CheckerBoard(int aDimension) – testConstructor_ValidDimension

<table>
<tr>
<td>

Input: = 8

State: N/A

</td>
<td>

Output:

State:

```
|   | 0| 1| 2| 3| 4| 5| 6| 7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |  |* |  |* |  |* |  |* |
|5 |* |o |* |o |* |o |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

A new CheckerBoard object with a dimension of 8x8

</td>
</tr>
</table>

CheckerBoard(int aDimension) – testConstructor_InvalidDimensionNegative

Input: -1 State: N/A	Output:None State: gives an error because of invalid dimension from negative number

CheckerBoard(int aDimension) – testConstructor_InvalidDimensionUpperFor16Board

Input: aDimension = 20 State: N/A	Output:N/A State: Gives error due to 20 being dimension

whatsAtPos(BoardPosition pos) – testWhatsAtPos_EmptyPosition

Input: pos = newBoardPosition(0,0) State: empty Board	Output: ‘ ‘ State: Board remains unchanged

whatsAtPos(BoardPosition pos) – testWhatsAtPos_OccupiedByPlayerX

Input:pos = new BoardPosition(1,1) State: Board with X at position (1,1)	Output: ‘X’ State: Board remains unchanged

whatsAtPos(BoardPosition pos) –testWhatsAtPos_OccupiedByPlayer0

Input:pos = new BoardPosition(2, 2) State: Board with ‘O’ at position (2, 2)	Output: ‘O’ State: Board remains unchanged

whatsAtPos(BoardPosition pos) – testWhatsAtPos_MinRow_MaxCol-X

<table>
<tr>
<td>

Input: pos = new BoardPosition(0, 7)

State:
```
|  | 0| 1| 2| 3| 4| 5| 6| 7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |  |* |  |* |  |* |  |* |
|5 |* |o |* |o |* |o |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

</td>
<td>

Output: ‘*’

State:
```
|  | 0| 1| 2| 3| 4| 5| 6| 7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |  |* |  |* |  |* |  |* |
|5 |* |o |* |o |* |o |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

</td>
</tr>
</table>

whatsAtPos(BoardPosition pos) – testWhatsAtPos_MaxRow_MinCol-X

Input:(8, 0)

State:

```
|   | 0| 1| 2| 3| 4| 5| 6| 7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |  |* |  |* |  |* |  |* |
|5 |* |o |* |o |* |o |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

Output: ‘*’

State:

```
|   | 0| 1| 2| 3| 4| 5| 6| 7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |  |* |  |* |  |* |  |* |
|5 |* |o |* |o |* |o |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

placePiece(BoardPosition pos, char player) – testPlacePieceBoundaryPosition

Input: pos = BoardPosition(0,0), player = ‘X’ State: Empty Board	Output: None State: X placed at position (0,0)

placePiece(BoardPosition pos, char player) –testPlacePieceValidO

Input: pos = BoardPosition(2,2), player = ‘O’ State: Empty Board	Output: None State: O placed at position (2,2)

placePiece(BoardPosition pos, char player) – testPlacePieceOnOccupiedPosition

Input: pos = BoardPosition(1,1), player = ‘O’ State: Board with X at position (1,1)	Output: error State: Unchanged

placePiece(BoardPosition pos, char player) – testPlacePieceValidX

Input: pos = BoardPosition(3,-3), player = X State: empty or initalized checker board	Output: None State: X placed at position (3,3)

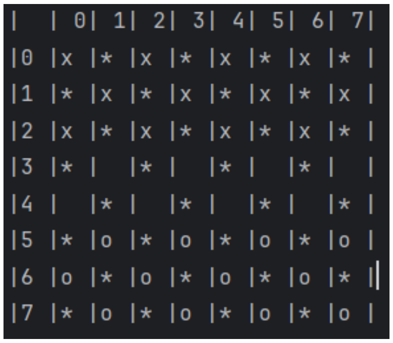
placePiece(BoardPosition pos, char player) – testPlacePieceChangeState

Input: pos = BoardPosition(3,3), player = X State: Empty Board	Output: None State: X placed at position (3,3) board state updated

placePiece(BoardPosition pos, char player) – TestPlacePieceInvalidPlayer

Input: pos = BoardPosition(2,2), player = Z State: An Empty CheckerBoard	Output: throws an error State: Unchanged

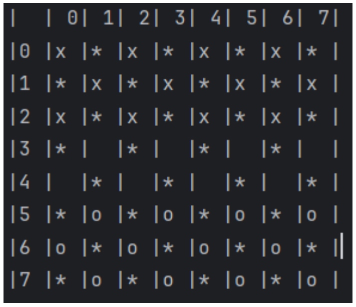
getPieceCounts(void) – testGetPieceCounts_PLAYER_ONE

Input: N/A State: CheckerBoard is initialized with a dimension of 8, pieces are placed in their default starting positions. 	Output: {12} State: Not changed from initial state

