CPSC 2150 Project Report

Abigail Barrett

Requirements Analysis

Functional Requirements:

1. As a player I can view the current game board on my turn, so that I know which spaces I can put my token in.
2. As a player I can view which player's turn it is, so that I know which player is adding the token.
3. As a player I can select the row to add my token to so that I can add my token to that position.
4. As a player I can select the column to add my token to so that I can add my token to that position.
5. As a player I can select a position with a token already in it, so I can see if I can place my token in that position.
6. As a player I can select a position not on the grid, so I can see if I can place my token in that position.
7. As a player I can place number_to_win of my markers in a row horizontally, so I can win the game.
8. As a player I can place number_to_win of my markers in a row vertically, so I can win the game.
9. As a player I can place number_to_win of my markers in a row diagonally, so I can win the game.
10. As a player I can select an empty position on the grid, so I can place my token in that position
11. As a player I can view that I made an error so that I know not to select that position.
12. As a player I can select another position after I made an error so that I can play a token in a valid position on the board.
13. As a player I can view when the game is over, so that I know when the game has finished.
14. As a player I can view which player won the game, so that I know who won the game.
15. As a player I can view the game board when the game is over, so I know what positions my opponent and I put our tokens in.
16. As a player I can view when the game is a draw, so I know no one won the game.
17. As a player I can choose to play the game again so that I can play another game.
18. As a player I can exit the program so that I cannot play another game.
19. As a player I can specify the number_of_rows, so that I can choose the size of my board.
20. As a player I can specify the number_of_columns, so that I can choose the size of my board.
21. As a player I can specify the number_to_win, so that I can choose how many tokens it takes to win the game.
22. As a player I can choose how many players will play the game, so that I can play with more players.
23. As a player I can choose the token of each player playing the game, so I can customize my token.
24. As a player I can view when I entered an invalid number_to_win, so I can choose a number_to_win that will make the game work correctly.
25. As a player I can view when I entered an invalid number_of_rows, so I can choose a number_of_rows that will make the game work correctly.

26. As a player I can view when I entered an invalid number_of_columns, so I can choose a number_of_columns that will make the game work correctly.

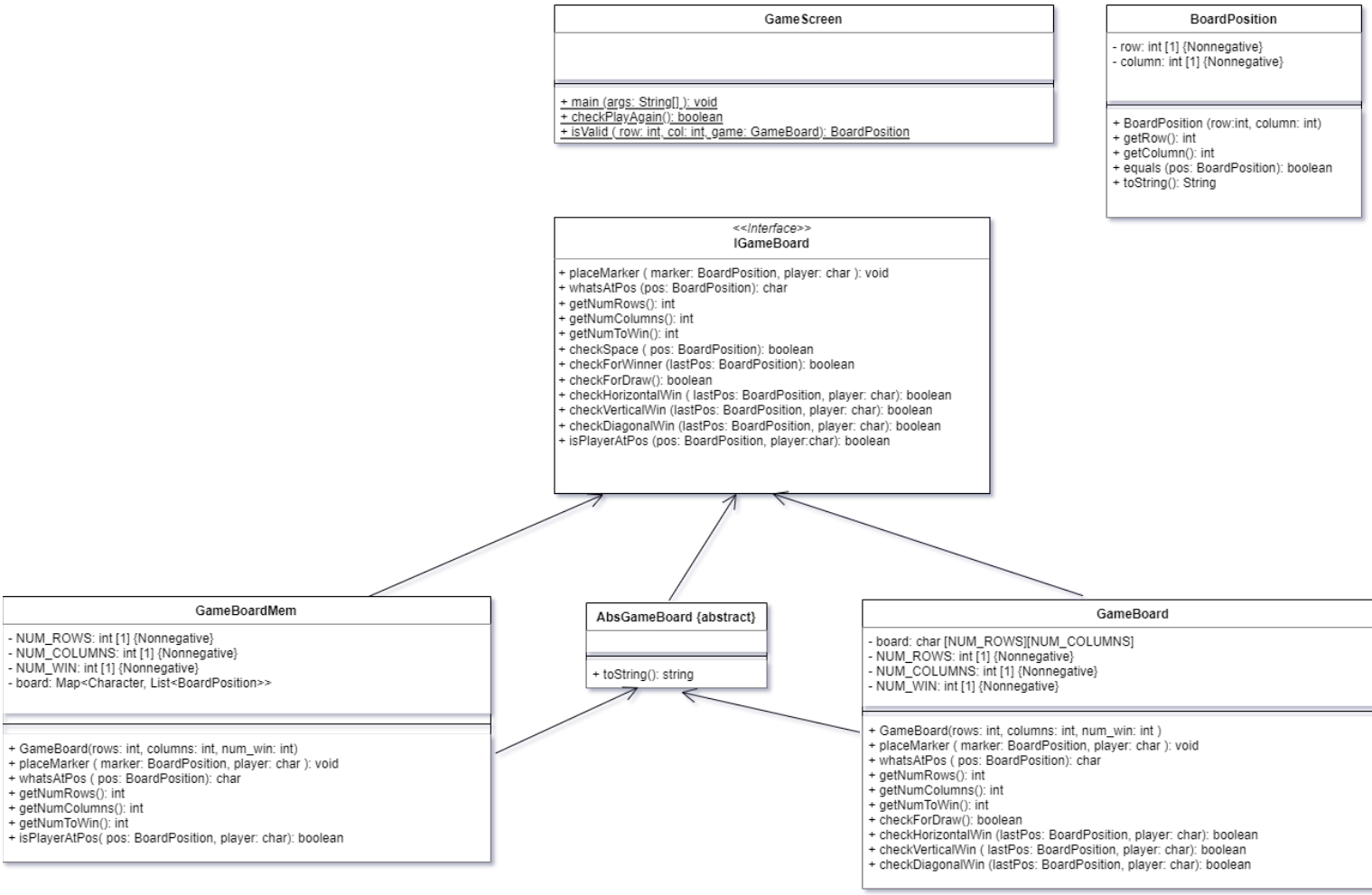
Non-Functional Requirements

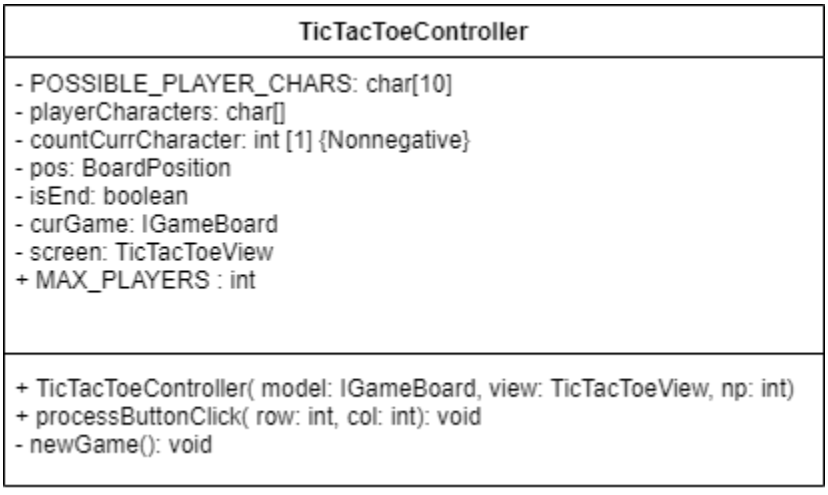
1. The system must be written in Java.
2. The system should run on unix.
3. The system should run without the user noticing any delays.
4. The system should display an easily readable GUI to the user.
5. The system should use memory implementation based on the GUI requirements.
6. The system should not crash when run
7. The system can run with either a fast implementation or a memory efficient implementation
8. If fast implementation, the board is of size user_entered_number_of_rows x user_entered_number_of_columns
9. If memory efficient implementation, the board has no allocated memory until user enters value
10. Player order of play goes in order of player tokens (provided in Controller)
11. Player order rotates in the same order of play

Deployment Instructions

Details in Projects 2-5.

