow to Move”)	Pass
3	toMove=redNormal		Pass
4	Run showStatus()	msg.getText().equals(“Red to Move”)	Pass
5	loser=redNormal		Pass
6	Run showStatus()	msg.getText().equals(“Yellow Wins!”)	Pass
7	loser=yellowNormal		Pass
8	Run showStatus()	msg.getText().equals(“Red Wins!”)	Pass

Unit Testing Bugs (Failures) and Enhancements

Test Case No.	**Step No.**	**Bugs**	**Enhancements**
2	1	Undo() fails when attempted to be applied more than once consecutively	Make an array that holds the last three positions, since three is the max of undos available. Once a new position is held, the oldest position can be erased.

INTEGRATION TESTING

TEST CASE 1: Test to verify module consisting of IntelliCheckers, StartPanel and CheckersFrame.

Method of Testing: Manual

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Run Intellicheckers Class	Start Panel/CheckersFrame are created and the window opens.	Pass
2	Verify StartPanel GUI objects present	IntelliCheckers creates a CheckersFrame object which instantiates a StartPanel object so the window should consist of all StartPanel gui objects.	Pass
3	Verify that Start button appears	Button that says “Start Game”	Pass

TEST CASE 2: Test to verify that Play Checkers window populates with all graphics, sounds, Labels and buttons.

Method of Testing: Manual

Precondition: Program running, start panel open

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Click Start Game button	Button click instantiates a new Checkers object which contains directions on creating the gui objects on the page.	Pass
2	Verify Checkers Board appears	8x8 black and gray checkers board appears in upper left hand corner	Pass

3	Verify Help and Volume buttons appear	Button with a ‘?’ and a button with a speaker graphic appear	Pass
4	Verify New Game and Undo buttons appear	Just below help/undo buttons, button titled “New Game” appears and just below that “Undo” button appears	Pass
5	Verify Difficulty Level selector and label appear	Under the Undo button, label is present that reads “Difficulty Level”. Just below that is a comboBox with Moderate level as default and Easy, Fairly Easy, Moderate, Bit Difficult and Tough as alternate difficulty levels	Pass
6	Verify 1 & 2 player modes available	Below Difficulty is a label that reads “Mode” and below that are two radio buttons labeled “1-player and 2-player”. Radio buttons present with default set to “1-player”.	Pass
7	Verify color choices available	Below Mode is “Colour” label with Radio buttons: “Red” and “Yellow”. Radio buttons present with default set to “Yellow”.	Pass

8	Verify directions present	Below checkers board should be a text area with a brief set of directions that show the current state of the game. Before new game starts: “Start a new game… Yellow is first” After a new game is started: “Yellow to move”. When in 2-player mode after yellow moves, then “Red to move”.	Pass
9	Verify checkers images loaded correctly	Below directions there should be four checkers icons that show the red and yellow normal and King pieces with corresponding labels that match the description of the pieces.	Pass

TEST CASE 3: Test to verify ‘Help’ class integration
Method of Testing: Manual
Precondition: “Start Game” button clicked on StartPanel

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Verify help window pops up	Click on the question mark icon button and a new window should pop up with complete directions on how to operate the game.	Fail: No gameplay directions present
		The Checkers class contains information on the sound played when the help button is clicked as well as	Pass

		the help icon and button. It also opens a new window.	

TEST CASE 4: Test to verify 'PlaySound' class integration
Method of Testing: Manual
Precondition: "Start Game" button clicked on StartPanel

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Run IntelliCheckers	'Start' sound should play automatically	Pass
2	Click Start Game	'Button' sound should play	Pass
3	Click any button on page	'Button sound plays	Pass

TEST CASE 5: Test to verify Module consisting of Checkers, CheckerMove and GameEngine through 1-player gameplay

Method of Testing: Manual

Precondition: ‘Start Game’ button clicked on StartPanel, ‘New Game’ Button clicked in Checkers. Sounds unmuted. 1-Player Selected. Yellow Color Selected.

