Project 5 CPSC 2150 Fall 2020
Kevin Mody

Testing:
public GameBoard(int r, int c, int w) and public GameBoardMem(int r, int c, int w)

Input	Output	Reason and Function Name

IGameBoard nb = new GameBoard(8, 8, 5);
 	
char[][] ab = new char[8][8];
for (int i = 0; i < 8; i++) {
for (int j = 0; j < 8; j++) {
ab[i][j] = ' ';
}
}
	nb.toString() = printBoard(ab)	This test case is special and distinct due to the fact it is somewhere in between the minimal and most board size constraints.
		
		testConstruct1

IGameBoard nb = new GameBoard(3, 3, 3);
char[][] ab = new char[3][3];
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
ab[i][j] = ' ';
}
}
	nb.toString() = printBoard(ab)	This test case is special and distinct because it is at the minimum of the board size constraints.
		
		testConstruct2

IGameBoard nb = new GameBoard(99, 99, 25);
char[][] ab = new char[99][99];
for (int i = 0; i < 99; i++) {
for (int j = 0; j < 99; j++) {
ab[i][j] = ' ';
}
}
	nb.toString() = printBoard(ab)	This test case is special and distinct because it is at the maximum of the board size constraints.
		
		testConstruct3

default public boolean checkSpace(BoardPosition pos)

Input	Output	Reason and Function Name

BoardPosition pos = new BoardPosition(0, 0);
 	
gb.placeMarker(pos, 'X');
	checkSpace(pos) = false	This test case is unique and distinct because it is testing the detection of a player's character at a space they have occupied.
		
		testCheckTakenSpace

BoardPosition pos = new BoardPosition(0, 0);	checkSpace(pos) = true;	This test case is unique and distinct because it is testing the detection of a blank character at an empty and available space. testCheckEmptySpace

default public boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input	Output	Reason and Function Name

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(2, 3);
BoardPosition pos3 = new BoardPosition(2, 4);
BoardPosition pos4 = new BoardPosition(2, 5);
BoardPosition pos5 = new BoardPosition(2, 6);
 	
gb.placeMarker(pos1, 'X');
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos3, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5,
	checkHorizontalWin(pos5, 'X') = true	This test case is unique and distinct because it tests the detection of a horizontal win when the last piece needed to win is placed to the right of consecutive tokens.
		
		testHorizontalWin1

'X');		

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(2, 3);
BoardPosition pos3 = new BoardPosition(2, 4);
BoardPosition pos4 = new BoardPosition(2, 5);
BoardPosition pos5 = new BoardPosition(2, 6);
 	
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos3, 'X');os
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
gb.placeMarker(pos1, 'X');
	checkHorizontalWin(pos5, ‘X’) = true	This test case is unique and distinct because it tests the detection of a horizontal win when the last piece needed to win is placed to the left of consecutive tokens.
		
		testHorizontalWin2

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(2, 3);
BoardPosition pos3 = new BoardPosition(2, 4);
BoardPosition pos4 = new BoardPosition(2, 5);
BoardPosition pos5 = new BoardPosition(2, 6);
 	
gb.placeMarker(pos1, 'X');
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
gb.placeMarker(pos3, 'X');
	checkHorizontalWin(pos3, ‘X’) = true	This test case is unique and distinct because it tests the detection of a horizontal win when the last piece needed to win is placed between two sets of two consecutive tokens.
		
		testHorizontalWin3

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(2, 3);
	checkHorizontalWin(pos5, ‘X’) = false	This test case is unique and distinct because it tests for mistakenly detecting a win when the

BoardPosition pos3 = new BoardPosition(2, 4);
BoardPosition pos4 = new BoardPosition(2, 5);
BoardPosition pos5 = new BoardPosition(2, 7);
 	
gb.placeMarker(p1, 'X');
gb.placeMarker(p2, 'X');
gb.placeMarker(p3, 'X');
gb.placeMarker(p4, 'X');
gb.placeMarker(p5, 'X');
		last piece needed to win is placed in the same row but separated by a space.
		
		testHorizontalWin4

default public boolean checkVerticalWin(BoardPosition lastPos, char player)

Input	Output	Reason and Function Name

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(3, 2);
BoardPosition pos3 = new BoardPosition(4, 2);
BoardPosition pos4 = new BoardPosition(5, 2);
BoardPosition pos5 = new BoardPosition(6, 2);
 	