```
|  | 0| 1| 2| 3| 4| 5| 6| 7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |  |* |  |* |  |* |  |* |
|5 |* |o |* |o |* |o |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

getViableDirections(void) – testGetViableDirections_PLAYER_ONE

Input: N/A State: CheckerBoard is initialized with a dimension of 8, pieces are placed in their default starting positions.	Output:[SE, SW] State: Not changed from initial state.

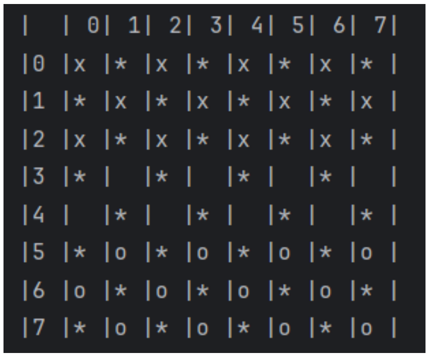


```
|  | 0| 1| 2| 3| 4| 5| 6| 7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |  |* |  |* |  |* |  |* |
|5 |* |o |* |o |* |o |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

addViableDirections(char player, DirectionEnum dir) –
testAddViableDirections_New_NE_Direction_for_PLAYER_ONE

Input: player = ‘x’,
dir = DirectionEnum.NE

State: CheckerBoard is initialized with a dimension of 8, pieces are placed in their default starting positions, and the viable directions have not been modified yet



```
|  | 0| 1| 2| 3| 4| 5| 6| 7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |  |* |  |* |  |* |  |* |
|5 |* |o |* |o |* |o |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

Output: N/A

State:The hashmap of viableDirections for the player 'x' is updated to include DirectionEnum.NE.

getRowNum(void) – testGetRowNum

Input: N/A

State: Board is initalized with 8 x 8

Output: 8

State: Unchanged

getColNum(void) – testGetColNum

Input: N/A State: Board initialized with 8 x 8	Output: 8 State: Unchanged

checkPlayerWin(Character player) – testPlayerWinNoWinner

Input: player = X State: Board with no winning board for X	Output: false State: Unchanged

checkPlayerWin(Character player) – testCheckPlayerWinWinner

Input: player = O State: Board with winning Board for O	Output: True State: Unchanged

crownPiece(BoardPosition posOfPlayer) – testValidPosPlayerOne

Input: posOfPlayer = BoardPosition(2,2) State: The game board contains Player One's piece at position (2, 2).	Output: The piece at position (2, 2) is converted to an uppercase king piece. State: The game board is updated with the crowned piece. Movable directions and map of pieces are unchanged.

crownPiece(BoardPosition posOfPlayer) – testValidPosPlayerTwo

Input: posOfPlayer = BoardPosition(5,3) State: The game board contains Player Two's piece at position (5, 3).	Output: The piece at position (5, 3) is converted to an uppercase king piece. State: The game board is updated with the crowned piece. Movable directions and map of pieces are unchanged.