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Verify computer opponent functionality	Player moves yellow piece. Computer opponent automatically moves red piece.	Pass
2	Verify PlaySound	When player or opponent moves piece, PlaySound should play	Pass
3	Verify piece selection and clickChecker sound	When the player selects a piece, the gray square (gameboard location) that the piece is on should be highlighted orange. Selecting a piece should play the clickChecker sound.	Pass
4	Verify double jump possible	When a double jump is possible, player clicks piece to execute the double jump, jumps the first piece, then clicks the final box to complete the double jump. (A greater string of jumps can be executed using the same method if moves are possible)	Pass

5	Verify computer opponent difficulty level	The computer opponent should become more or less difficult depending on the difficulty level selected.	Pass
6	Verify force jumps	When either player has an opportunity to jump an opponent's piece, they must take that opportunity.	Pass
7	Verify Red color selection functionality	Opponent clicks the red color button then starts a new game. The Computer should move first automatically.	Pass
8	Verify GameWin window pops up with sound	When either opponent wins the game, a window should pop up that reads "(winning color) Wins!" and the win sound should play.	Pass

TEST CASE 6: Test to verify 'GameWin' class integration
Method of Testing: Manual
Precondition: 'Start Game' button clicked on StartPanel, 'New Game' Button clicked in Checkers. Sounds unmuted. 1-Player Selected. Yellow Color Selected.

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Play through a whole game with either player winning		Pass
2	Verify GameWin window pops up with sound	When either opponent wins the game, a window should pop up that reads "(winning color) Wins!" and the win sound should play.	Pass

TEST CASE 7: Test to verify 2-player gameplay
Method of Testing: Manual
Precondition: ‘Start Game’ button clicked on StartPanel, ‘New Game’ Button clicked in Checkers. Sounds unmuted. 2-Player Selected.

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Verify 2-player mode functionality	When a game is started with 2-player button selected the yellow player starts first. Followed by the red player. Game play is the same as 1-player but the computer player is replaced by a human player.	Pass

Integration Testing Bugs (Failures) and Enhancements

Test Case No.	Step No.	Bugs	Enhancements
3	1	No gameplay directions present	Add gameplay directions to help window

SYSTEM TESTING

TEST CASE 1: Test to verify IntelliCheckers Game starts and StartPanel and CheckersFrame opens successfully

Method of Testing: Manual

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Run program by double clicking IntelliCheckers .jar file.	Start Panel/CheckersFrame opens.	Pass
2	Verify that start sound plays	Synthetic vocal opening sound plays	Pass
3	Verify that game title, source and authors appear.	Title: “CHECKERS” Source: “CS 3230 INTELLIGENT SYSTEMS” Authors: “A.M.H.H. ABEVKOON” “B.P.P. FERNANDO” “C.S.N.J. FERNANDO” “K.C.B. GAJASINGHE”	Pass
4	Verify that checkers graphic appears	Black diagonal checkerboard against grey background	Pass
5	Verify that Start button appears	Button that says “Start Game”	Pass

TEST CASE 2: Test to verify that Checkers window populates with all graphics, sounds, Labels and buttons.
Method of Testing: Manual
Precondition: Program running, start panel open

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Click Start Game button	Button sound plays Checkers Window Opens	Pass
2	Verify Checkers Board appears	8x8 black and gray checkers board appears in upper left hand corner	Pass
3	Verify Help and Volume buttons appear	Button with a '?' and a button with a speaker graphic appear	Pass
4	Verify New Game and Undo buttons appear	Just below help/undo buttons, button titled "New Game" appears and just below that "Undo" button appears	Pass
5	Verify Difficulty Level selector and label appear	Under the Undo button, label is present that reads "Difficulty Level". Just below that is a comboBox with Moderate level as default and Easy, Fairly Easy, Moderate, Bit Difficult and Tough as alternate difficulty levels	Pass
6	Verify 1 & 2 player modes available	Below Difficulty is a label that reads "Mode" and below that are two radio buttons labeled "1-player and 2-player".	Pass