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos3, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
gb.placeMarker(pos1, 'X');
	checkVerticalWin(pos1, ‘X’) = true	This test case is unique and distinct because it tests the detection of a win when the last piece needed to win is placed above consecutive tokens.
		
		testVerticalWin1

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(3, 2);
BoardPosition pos3 = new BoardPosition(4, 2);
BoardPosition pos4 = new BoardPosition(5, 2);
BoardPosition pos5 =
	checkVerticalWin(pos5, ‘X’) = true	This test case is unique and distinct because it tests the detection of a win when the last piece needed to win is placed below consecutive tokens.

new BoardPosition(6, 2);
 	
gb.placeMarker(pos1, 'X');
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos3, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
		testVerticalWin2

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(3, 2);
BoardPosition pos3 = new BoardPosition(4, 2);
BoardPosition pos4 = new BoardPosition(5, 2);
BoardPosition pos5 = new BoardPosition(6, 2);
 	
gb.placeMarker(pos1, 'X');
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
gb.placeMarker(pos3, 'X');
	checkVerticalWin(pos3, ‘X’) = true	This test case is unique and distinct because it tests the detection of a win when the last piece needed to win is placed between two sets of consecutive tokens.
		
		testVerticalWin3

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(3, 2);
BoardPosition pos3 = new BoardPosition(4, 2);
BoardPosition pos4 = new BoardPosition(5, 2);
BoardPosition pos5 = new BoardPosition(7, 2);
 	
gb.placeMarker(pos1, 'X');
gb.placeMarker(pos2, 'X');
	checkVerticalWin(pos5, ‘X’) = false	This test case is unique and distinct because it tests for mistakenly detecting a win when the last piece needed to win is placed in the same row but separated by a space.
		
		testVerticalWin4

gb.placeMarker(pos3, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');		

default public boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason and Function Name

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(3, 3);
BoardPosition pos3 = new BoardPosition(4, 4);
BoardPosition pos4 = new BoardPosition(5, 5);
BoardPosition pos5 = new BoardPosition(6, 6);
 	
gb.placeMarker(pos1, 'X');
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos3, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
	checkDiagonalWin(pos5, ‘X’) = true	This test case is unique and distinct because it tests for the detection of a win with the left diagonal when the last piece is placed below the others in the sequence.
		
		testDiagonalWin1

BoardPosition pos1 = new BoardPosition(2, 6);
BoardPosition pos2 = new BoardPosition(3, 5);
BoardPosition pos3 = new BoardPosition(4, 4);
BoardPosition pos4 = new BoardPosition(5, 3);
BoardPosition pos5 = new BoardPosition(6, 2);
 	
gb.placeMarker(pos1, 'X');
gb.placeMarker(pos2, 'X');
	checkDiagonalWin(pos5, ‘X’) = true	This test case is unique and distinct because it tests for the detection of a win with the right diagonal when the last piece is placed above the others in the sequence.
		
		testDiagonalWin2

gb.placeMarker(pos3, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');		

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(3, 3);
BoardPosition pos3 = new BoardPosition(4, 4);
BoardPosition pos4 = new BoardPosition(5, 5);
BoardPosition pos5 = new BoardPosition(6, 6);
 	
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos3, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
gb.placeMarker(pos1, 'X');
	checkDiagonalWin(pos5, ‘X’) = true	This test case is unique and distinct because it tests for the detection of a win with the left diagonal when the last piece is placed above the others in the sequence.
		
		testDiagonalWin3

BoardPosition pos1 = new BoardPosition(2, 6);
BoardPosition pos2 = new BoardPosition(3, 5);
BoardPosition pos3 = new BoardPosition(4, 4);
BoardPosition pos4 = new BoardPosition(5, 3);
BoardPosition pos5 = new BoardPosition(6, 2);
 	
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos3, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
gb.placeMarker(pos1, 'X');
	checkDiagonalWin(pos1, ‘X’) = true	This test case is unique and distinct because it tests for the detection of a win with the right diagonal when the last piece is placed below the others in the sequence.
		
		testDiagonalWin4

BoardPosition pos1 = new BoardPosition(2, 2);
BoardPosition pos2 = new BoardPosition(3, 3);
BoardPosition pos3 = new BoardPosition(4, 4);
BoardPosition pos4 = new BoardPosition(5, 5);
BoardPosition pos5 = new BoardPosition(7, 7);
 	
gb.placeMarker(pos1, 'X');
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos3, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
	checkDiagonalWin(pos5, ‘X’) = false	This test case is unique and distinct because it tests for the mistaken detection of a win when the last piece to win is placed in the same left diagonal as a consecutive sequence but with a space separating it.
		