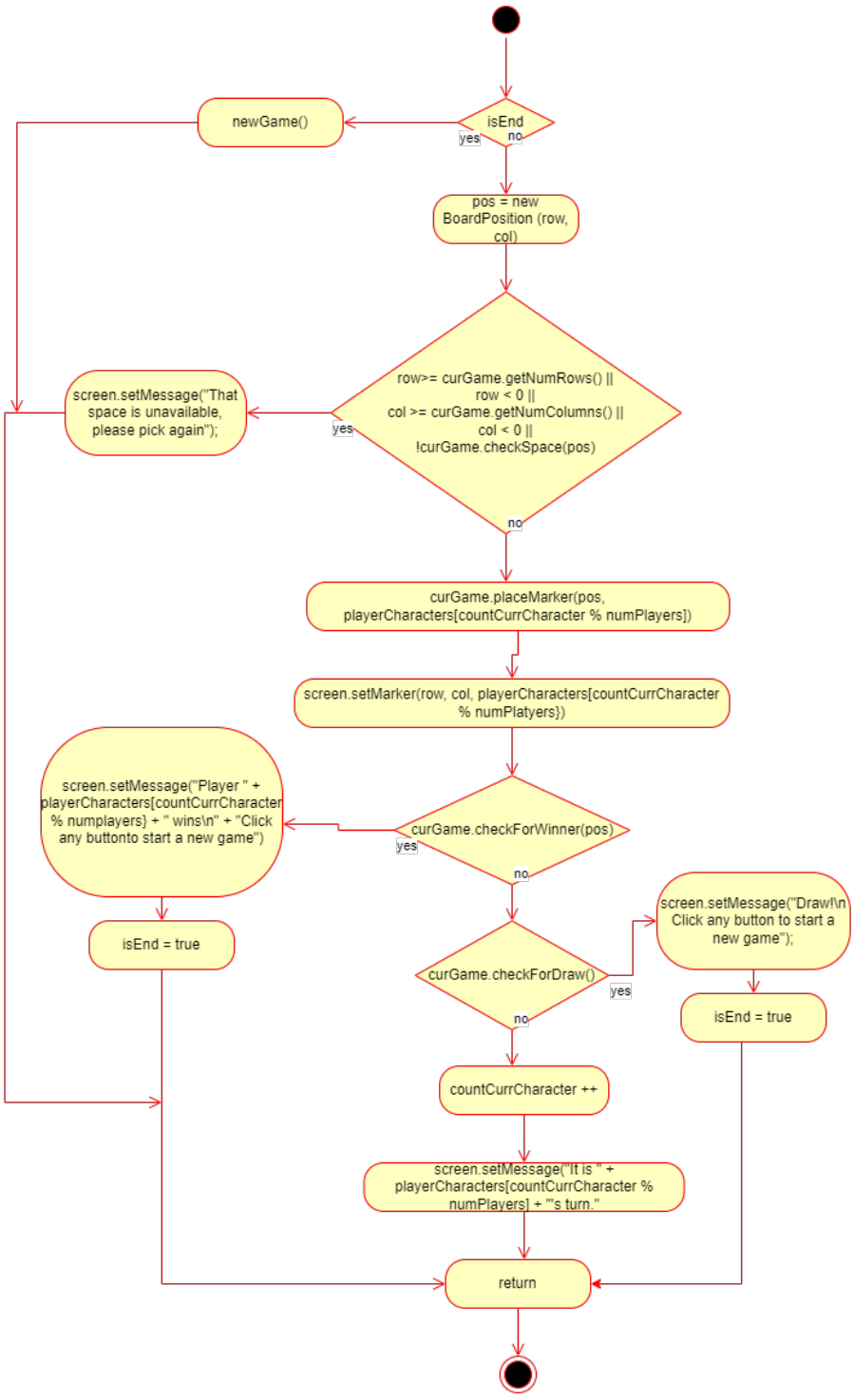
System Design


GameScreen
+ main (args: String[]): void
+ checkPlayAgain(): boolean
+ isValid (row: int, col: int, game: GameBoard): BoardPosition
BoardPosition
- row: int [1] {Nonnegative}
- column: int [1] {Nonnegative}
+ BoardPosition (row:int, column: int)
+ getRow(): int
+ getColumn(): int
+ equals (pos: BoardPosition): boolean
+ toString(): String
<<Interface>>
IGameBoard
+ placeMarker (marker: BoardPosition, player: char): void
+ whatsAtPos (pos: BoardPosition): char
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ checkSpace (pos: BoardPosition): boolean
+ checkForWinner (lastPos: BoardPosition): boolean
+ checkForDraw(): boolean
+ checkHorizontalWin (lastPos: BoardPosition, player: char): boolean
+ checkVerticalWin (lastPos: BoardPosition, player: char): boolean
+ checkDiagonalWin (lastPos: BoardPosition, player: char): boolean
+ isPlayerAtPos (pos: BoardPosition, player:char): boolean
GameBoardMem
- NUM_ROWS: int [1] {Nonnegative}
- NUM_COLUMNS: int [1] {Nonnegative}
- NUM_WIN: int [1] {Nonnegative}
- board: Map<Character, List<BoardPosition>>
+ GameBoard(rows: int, columns: int, num_win: int)
+ placeMarker (marker: BoardPosition, player: char): void
+ whatsAtPos (pos: BoardPosition): char
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ isPlayerAtPos(pos: BoardPosition, player: char): boolean
AbsGameBoard {abstract}
+ toString(): string
GameBoard
- board: char [NUM_ROWS][NUM_COLUMNS]
- NUM_ROWS: int [1] {Nonnegative}
- NUM_COLUMNS: int [1] {Nonnegative}
- NUM_WIN: int [1] {Nonnegative}
+ GameBoard(rows: int, columns: int, num_win: int)
+ placeMarker (marker: BoardPosition, player: char): void
+ whatsAtPos (pos: BoardPosition): char
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ checkForDraw(): boolean
+ checkHorizontalWin (lastPos: BoardPosition, player: char): boolean
+ checkVerticalWin (lastPos: BoardPosition, player: char): boolean
+ checkDiagonalWin (lastPos: BoardPosition, player: char): boolean


TicTacToeController
- POSSIBLE_PLAYER_CHARS: char[10]
- playerCharacters: char[]
- countCurrCharacter: int [1] {Nonnegative}
- pos: BoardPosition
- isEnd: boolean
- curGame: IGameBoard
- screen: TicTacToeView
+ MAX_PLAYERS : int
+ TicTacToeController(model: IGameBoard, view: TicTacToeView, np: int)
+ processButtonClick(row: int, col: int): void
- newGame(): void

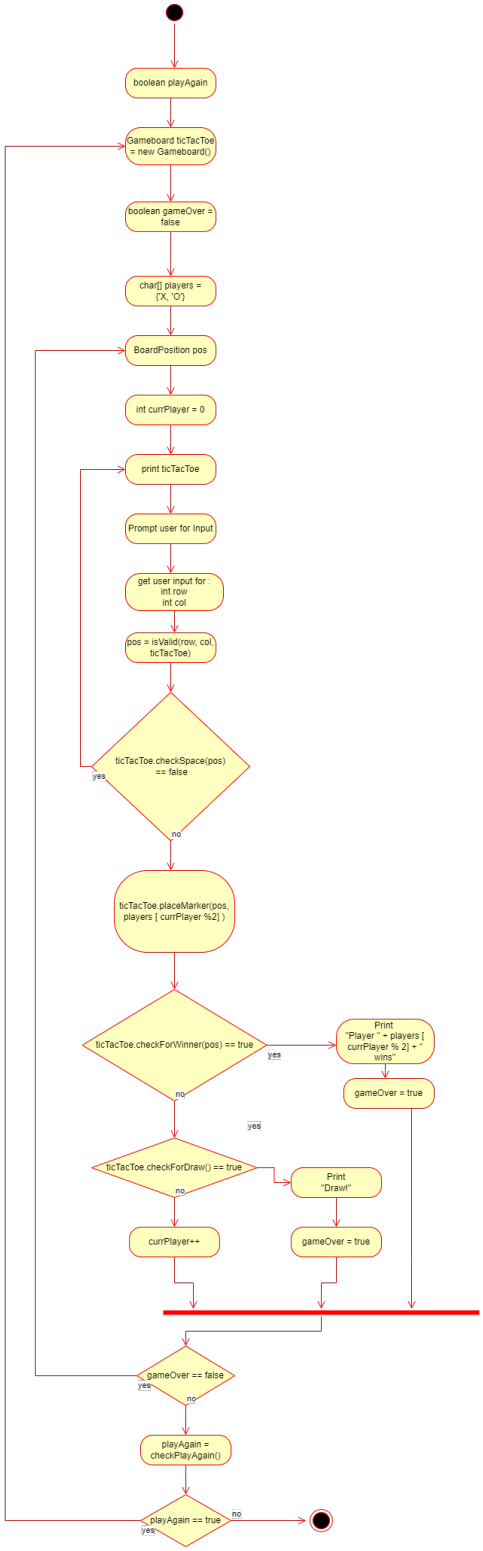
TicTacToeController:

processButtonClick

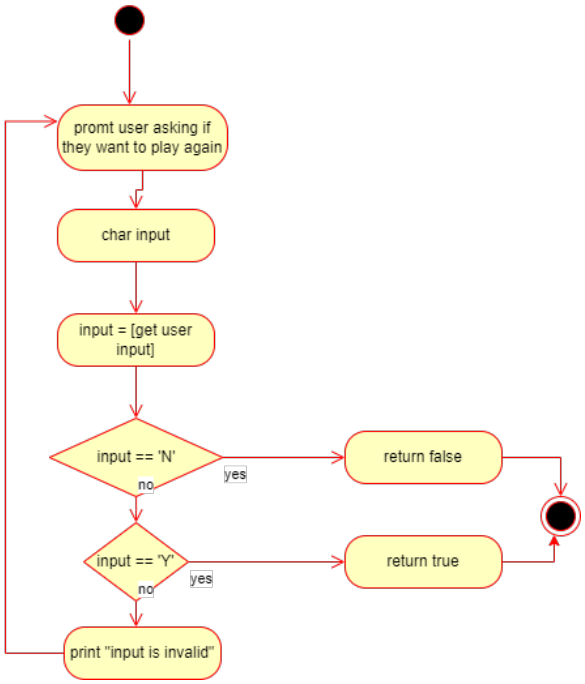


GameScreen:

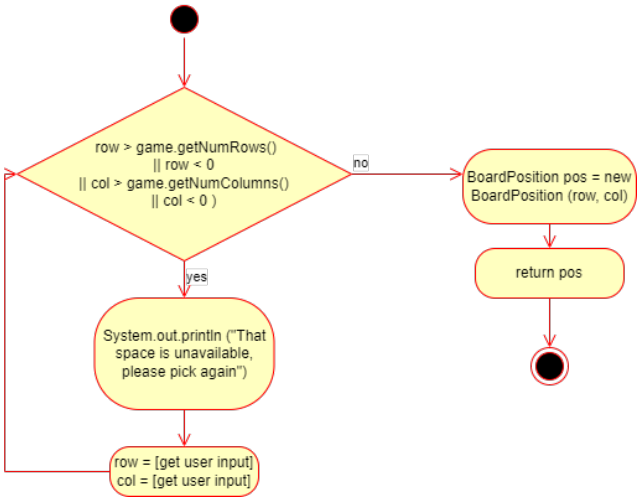
Main



checkPlayAgain()

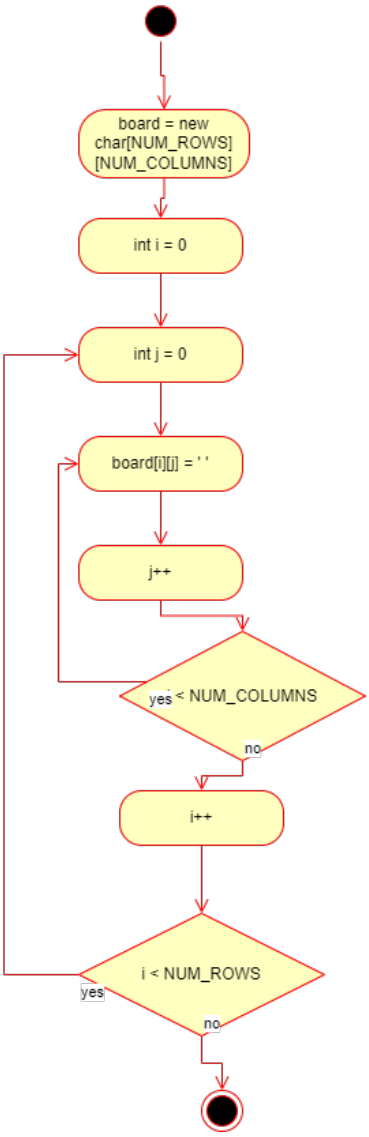


isValid

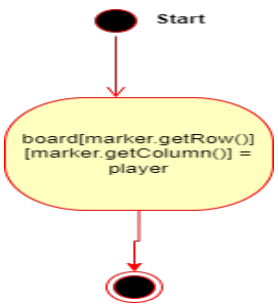


GameBoard:

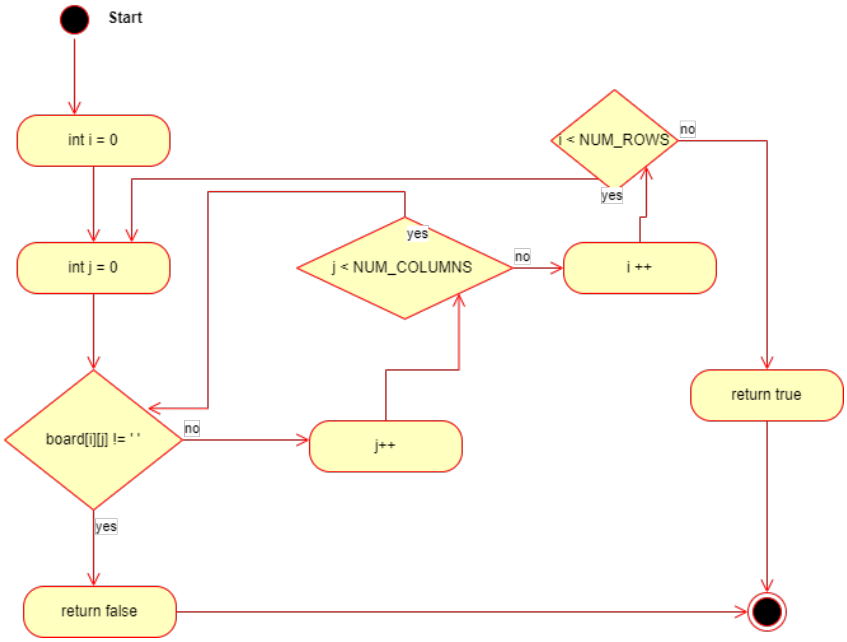
Constructor



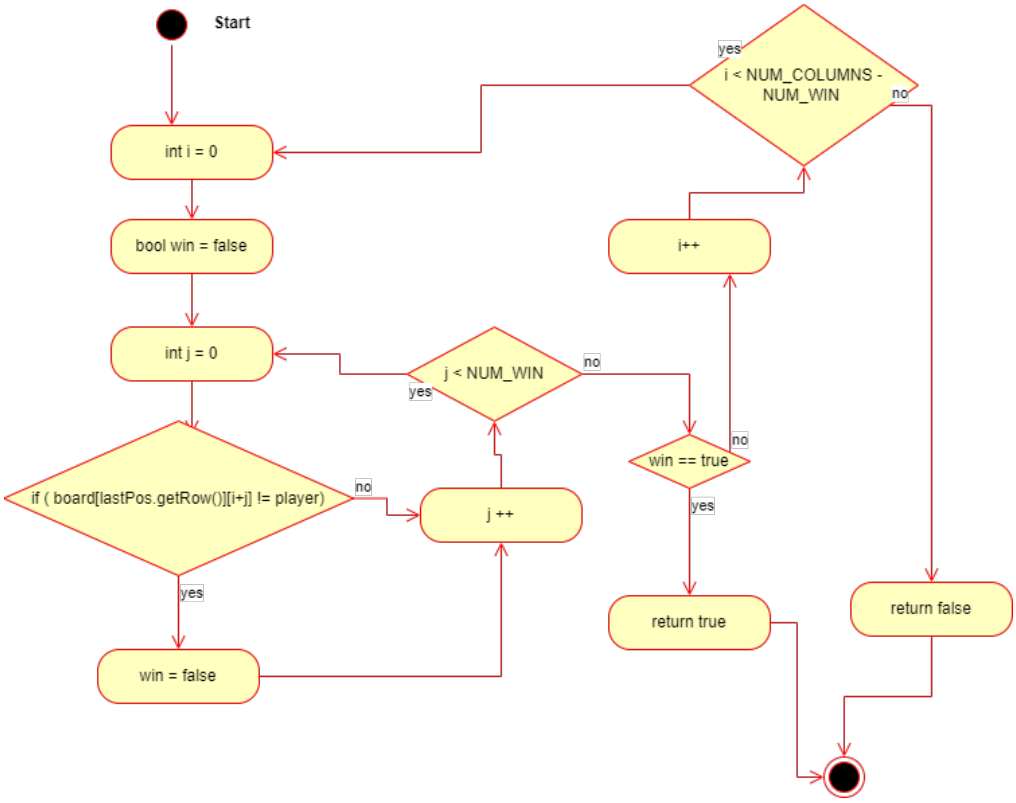
Place Marker



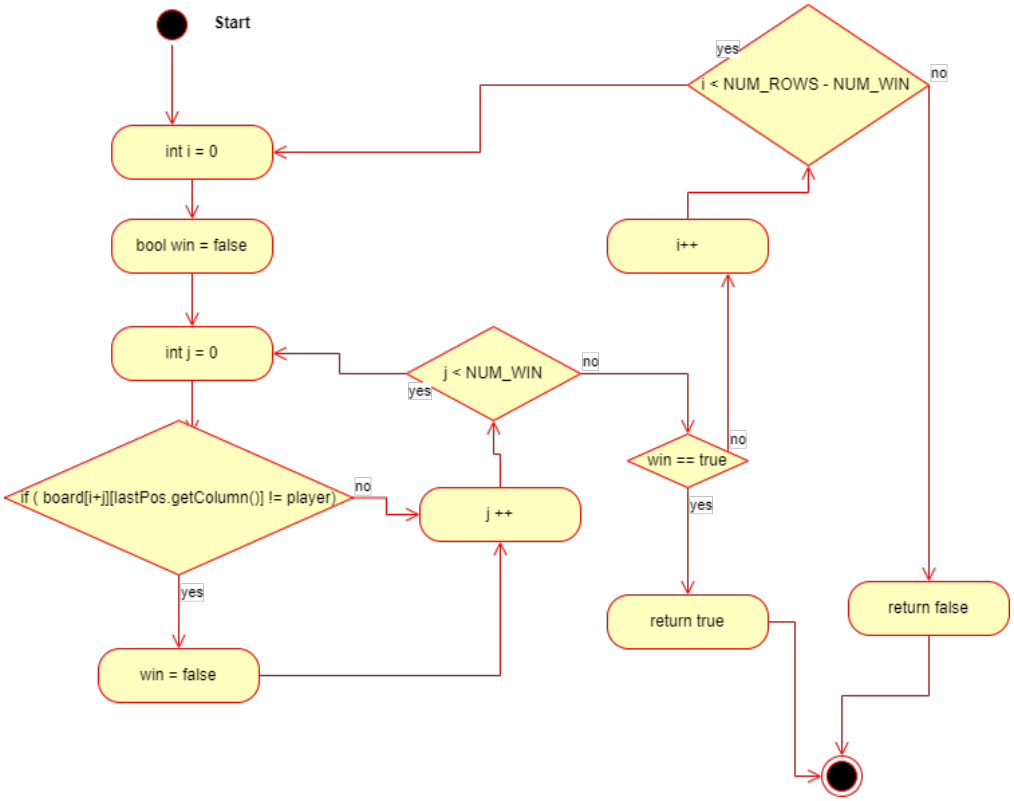
CheckForDraw



CheckHorizontalWin

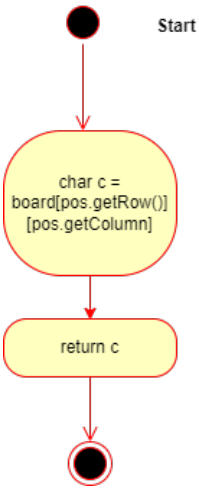


checkVerticalWin

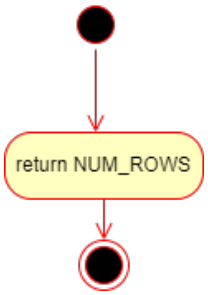


checkDiagonal Win

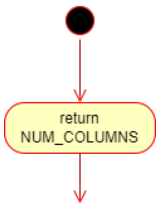
WhatsAtPosition



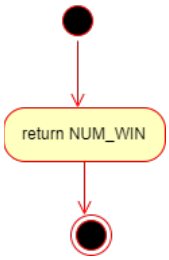
getNumRows



getNumColumns

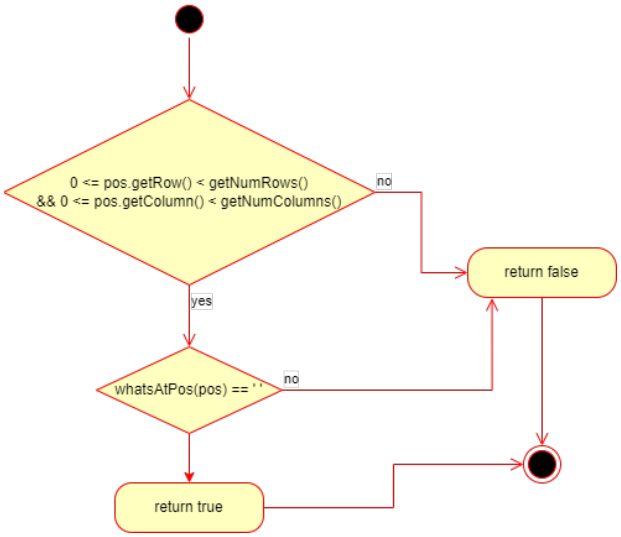


getNumToWin

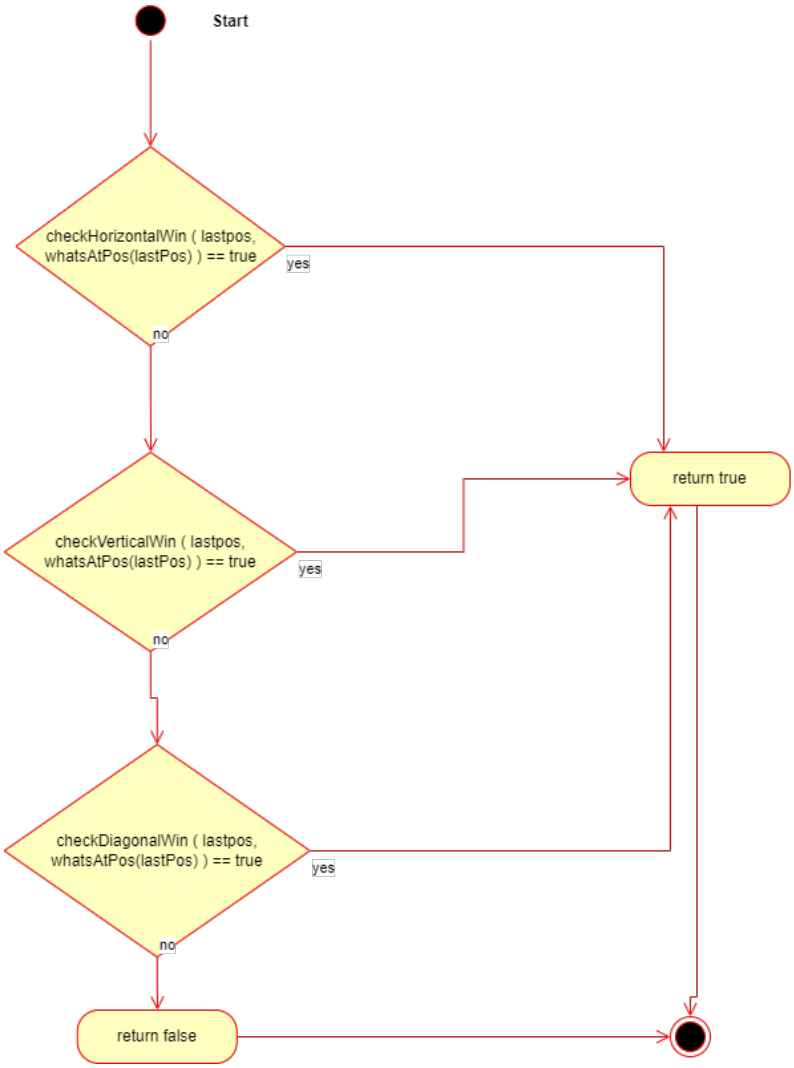


IGameBoard:

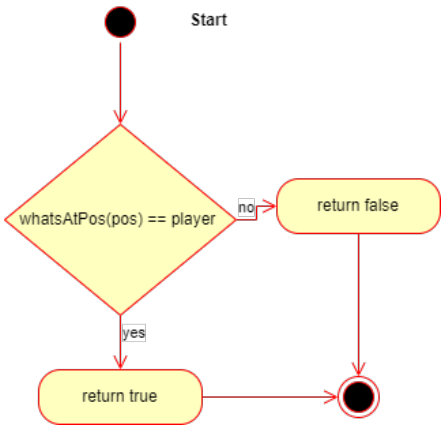
checkSpace



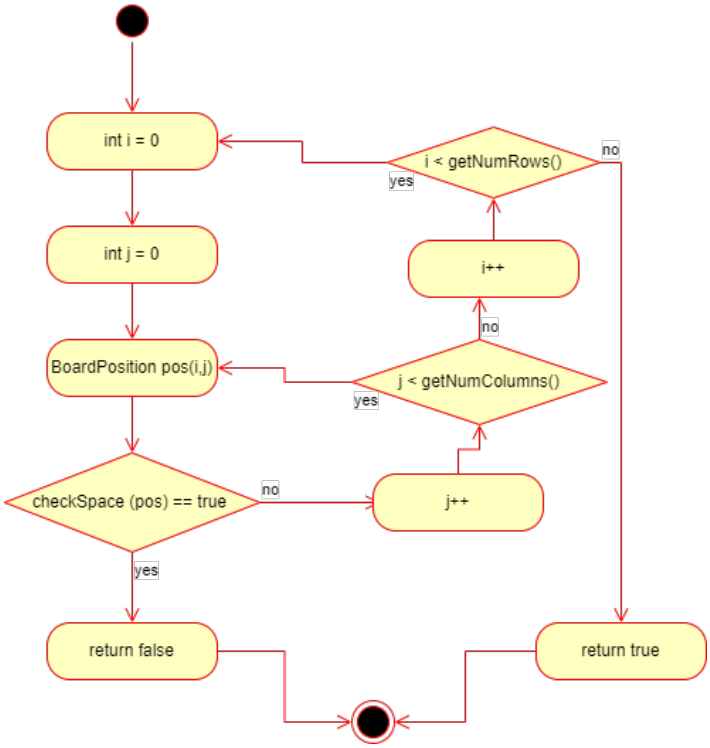
checkForWinner



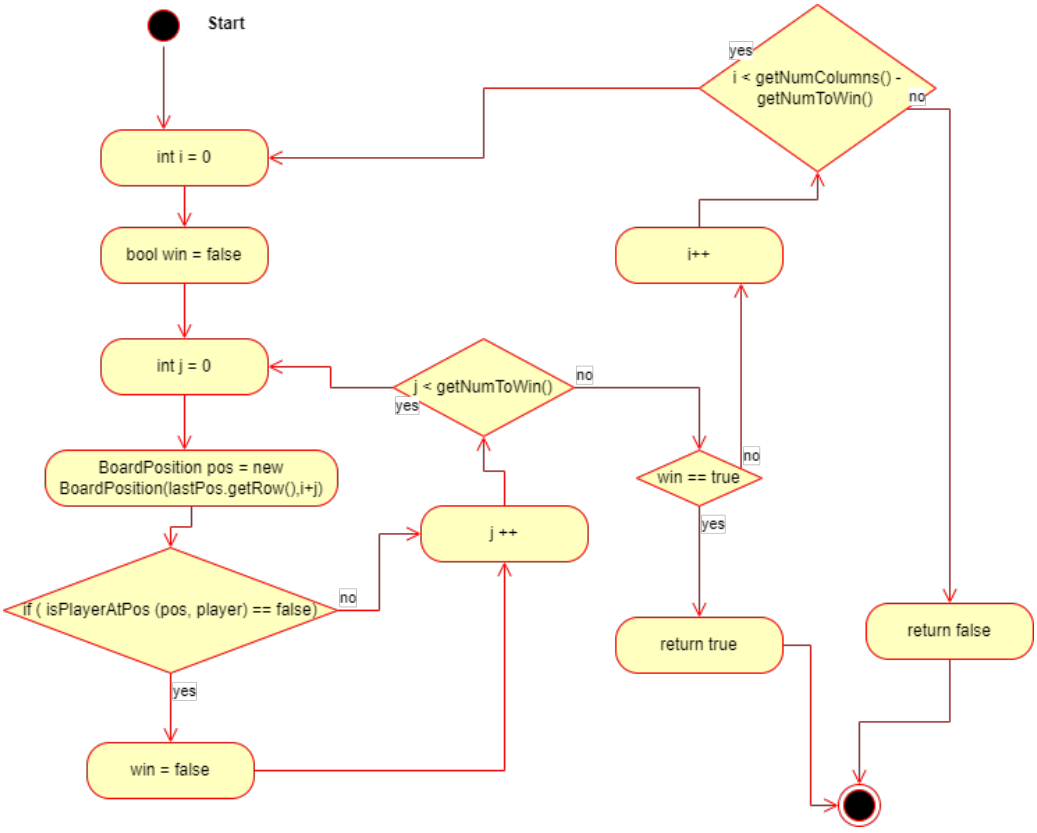
isPlayerAtPos



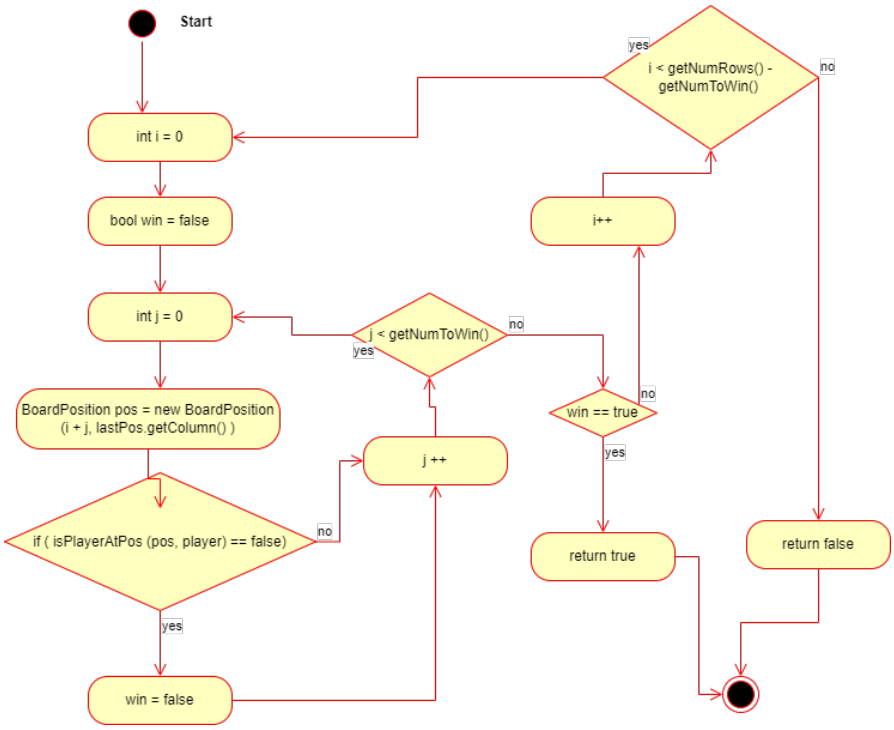
checkForDraw



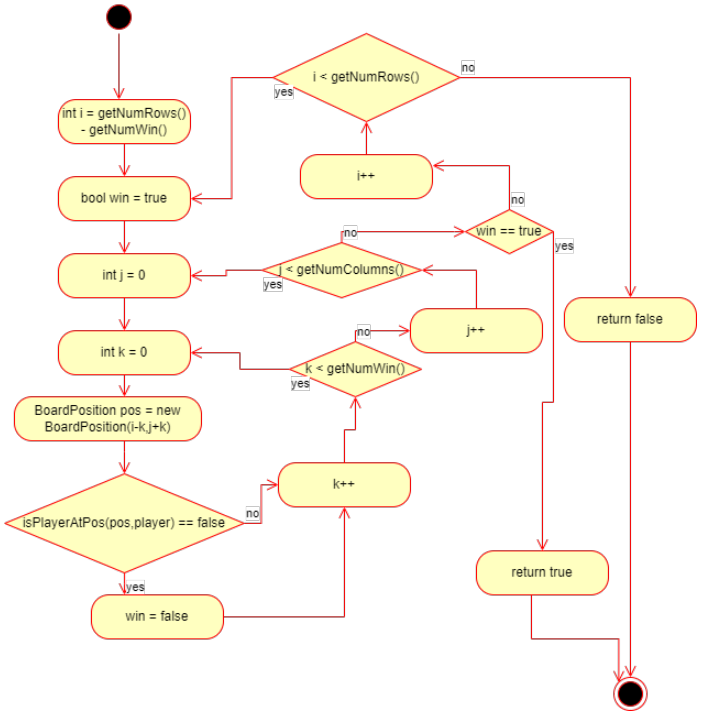
checkHorizontalWin



checkVerticalWin

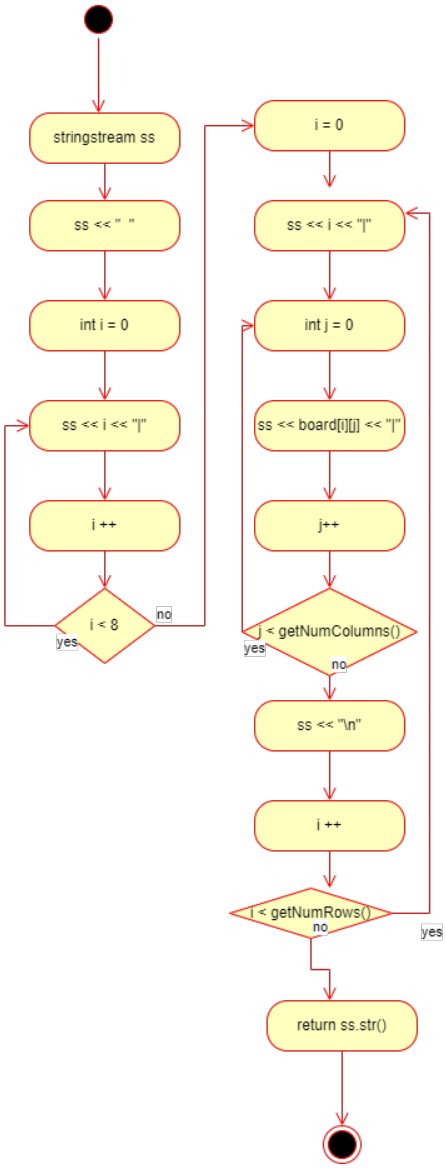


checkDiagonalWin



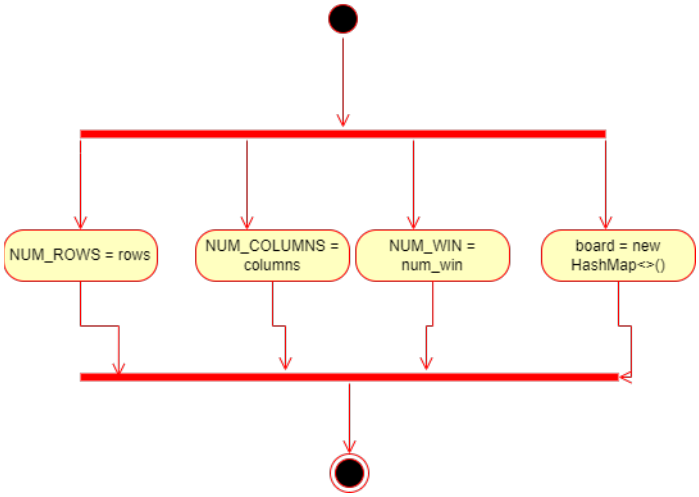
AbsGameBoard:

toString