		Radio buttons present with default set to “1-player”.	
7	Verify color choices available	Below Mode is “Colour” label with Radio buttons: “Red” and “Yellow”. Radio buttons present with default set to “Yellow”.	Pass
8	Verify directions present	Below checkers board should be a text area with a brief set of directions that show the current state of the game. Before new game starts: “Start a new game… Yellow is first” After a new game is started: “Yellow to move”. When in 2-player mode after yellow moves, then “Red to move”.	Pass
9	Verify checkers images loaded correctly	Below directions there should be four checkers icons that show the red and yellow normal and King pieces with corresponding labels that match the description of the pieces.	Pass

TEST CASE 3: Test to verify ‘Help’ button
Method of Testing: Manual
Precondition: “Start Game” button clicked on StartPanel

Step No.	Execution Steps	Expected Results	Pass / Fail / Description

1	Verify button highlights on mouse hover	Hover mouse over button and button should highlight a light blue	Pass
2	Verify help window pops up	Click on the question mark icon button and a new window should pop up with complete directions on how to operate the game.	Fail: pop up window does not contain information on how to play the game
3	Verify button sound	A click on the help button should play a button sound	Pass

TEST CASE 4: Test to verify 'Sound' button
Method of Testing: Manual
Precondition: "Start Game" button clicked on StartPanel

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Verify button highlights on mouse hover	Hover mouse over button with speaker icon and button should highlight a light blue	Pass
2	Verify button icon changes	Click on the button and the graphic should change to a muted-speaker icon	Pass
3	Verify button mute and sound	A first click on the sound button should mute all sounds so user will not hear anything. A second click will unmute sounds so a button sound will be played. Etc.etc. the cycle continues.	Pass

4	Verify all sounds muted	After muting sound, and sound button shows mute icon, click on any other button on the Checkers frame. No other button sounds will play when other buttons are clicked.	Pass

TEST CASE 5: Test to verify 'New Game' button
Method of Testing: Manual
Precondition: "Start Game" button clicked on StartPanel. Sounds unmuted

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Verify button highlights on mouse hover	Hover mouse over button with speaker icon and button should highlight a light blue	Pass
2	Verify button sound	A click on the New Game button should play a button sound	Pass
3	Verify checkers pieces appear when button clicked	Clicking the 'New Game' button should cause the board to fill with checkers pieces on gray spaces only with 12 pieces on each side.	Pass
4	Verify Undo button is unenabled	After clicking New Game and before moving a piece, 'Undo' button should be unenabled	Pass

5	Verify accurate directions when new game clicked	If the yellow radio button is selected, clicking 'New Game' should cause the Directions area to say "Yellow to move". If red radio button is selected, clicking New Game button should cause computer to move a yellow piece and the directions area will say "Red to move"	Pass
6	Verify board resets after New Game clicked	After clicking New Game and moving a piece, clicking New Game again should cause the board to reset.	Pass

TEST CASE 6: Test to verify 'Undo' button
Method of Testing: Manual
Precondition: 'Start Game' button clicked on StartPanel. 'New Game' Button clicked in Checkers. Sounds unmuted.

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Verify Undo button is enabled after moving a piece	Moving a checkers piece is the only way to enable the Undo button	Fail: Undo button only becomes available after two moves
2	Verify button sound	A click on the undo button should play a button sound	Pass

3	Verify undo button functionality	Clicking Undo after moving a checkers piece should set the game board back one move. If in 1-player mode, the most recent computer opponent move is also undone. If in two player mode, which ever player moved most recently, that movement is undone.	Pass
4	Verify difficulty level functionality	Undo should not become enabled when ‘Tough’ difficulty setting is selected	Pass

TEST CASE 7: Test to verify number of players
Method of Testing: Manual
Precondition: ‘Start Game’ button clicked on StartPanel, ‘New Game’ Button clicked in Checkers. Sounds unmuted.

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Verify that default is set to 1-player	When the Checkers screen opens, the radio button should automatically be set to 1-Player	Pass
2	Verify Difficulty level and color are visible	Difficulty level and color should automatically be visible on the Checkers screen because mode is set to 1-player by default.	Pass
3	Verify option sound	Clicking 1-player or 2-player should play an option sound	Pass

4	Verify 2-player makes difficulty and color invisible	Clicking the 2-player radio button should cause the difficulty label and combo box disappear as well as the color label and radio buttons.	Pass
5	Verify 1-player mode functionality	After clicking the 1-player radio button, and then clicking new game, The human player (yellow) moves and the computer opponent automatically moves	Pass
6	Verify 2-player mode functionality	After clicking 2-player mode and starting a new game, human player 1 (yellow) moves first, then human player 2 (red) moves second.	Pass

TEST CASE 8: Test to verify color selection
Method of Testing: Manual
Precondition: ‘Start Game’ button clicked on StartPanel, ‘New Game’ Button clicked in Checkers. Sounds unmuted.