crownPiece(BoardPosition posOfPlayer) – testAlreadyCrowned

Input: posOfPlayer = BoardPosition(0, 2) State: The game board contains a piece at position (0, 2) that is already crowned.	Output: The function does not modify the piece at the given position. State: The game board, movable directions, and map of pieces are unchanged.

crownPiece(BoardPosition posOfPlayer) – testCrownPieceMultiplePieces

Input: posOfPlayer = BoardPosition(0,2)
posOfPlayer = BoardPosition(0,4)
posOfPlayer = BoardPosition(0,6)
State: The given position is invalid or out of bounds.
	Output: The pieces at positions (0,2), (0,4), and (0,6) are crowned
	State: The game board, movable directions, and map of pieces are unchanged.

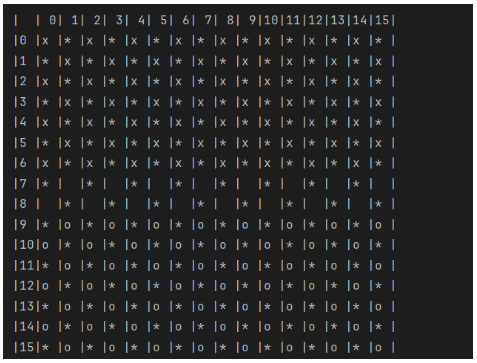
movePiece(BoardPosition startingPos, DirectionEnum dir) – testValidMove

Input: startingPos = BoardPosition(2, 2) dir = DirectionEnum.SE State: The game board contains a piece at position (2, 2), and the position its moving to is empty.	Output: The piece at position (2, 2) is moved to the southeast. The old location now has an empty position. State: The game board is updated with the piece moved to the new position. Movable directions and map of pieces are unchanged.

movePiece(BoardPosition startingPos, DirectionEnum dir) – testMovePieceSize16Board

Input: startingPos = BoardPosition(1, 15) dir = DirectionEnum.SW	Output: The piece at position (1, 15) is moved to the southeast. The old location now has an empty position. State: The game board is updated with the piece moved to the new position. Movable directions and map of pieces are unchanged.

State:



```
|  | 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|
|0 |x |* |x |* |x |* |x |* |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |x |* |x |* |x |* |x |* |
|3 |* |x |* |x |* |x |* |x |* |x |* |x |* |x |* |x |
|4 |x |* |x |* |x |* |x |* |x |* |x |* |x |* |x |* |
|5 |* |x |* |x |* |x |* |x |* |x |* |x |* |x |* |x |
|6 |x |* |x |* |x |* |x |* |x |* |x |* |x |* |x |* |
|7 |* |  |* |  |* |  |* |  |* |  |* |  |* |  |* |  |
|8 |  |* |  |* |  |* |  |* |  |* |  |* |  |* |  |* |
|9 |* |o |* |o |* |o |* |o |* |o |* |o |* |o |* |o |
|10|o |* |o |* |o |* |o |* |o |* |o |* |o |* |o |* |
|11|* |o |* |o |* |o |* |o |* |o |* |o |* |o |* |o |
|12|o |* |o |* |o |* |o |* |o |* |o |* |o |* |o |* |
|13|* |o |* |o |* |o |* |o |* |o |* |o |* |o |* |o |
|14|o |* |o |* |o |* |o |* |o |* |o |* |o |* |o |* |
|15|* |o |* |o |* |o |* |o |* |o |* |o |* |o |* |o |
```

movePiece(BoardPosition startingPos, DirectionEnum dir) – testEdgeCases

Input: startingPos = BoardPosition(1,7)
dir = DirectionEnum.SW

State: Position (1, 7) contains a piece at the edge of the game board.

Output: The function correctly handles moving the piece to a new position.