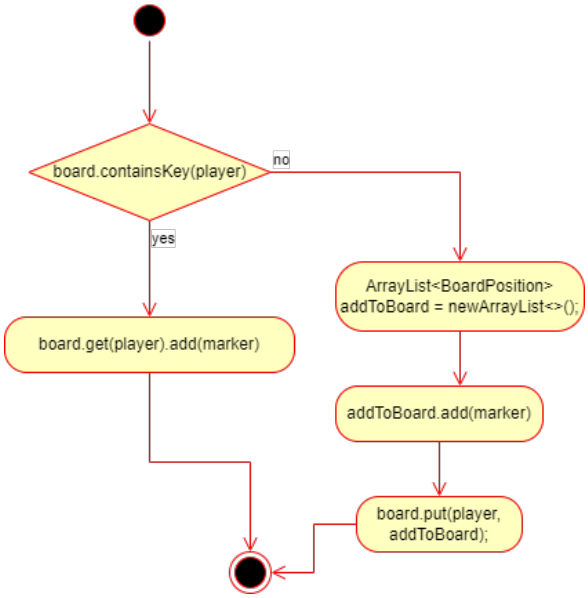


GameBoardMem:

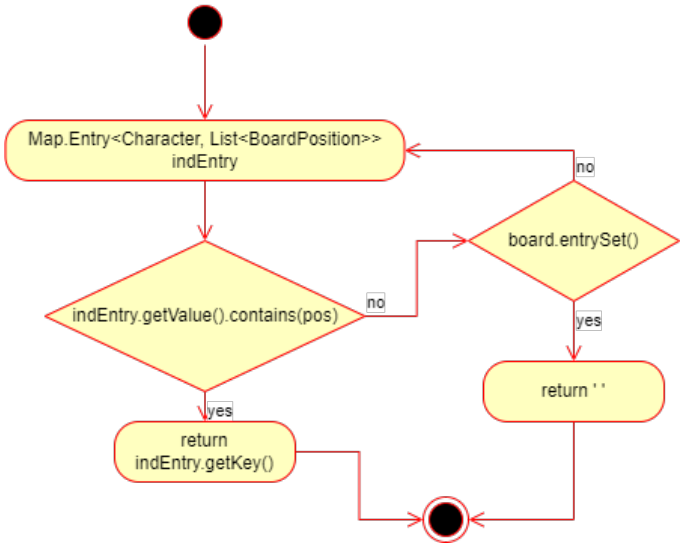
Constructor



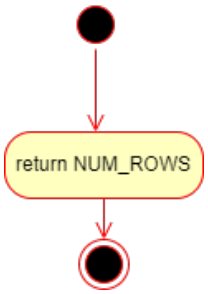
PlaceMarker



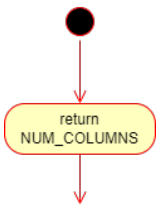
WhatsAtPos



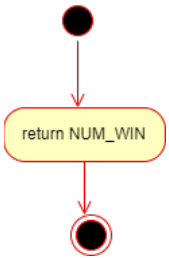
getNumRows



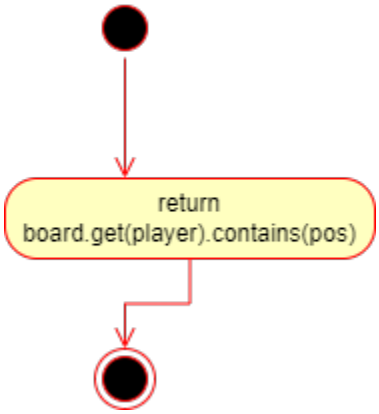
getNumColumns



getNumToWin



isPlayerAtPos



Test Cases

JUnit test function.
Test the following methods:

`Constructor (final int rows, final int columns, final int num_win)`

<table>
<tr>
<td>Input:

Rows = 3
Columns = 3
Num_win = 3</td>
<td>Output:

State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
This test case is unique and distinct because I am creating an empty GameBoard of the minimum size.

Function Name:
testGameBoardConstructor_min_num_row_col</td>
</tr>
<tr>
<td>Input:

Rows = 100
Columns = 100
Num_win = 25</td>
<td>Output:

State:
100x100 empty table</td>
<td>Reason:
This test case is unique and distinct because I am creating an empty GameBoard of the maximum size.

Function Name:
testGameBoardConstructor_max_num_row_col</td>
</tr>
<tr>
<td>Input:

Rows = 7
Columns = 5
Num_win = 4</td>
<td>Output:

State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
This test case is unique and distinct because I am creating an empty GameBoard of different numbers of rows and columns.

Function Name:
testGameBoardConstructor_diff_num_row_col</td>
</tr>
</table>

`Boolean checkSpace (BoardPosition pos)`

<table>
<tr>
<td>Input:

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>O</td><td></td><td></td></tr>
</table>

pos.getRow() = 1
pos.getColumn() = 0</td>
<td>Output:

checkSpace = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because the space is empty.

Function Name:
testCheckSpace_empty_space</td>
</tr>
<tr>
<td>Input:

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>O</td><td></td><td></td></tr>
</table>

pos.getRow() = 0
pos.getColumn() = 0</td>
<td>Output:

checkSpace = false

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because the space is full and we are checking a unique character.

Function Name:
testCheckSpace_not_space</td>
</tr>
<tr>
<td>Input:

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>O</td></tr>
</table>

pos.getRow() = 1
pos.getColumn() = 0</td>
<td>Output:

checkSpace = false

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because every space is full and we are checking a unique character.

Function Name:
testCheckSpace_full_space</td>
</tr>
</table>

`boolean checkHorizontalWin (BoardPosition lastPos, char player)`

<table>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td>X</td><td>X</td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td></td></tr></table>
lastPos.getRow() = 0
lastPos.getColumn() = 1

player = X</td>
<td>Output:

checkHorizontalWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because there is a horizontal win and our last placed character is in the middle.

Function Name:
testHorizontalWin_is_Win_Add_Middle</td>
</tr>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td>X</td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td></td></tr></table>
lastPos.getRow() = 0
lastPos.getColumn() = 1

player = X</td>
<td>Output:

checkHorizontalWin = false

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because there is not a number in a row that equals the number to win.

Function Name:
testHorizontalWin_no_Win</td>
</tr>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td>X</td><td>X</td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td></td></tr></table>
lastPos.getRow() = 0
lastPos.getColumn() = 2

player = X</td>
<td>Output:

checkHorizontalWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because there is a horizontal win and our last placed token is the rightmost on the win.

Function Name:
testHorizontalWin_is_Win_Add_Right</td>
</tr>
<tr>
<td>Input:

State:</td>
<td>Output:

checkHorizontalWin =</td>
<td>Reason:
This test case is unique and distinct because there is a horizontal win but</td>
</tr>
</table>

<table>
<tr>
<td>Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td>X</td><td>X</td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td></td></tr></table>
lastPos.getRow() = 0
lastPos.getColumn() = 0

player = X</td>
<td>State of the board is unchanged</td>
<td>our last placed token is the leftmost on the win.

Function Name:
testHorizontalWin_is_Win_Add_Left</td>
</tr>
</table>

`boolean checkVerticalWin (BoardPosition lastPos, char player)`

<table>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>O</td><td></td></tr><tr><td>2</td><td>X</td><td>O</td><td></td></tr></table>
lastPos.getRow() = 1
lastPos.getColumn() = 0

player = X</td>
<td>Output:

checkVerticalWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because there is a vertical win and our last placed character is in the middle.

Function Name:
testVerticalWin_is_Win_Add_Middle</td>
</tr>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td></td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td></td></tr></table>
lastPos.getRow() = 1
lastPos.getColumn() = 0

player = X</td>
<td>Output:

checkVerticalWin = false

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because there is not a number in a vertical that equals the number to win.

Function Name:
testVerticalWin_no_Win</td>
</tr>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>O</td><td></td></tr><tr><td>2</td><td>X</td><td>O</td><td></td></tr></table>
lastPos.getRow() = 0
lastPos.getColumn() = 0

player = X</td>
<td>Output:

checkVerticalWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because there is a vertical win and our last placed token is the uppermost on the win.

Function Name:
testVerticalWin_is_Win_Add_Upper</td>
</tr>
<tr>
<td>Input:

State:</td>
<td>Output:

checkVerticalWin = true</td>
<td>Reason:
This test case is unique and distinct because there is a vertical win but</td>
</tr>
</table>

<table>
<tr>
<td>Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>O</td><td></td></tr><tr><td>2</td><td>X</td><td>O</td><td></td></tr></table>
lastPos.getRow() = 2
lastPos.getColumn() = 0

player = X</td>
<td>State of the board is unchanged</td>
<td>our last placed token is the lowermost on the win.

Function Name:
testVerticalWin_is_Win_Add_Lower</td>
</tr>
</table>

Boolean checkDiagonalWin (BoardPosition lastPos, char player)

<table>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td>X</td></tr></table>
lastPos.getRow() = 1
lastPos.getColumn() = 1

player = X</td>
<td>Output:

checkDiagonalWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests the left diagonal and the last placed character is in the middle.

Function Name:
testDiagonallWin_is _Win_Add_Middle_Left_D</td>
</tr>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td>X</td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td></td><td>X</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>
lastPos.getRow() = 0
lastPos.getColumn() = 1

player = X</td>
<td>Output:

checkDiagonalWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests the left diagonal and the last placed character is the upper/leftmost and it is not on the main diagonal.

Function Name:
testDiagonallWin_is _Win_Add_Up/Left_Left_D</td>
</tr>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td>O</td><td>O</td><td>X</td></tr></table>
lastPos.getRow() = 2
lastPos.getColumn() = 2

player = X</td>
<td>Output:

checkDiagonalWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests the left diagonal and the last placed character is the lowest/rightmost.

Function Name:
testDiagonallWin_is _Win_Add_Low/Right_Left_D</td>
</tr>
<tr>
<td>Input:</td>
<td>Output:</td>
<td>Reason:</td>
</tr>
</table>

<table>
<tr>
<td>State:
Num_win = 3

<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td></td><td>X</td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td><td></td></tr><tr><td>2</td><td>X</td><td>O</td><td>O</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>
lastPos.getRow() = 1
lastPos.getColumn() = 1

player = X</td>
<td>checkDiagonalWin = true

State of the board is unchanged</td>
<td>This test case is unique and distinct because it tests the right diagonal and the last placed character is in the middle and it is not on the main diagonal.

Function Name:
testDiagonallWin_is _Win_Add_Middle_Right_D</td>
</tr>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td>X</td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td>X</td><td>O</td><td>O</td></tr></table>
lastPos.getRow() = 0
lastPos.getColumn() = 0

player = X</td>
<td>Output:

checkDiagonalWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests the right diagonal and the last placed character is the upper/rightmost.