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Verify that default color is set to ‘Yellow’	After starting a new game, the yellow radio button should be automatically selected.	Pass
2	Verify option sound	When clicking either the red or yellow radio button, the option sound should play.	Pass

3	Verify color functionality	When yellow is selected and a new 1-player game is started, the human player is yellow, so they start. When red is selected and a new 1-player game is started the computer opponent is yellow so they start. *Yellow ALWAYS moves first	Pass

TEST CASE 9: Test to verify 1-player gameplay
Method of Testing: Manual
Precondition: ‘Start Game’ button clicked on StartPanel, ‘New Game’ Button clicked in Checkers. Sounds unmuted. 1-Player Selected. Yellow Color Selected.

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Verify computer opponent functionality	Player moves yellow piece. Computer opponent automatically moves red piece.	Pass
2	Verify PlaySound	When player or opponent moves piece, PlaySound should play	Pass
3	Verify piece selection and clickChecker sound	When the player selects a piece, the gray square (gameboard location) that the piece is on should be highlighted orange. Selecting a piece should play the clickChecker sound.	Pass

4	Verify double jump possible	When a double jump is possible, player clicks piece to execute the double jump, jumps the first piece, then clicks the final box to complete the double jump. (A greater string of jumps can be executed using the same method if moves are possible)	Pass
5	Verify computer opponent difficulty level	The computer opponent should become more or less difficult depending on the difficulty level selected.	Pass
6	Verify force jumps	When either player has an opportunity to jump an opponent's piece, they must take that opportunity.	Pass
7	Verify Red color selection functionality	Opponent clicks the red color button then starts a new game. The Computer should move first automatically.	Pass
8	Verify GameWin window pops up with sound	When either opponent wins the game, a window should pop up that reads "(winning color) Wins!" and the win sound should play.	Pass

TEST CASE 10: Test to verify 2-player gameplay
Method of Testing: Manual
Precondition: 'Start Game' button clicked on StartPanel, 'New Game' Button clicked in Checkers. Sounds unmuted. 2-Player Selected.

Step No.	Execution Steps	Expected Results	Pass / Fail / Description

1	Verify 2-player mode functionality	When a game is started with 2-player button selected the yellow player starts first. Followed by the red player. Game play is the same as 1-player but the computer player is replaced by a human player.	Pass

TEST CASE 11: Test to verify labels with corresponding radio button selectino
Method of Testing: Manual
Precondition: ‘Start Game’ button clicked on StartPanel. Sounds unmuted.

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Verify 1-Player radio button is selected	Whenever the yellow radio button is selected, the label next to the red piece graphic should say “Opponent’s Piece”. Next to red King: “Opponent’s King”. Next to yellow piece: “Your Piece”. Next to yellow king: “Your King”	Pass
2	Click ‘red’ radio button	Label next to the red piece graphic should change to “Your Piece”. Next to red King: “Your King”. Next to yellow piece: “Opponent’s Piece”. Next to yellow king: “Opponent’s King”	Fail: labels do not change with corresponding color radio button selection
3	Click ‘2-player’ radio button	Label next to the red piece graphic should change to “Player 2 Piece”. Next to red King: “Player 2 King”. Next	Fail: labels do not change for 2-player radio button selection

		to yellow piece: "Player 1 Piece". Next to yellow king: "Player 1 King"	

System Testing Bugs (Failures) and Enhancements

Test Case No.	Step No.	Bugs	Enhancements
3	2	Clicking 'Help' Button makes window pop up but window does not contain any text.	Add directions and information to pop up window.
6	1	Undo button only becomes available after two moves	Implement functionality that causes undo button to become available after only one move.
11	2,3	Labels do not change based on which radio buttons are selected	Implement functionality that causes labels to change with corresponding radio button selection.