State: The game board, movable directions, and map of pieces remain consistent with the position update.

jumpPiece(BoardPosition startingPos, DirectionEnum dir) – testjumpPiece_to_Occupied_Space

Input:(BoardPosition(3, 3), NW)

State:
```
|   | 0| 1| 2| 3| 4| 5| 6| 7|
|0  |x |* |x |* |x |* |x |* |
|1  |* |x |* |x |* |x |* |x |
|2  |x |* |  |* |x |* |x |* |
|3  |* |  |* | x|* |  |* |  |
|4  |  |* |o |* | o|* |  |* |
|5  |* |o |* |o |* |  |* | |
|6  |o |* |o |* |o |* |o |* |
|7  |* |o |* |o |* |o |* |o |
```
pieceCount = {

Output: Error stating invalid space, try again

State:
```
|   | 0| 1| 2| 3| 4| 5| 6| 7|
|0  |x |* |x |* |x |* |x |* |
|1  |* |x |* |x |* |x |* |x |
|2  |x |* |  |* |x |* |x |* |
|3  |* |  |* | x|* |  |* |  |
|4  |  |* |  |* | o|* |  |* |
|5  |* |o |* |o |* |  |* |o |
|6  |o |* |o |* |o |* |o |* |
|7  |* |o |* |o |* |o |* |o |
```

<table>
<tr><td>PLAYER_ONE: 12
PLAYER_TWO: 12
}</td><td>pieceCount = {
PLAYER_ONE: 12
PLAYER_TWO: 12
}</td></tr>
</table>

jumpPiece(BoardPosition startingPos, DirectionEnum dir) –
testjumpPiecePlayerTwoOverPlayerOne

<table>
<tr><td>Input:(BoardPosition(4, 4), NW)

State: | | 0| 1| 2| 3| 4| 5| 6| 7|
0	x	*	x	*	x	*	x	*
1	*	x	*	x	*	x	*	x
2	x	*		*	x	*	x	*
3	*		*	x	*		*	
4		*		*	o	*		*
5	*	o	*	o	*		*	o
6	o	*	o	*	o	*	o	*
7 |* |o |* |o |* |o |* |o | pieceCount = { PLAYER_ONE: 12 PLAYER_TWO: 12 }</td><td>Output: BoardPosition(2, 2) State:		0| 1| 2| 3| 4| 5| 6| 7
0	x	*	x	*	x	*	x	*
1	*	x	*	x	*	x	*	x
2	x	*	o	*	x	*	x	*
3	*		*		*		*	
4		*		*		*		*
5	*	o	*	o	*		*	o
6	o	*	o	*	o	*	o	*
7	*	o	*	o	*	o	*	o	 pieceCount = { PLAYER_ONE: 11 PLAYER_TWO: 12 }</td></tr>
</table>

jumpPiece(BoardPosition startingPos, DirectionEnum dir) –
testjumpPiecePlayerOneOverPlayerTwo

<table>
<tr>
<td>Input:(BoardPosition(3, 3), SE)

State: | | 0| 1| 2| 3| 4| 5| 6| 7|
0	x	*	x	*	x	*	x	*
1	*	x	*	x	*	x	*	x
2	x	*		*	x	*	x	*
3	*		*	x	*		*	
4		*		*	o	*		*
5	*	o	*	o	*		*	o
6	o	*	o	*	o	*	o	*
7	*	o	*	o	*	o	*	o	 pieceCount = { PLAYER_ONE: 12 PLAYER_TWO: 12 }</td>
<td>Output:BoardPosition(5, 5) State:		0	1	2	3	4	5	6	7
0	x	*	x	*	x	*	x	*
1	*	x	*	x	*	x	*	x
2	x	*		*	x	*	x	*
3	*		*		*		*	
4		*		*		*		*
5	*	o	*	o	*	x	*	o
6	o	*	o	*	o	*	o	*
7	*	o	*	o	*	o	*	o	 pieceCount = { PLAYER_ONE: 12 PLAYER_TWO: 11 }</td>
</tr>
</table>

playerLostPieces(int numPieces, char player, HashMap pieceCounts) – testplayerLostPiecesPlayerOneLost1

<table>
<tr>
<td>Input:(1, PLAYER_ONE)