		testDiagonalWin5

BoardPosition pos1 = new BoardPosition(2, 6);
BoardPosition pos2 = new BoardPosition(3, 5);
BoardPosition pos3 = new BoardPosition(4, 4);
BoardPosition pos4 = new BoardPosition(5, 3);
BoardPosition pos5 = new BoardPosition(7, 1);
 	
gb.placeMarker(p1, 'X');
gb.placeMarker(p2, 'X');
gb.placeMarker(p3, 'X');
gb.placeMarker(p4, 'X');
gb.placeMarker(p5, 'X');
	checkDiagonalWin(pos5, ‘X’) = false	This test case is unique and distinct because it tests for the mistaken detection of a win when the last piece to win is placed in the same right diagonal as a consecutive sequence but with a space separating it.
		
		testDiagonalWin6

BoardPosition pos1 = new BoardPosition(2, 6);
BoardPosition pos2 = new BoardPosition(3, 5);
BoardPosition pos4 = new BoardPosition(5, 3);
BoardPosition pos5 = new BoardPosition(7, 1);
BoardPosition pos6 = new BoardPosition(2, 2);
BoardPosition pos7 =
	checkDiagonalWin(pos9, ‘X’) = false	This test case is unique and distinct because it tests for the mistaken detection of a win when there is an empty space left between two crossing diagonals.
		
		testDiagonalWin7

new BoardPosition(3,3);
BoardPosition pos8 = new BoardPosition(5,5);
BoardPosition pos9 = new BoardPosition(6, 6);
 	
gb.placeMarker(pos1, 'X');
gb.placeMarker(pos2, 'X');
gb.placeMarker(pos4, 'X');
gb.placeMarker(pos5, 'X');
gb.placeMarker(pos6, 'X');
gb.placeMarker(pos7, 'X');
gb.placeMarker(pos8, 'X');
gb.placeMarker(pos9, 'X');		

public boolean checkForDraw()

Input	Output	Reason and Function Name

for(int i = 0; i<8; i++){
for(int j = 0; j<8; j++){
BoardPosition temp = new BoardPosition(i, j);
 	
gb.placeMarker(temp, 'X');
}
}
	gb.checkForDraw() = false;	This test case is unique and distinct because it tests for the mistaken detection of a draw when the board is filled with a single character throughout, which should be winning instead.
		
		testCheckForDraw1

for(int i = 0; i<8; i++){
for(int j = 0; j<8; j++){
BoardPosition temp = new BoardPosition(i, j);
if(i%2==0) {
if(j%2==0) {
	gb.checkForDraw() = true	This test case is unique and distinct because it tests for the detection of a draw when the board is filled with alternating characters so that no wins are

gb.placeMarker(temp, 'X');
}
else{
 	
gb.placeMarker(temp, 'O');
}
}
else{
if(j%2==0){
 	
gb.placeMarker(temp, 'O');
}
else{
 	
gb.placeMarker(temp, 'X');
}
}
}
}
		present.
		
		testCheckForDraw2

BoardPosition p = new BoardPosition(3, 3);
gb.placeMarker(p);
	gb.checkForDraw() = false	This test case is unique and distinct because it tests for mistaken detection of a draw when there is only one marker on the board.
		
		testCheckForDraw3

for(int i = 0; i<7; i++){
for(int j = 0; j<8; j++){
BoardPosition temp = new BoardPosition(i, j);
 	
gb.placeMarker(temp, 'X');
}
}
for(int k = 0; k<7; k++){
BoardPosition t = new BoardPosition(7, k);
gb.placeMarker(t, 'X');
}
	gb.checkForDraw() = false	This test case is unique and distinct because it tests for mistaken detection of a draw when there is one blank space.
		
		testCheckForDraw4

public char whatsAtPos(BoardPosition pos)