Function Name:
testDiagonallWin_is _Win_Add_Up/Right_Right_D</td>
</tr>
<tr>
<td>Input:

State:
Num_win = 3
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td>X</td></tr><tr><td>1</td><td></td><td>X</td><td></td></tr><tr><td>2</td><td>X</td><td>O</td><td>O</td></tr></table>
lastPos.getRow() = 2
lastPos.getColumn() = 0

player = X</td>
<td>Output:

checkDiagonalWin = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests the right diagonal and the last placed character is the lowest/leftmost.

Function Name:
testDiagonallWin_is _Win_Add_Low/Left_Right_D</td>
</tr>
<tr>
<td>Input:

State:
Num_win = 3</td>
<td>Output:

checkDiagonalWin = false</td>
<td>Reason:
This test case is unique and distinct because it tests the left</td>
</tr>
</table>

	0	1	2
0	X		
1		X	
2	O	O	O

lastPos.getRow() = 2
lastPos.getColumn() = 2

player = O

State of the board is unchanged

diagonal and there is no diagonal win.
Function Name:
testDiagonallWin_no_Win

`boolean checkForDraw ()`

- Create 4 distinct test cases

<table>
<tr>
<td>Input:

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>O</td><td>X</td><td>O</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td></tr>
</table>
</td>
<td>Output:

checkForDraw = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a full board on the minimum board size.

Function Name:
testCheckForDraw_is_Draw</td>
</tr>
<tr>
<td>Input:

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table>
</td>
<td>Output:

checkForDraw = false

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests an empty board.

Function Name:
testCheckForDraw_is_Empty</td>
</tr>
<tr>
<td>Input:

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>O</td><td>X</td><td>O</td></tr>
<tr><td>1</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td>O</td><td>O</td><td>X</td></tr>
</table>
</td>
<td>Output:

checkForDraw = false

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a partially full board.

Function Name:
testCheckForDraw_is_Partially_Full</td>
</tr>
<tr>
<td>Input:

State:

100x100 board that is full</td>
<td>Output:

checkForDraw = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a full board on the maximum board size.

Function Name:
testCheckForDraw_is_Full_Max_Board</td>
</tr>
</table>

`Char whatsAtPos (BoardPosition pos)`

- Create 5 distinct test cases

<table>
<tr>
<td>Input:

Pos.getRow() = 0;
Pos.getColumn() = 0

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output:

checkWhatsAtPos = ‘ ‘

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a character that is empty.

Function Name:
testWhatsAtPos_is_Empty</td>
</tr>
<tr>
<td>Input:

Pos.getRow() = 0;
Pos.getColumn() = 0

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output:

checkWhatsAtPos = X

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a unique character.

Function Name:
testWhatsAtPos_is_X_Unique_Char</td>
</tr>
<tr>
<td>Input:

Pos.getRow() = 0;
Pos.getColumn() = 0

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output:

checkWhatsAtPos = O

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a unique character.

Function Name:
testWhatsAtPos_is_O_Dist_Char</td>
</tr>
<tr>
<td>Input:

Pos.getRow() = 2
Pos.getColumn() = 2

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td>X</td></tr></table></td>
<td>Output:

checkWhatsAtPos = X

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a unique character at the bottom left corner.

Function Name:
testWhatsAtPos_is_Dist_Char_Lower_Corner</td>
</tr>
</table>

<table>
<tr>
<td>Input:

Pos.getRow() = 0
Pos.getColumn() = 2

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>1</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>X</td><td>X</td></tr>
</table>
</td>
<td>Output:

checkWhatsAtPos = O

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests a unique character on a completely full board in the upper right corner.

Function Name:
testWhatsAtPos_is
_Dist_Char_Full_Upper_Right</td>
</tr>
</table>

`Boolean isPlayerAtPos (BoardPosition pos, char player)`

- Create 5 distinct test cases

<table>
<tr>
<td>Input:

Pos.getRow() = 0;
Pos.getColumn() = 0

Player = X

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output:

checkIsPlayerAtPos = false

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests an empty board with a nonempty player character. It compares a nonwhite space to a whitespace.

Function Name:
testisPlayerAtPos_is _empty_vs_Unique_player</td>
</tr>
<tr>
<td>Input:

Pos.getRow() = 0;
Pos.getColumn() = 0

Player = ‘ ‘

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output:

checkIsPlayerAtPos = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests an empty board with an empty player character.

Function Name:
testisPlayerAtPos_is _empty_vs_empty_player</td>
</tr>
<tr>
<td>Input:

Pos.getRow() = 0;
Pos.getColumn() = 0

Player = X

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output:

checkIsPlayerAtPos = true

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests an non empty position on the board with the correct player character.

Function Name:
testisPlayerAtPos_is _true</td>
</tr>
<tr>
<td>Input:

Pos.getRow() = 0;
Pos.getColumn() = 0</td>
<td>Output:

checkIsPlayerAtPos = false

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it tests an non empty position on the board</td>
</tr>
</table>

<table>
<tr>
<td>Player = X

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td></td>
<td>with the incorrect player character.

Function Name:
testisPlayerAtPos_is
_false</td>
</tr>
<tr>
<td>Input:

Pos.getRow() = 1
Pos.getColumn() = 1

Player = A

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>O</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output:

checkIsPlayerAtPos = false

State of the board is unchanged</td>
<td>Reason:
This test case is unique and distinct because it compares a new character that has not been placed on the board.

Function Name:
testisPlayerAtPos_is
_new_Char</td>
</tr>
</table>

`Void placeMarker (BoardPosition marker, char player)`

- Create 5 distinct test cases

<table>
<tr>
<td>Input:

marker.getRow() = 0;
marker.getColumn() = 0

Player = X

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output:

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
This test case is unique and distinct because it tests an empty board with a unique character.

Function Name:
testPlaceMarker_Empty</td>
</tr>
<tr>
<td>Input:

marker.getRow() = 2
marker.getColumn() = 0

Player = O

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td>X</td><td>O</td></tr><tr><td>1</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td></td><td>X</td><td>X</td></tr></table></td>
<td>Output:

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td>X</td><td>O</td></tr><tr><td>1</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>O</td><td>X</td><td>X</td></tr></table></td>
<td>Reason:
This test case is unique and distinct because it tests an almost full board with a unique character on the leftmost lower position.

Function Name:
testPlaceMarker_Almost_Full</td>
</tr>
<tr>
<td>Input:

marker.getRow() = 2
marker.getColumn() = 0

Player = A

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table></td>
<td>Output:

State:

<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>O</td><td></td><td></td></tr><tr><td>1</td><td>X</td><td></td><td></td></tr><tr><td>2</td><td>A</td><td></td><td></td></tr></table></td>
<td>Reason:
This test case is unique and distinct because I am placing a marker representing a player who has not been placed on this board before.

Function Name:
testPlaceMarker_New_Char</td>
</tr>
<tr>
<td>Input:

marker.getRow() = 0
marker.getColumn() = 2</td>
<td>Output:

State:</td>
<td>Reason:
This test case is unique and distinct because I am placing a</td>
</tr>
</table>

<table>
<tr>
<td>Player = A

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>O</td><td></td><td></td></tr>
<tr><td>1</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table>
</td>
<td>
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>O</td><td></td><td>A</td></tr>
<tr><td>1</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table>
</td>
<td>unique marker in the upper right corner of the board.

Function Name:
testPlaceMarker_Upper_Right</td>
</tr>
<tr>
<td>Input:

marker.getRow() = 2
marker.getColumn() = 2

Player = A

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>O</td><td></td><td></td></tr>
<tr><td>1</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table>
</td>
<td>Output:

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>O</td><td></td><td></td></tr>
<tr><td>1</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>A</td></tr>
</table>
</td>
<td>Reason:
This test case is unique and distinct because I am placing a unique marker in the lower right corner of the board.

Function Name:
testPlaceMarker_Lower_Right</td>
</tr>
</table>