ACCEPTANCE TESTING

TEST CASE 1: Acceptance testing 1-Player Easy/Yellow
Method of Testing: Manual
Precondition: Application successfully launched, easy difficulty selected, yellow color selected, 'New Game' clicked

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. High likelihood of player winning.	Pass

TEST CASE 2: Acceptance testing 1-Player Easy/Red
Method of Testing: Manual
Precondition: Application successfully launched, easy difficulty selected, red color selected, 'New Game' clicked

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. High likelihood of player winning.	Pass

TEST CASE 3: Acceptance testing 1-Player Fairly Easy/Yellow
Method of Testing: Manual
Precondition: Application successfully launched,fairly easy difficulty selected, yellow color selected, ‘New Game’ clicked

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. High likelihood of player winning.	Pass

TEST CASE 4: Acceptance testing 1-Player Fairly Easy/Red
Method of Testing: Manual
Precondition: Application successfully launched, fairly easy difficulty selected, red color selected, ‘New Game’ clicked

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. High likelihood of player winning.	Pass

TEST CASE 5: Acceptance testing 1-Player Moderate/Yellow
Method of Testing: Manual
Precondition: Application successfully launched, moderate difficulty selected, yellow color selected, ‘New Game’ clicked

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. Moderate likelihood of player winning.	Pass

TEST CASE 6: Acceptance testing 1-Player Moderate/Red
Method of Testing: Manual
Precondition: Application successfully launched, moderate difficulty selected, red color selected, ‘New Game’ clicked

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. Moderate likelihood of player winning.	Pass

TEST CASE 7: Acceptance testing 1-Player Bit Difficult/Yellow
Method of Testing: Manual
Precondition: Application successfully launched, bit difficult difficulty selected, yellow color selected, 'New Game' clicked

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. Low likelihood of player winning.	Pass

TEST CASE 8: Acceptance testing 1-Player Bit Difficult/Red
Method of Testing: Manual
Precondition: Application successfully launched, bit difficult difficulty selected, red color selected, 'New Game' clicked

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. Low likelihood of player winning.	Pass

TEST CASE 9: Acceptance testing 1-Player Tough/Yellow
Method of Testing: Manual
Precondition: Application successfully launched,tough difficulty selected, yellow color selected, ‘New Game’ clicked

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. Very low likelihood of player winning.	Pass

TEST CASE 10: Acceptance testing 1-Player Tough/Red
Method of Testing: Manual
Precondition: Application successfully launched, tough difficulty selected, red color selected, ‘New Game’ clicked

Step No.	Execution Steps	Expected Results	Pass / Fail / Description
1	Play 2 complete game of Checkers	Game plays to completion with no errors and displays winner at the end. Very low likelihood of player winning.	Pass

TEST CASE 11: Acceptance testing 2-Player
Method of Testing: Manual
Precondition: Application successfully launched, 2-Play selected, ‘New Game’ clicked

Step No.	**Execution Steps**	**Expected Results**	**Pass / Fail / Description**
1	Play 2 complete game of Checkers, alternating who goes first.	Game plays to completion with no errors and displays winner at the end.	Pass

Acceptance Testing Bugs (Failures) and Enhancements

Test Case No.	**Step No.**	**Bugs**	**Enhancements**
None			

All Testing Bugs and Enhancements

Testing Type	**TestCase No.**	**Step No.**	**Bug**	**Enhancement**
Unit	2	1	Undo() fails when attempted to be applied more than once consecutively	Make an array that holds the last three positions, since three is the max of undos available. Once a new position is held, the oldest position can be erased.
Integration	3	1	No gameplay directions present	Add gameplay directions to help window
System	3	2	Clicking 'Help' Button makes window pop up but window does not contain any text.	Add directions and information to pop up window.
System	6	1	Undo button only becomes available after two moves	Implement functionality that causes undo button to become available after only one move.
System	11	2,3	Labels do not change based on which radio buttons are selected	Implement functionality that causes labels to change with corresponding radio button selection.