State: pieceCount = {
PLAYER_ONE: 12
PLAYER_TWO: 12
}</td>
<td>Output: void

State: pieceCount = {
PLAYER_ONE: 11
PLAYER_TWO: 12
}</td>
</tr>
</table>

scanSurroundingPositions(BoardPosition startingPos) – testscanSurroundingPositionsInMiddle

Input:BoardPosition(4,4)

State:
```
|   |  0|  1|  2|  3|  4|  5|  6|  7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |   |* |   |* | o |* |   |* |
|5 |* |o |* |o |* |  |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

Output: {NE: EMPTY_POS,
SE: EMPTY_POS,
SW: PLAYER_TWO,
NW: EMPTY_POS}

State: [board is unchanged]

scanSurroundingPositions(BoardPosition startingPos) – testscanSurroundingPositionsAtTop

Input:BoardPosition(0,3)

State:
```
|   |  0|  1|  2|  3|  4|  5|  6|  7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |   |* |   |* | o |* |   |* |
|5 |* |o |* |o |* |  |* |o |
|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |
```

Output: {SE: PLAYER_ONE,
SW: PLAYER_ONE}

State: [board is unchanged]

scanSurroundingPositions(BoardPosition startingPos) – testscanSurroundingPositionsOnRight

Input:BoardPosition(1, COL_NUM)

State:
```
|   |  0|  1|  2|  3|  4|  5|  6|  7|
|0 |x |* |x |* |x |* |x |* |
|1 |* |x |* |x |* |x |* |x |
|2 |x |* |x |* |x |* |x |* |
|3 |* |  |* |  |* |  |* |  |
|4 |   |* |   |* | o |* |   |* |
|5 |* |o |* |o |* |  |* |o |
```

Output: {SW: PLAYER_ONE,
NW: PLAYER_ONE}

State: [board is unchanged]

<table>
<tr><td>|6 |o |* |o |* |o |* |o |* |
|7 |* |o |* |o |* |o |* |o |</td><td></td></tr>
</table>

scanSurroundingPositions(BoardPosition startingPos) – testScanSurroundingPositionsAtBottom

<table>
<tr><td>Input:BoardPosition(ROW_NUM, 1)

State: | | 0| 1| 2| 3| 4| 5| 6| 7|
0	x	*	x	*	x	*	x	*
1	*	x	*	x	*	x	*	x
2	x	*	x	*	x	*	x	*
3	*		*		*		*	
4		*		*	o	*		*
5	*	o	*	o	*		*	o
6	o	*	o	*	o	*	o	*
7	*	o	*	o	*	o	*	o	</td><td>Output: {NE: PLAYER_TWO, NW: PLAYER_TWO} State: [board is unchanged]</td></tr>
</table>

scanSurroundingPositions(BoardPosition startingPos) – testScanSurroundingPositionsOnLeft

<table>
<tr><td>Input:BoardPosition(2, 0)

State: | | 0| 1| 2| 3| 4| 5| 6| 7|
0	x	*	x	*	x	*	x	*
1	*	x	*	x	*	x	*	x
2	x	*	x	*	x	*	x	*
3	*		*		*		*	
4		*		*	o	*		*
5	*	o	*	o	*		*	o
6	o	*	o	*	o	*	o	*
7	*	o	*	o	*	o	*	o	</td><td>Output: {NE: PLAYER_ONE, SE: EMPTY_POS} State: [board is unchanged]</td></tr>
</table>

getDirection(DirectionEnum dir) – testgetDirectionNE

Input: DirectionEnum.NE State: N/A	Output: BoardPosition(1, 1) State: N/A

What tests did each team member write? Just tell me the names of the functions (unless for some reason multiple team members wrote functions for the same method. In that case, tell me which tests specifically by giving me the test names)

Cary Dill	whatsAtPos, CheckerBoard constructor, PlacePiece, getColNum, getRowNum, checkPlayerWin
Jared Burney	getPieceCounts, getViableDirections,addViableDirections
Ryan Kissel	scanSurroundingPositions, jumpPiece, getDirection, playerLostPieces
Christian Dew	crownPiece, movePiece