Input	Output	Reason and Function Name
BoardPosition p = new BoardPosition(0,0); gb.placeMarker(pos, 'X');	whatsAtPos(p) = ‘X’	This test case is unique and distinct because it tests the detection of X at position <0,0> after it has been marked there. testWhatsAtPos1
BoardPosition p= new BoardPosition(0,0);	whatsAtPos(p) = ‘ ‘	This test case is unique and distinct because it tests the detection of a blank character at an unmarked position. testWhatsAtPos2
BoardPosition p = new BoardPosition(0,0); gb.placeMarker(p, 'X');	whatsAtPos(p) != ‘ ‘	This test case is unique and distinct because it tests for the mistaken detection of a blank character at an marked position. testWhatsAtPos3
BoardPosition p = new BoardPosition(0,0);	whatsAtPos(p) != ‘X’	This test case is unique and distinct because it tests for the mistaken detection of a character at an unmarked position. testWhatsAtPos4
BoardPosition p = new BoardPosition(0,0); gb.placeMarker(p, 'O');	whatsAtPos(p) != ‘X’	This test case is unique and distinct because it tests for the mistaken detection of a character where another one has been marked.

		testWhatsAtPos5

default public boolean isPlayerAtPos(BoardPosition pos, char player)

Input	Output	Reason and Function Name

BoardPosition p = new BoardPosition(1, 1);
gb.placeMarker(p, 'O');
	isPlayerAtPos(p, ‘O’) = true	This test case is unique and distinct because it tests the detection of a character in a place it has been marked.
		
		testIsPlayerAtPos1

BoardPosition p = new BoardPosition(1, 1);	isPlayerAtPos(p, ‘O’); = false	This test case is unique and distinct because it tests for the mistaken detection of a character in an empty space. testIsPlayerAtPos2

BoardPosition p = new BoardPosition(1, 1);
gb.placeMarker(p, 'X');
	isPlayerAtPos(p, ‘O’) = false	This test case is unique and distinct because it tests for accuracy of character detection when given a character different from the one occupying the given space.
		
		testIsPlayerAtPos3

BoardPosition p1 = new BoardPosition(1, 1);
BoardPosition p2 = new BoardPosition(3, 1);
BoardPosition p3 = new BoardPosition(2, 1);
 	
gb.placeMarker(p1, 'X');
gb.placeMarker(p2, 'X');
	isPlayerAtPos(pos3, ‘X’) = false	This test case is unique and distinct because it tests for mistaken detection of a character in an empty space between two that it occupies.
		
		testIsPlayerAtPos4

BoardPosition p1 = new BoardPosition(1, 1);
        BoardPosition p2 = new BoardPosition(1, 3);
        BoardPosition p3 = new BoardPosition(1, 2);
 	
        gb.placeMarker(p1, 'X');
        gb.placeMarker(p2, 'X');
        gb.placeMarker(p3, 'O');
	sPlayerAtPos(pos3, ‘X’) = false	This test case is unique and distinct because it tests for mistaken detection of a character in s space occupied by a different character between two occupied by the tested character.
		
		testIsPlayerAtPos5

public void placeMarker(BoardPosition marker, char player)

Input	**Output**	**Reason and Function Name**

char[][] a = new char[8][8];
        for (int i = 0; i<8; i++) {
            for (int j = 0; j < 8; j++) {
                a[i][j] = ' ';
            }
        }
        a[0][0] = 'X';
 	
        BoardPosition pos = new BoardPosition(0, 0);
        gb.placeMarker(pos, 'X');
	printBoard(a) = gb.toString()	This test case is unique and distinct because it tests placing a marker in the top left corner or a normal sized board.
		
		testPlaceMarker1

char[][] a = new char[8][8];
        for (int i = 0; i<8; i++) {
            for (int j = 0; j < 8; j++) {
                a[i][j] = ' ';
            }
        }
        a[7][7] = 'X';
	printBoard(a) = gb.toString()	This test case is unique and distinct because it tests placing a marker in the top left corner or a normal sized board.
		
		testPlaceMarker2

char[][] a = new char[8][8];
        for (int i = 0; i<8; i++) {
            for (int j = 0; j < 8; j++) {
                a[i][j] = ' ';
            }
	printBoard(a) = gb.toString()	This test case is unique and distinct because it tests placing a marker in the bottom left corner or a normal sized

}
a[0][7] = 'X';
 	
BoardPosition pos = new BoardPosition(0, 7);
gb.placeMarker(pos, 'X');
		board.
		
		testPlaceMarker3

char[][] a = new char[8][8];
for (int i = 0; i<8; i++) {
for (int j = 0; j < 8; j++) {
a[i][j] = ' ';
}
}
a[7][0] = 'X';
 	
BoardPosition pos = new BoardPosition(7, 0);
gb.placeMarker(pos, 'X');
	printBoard(a) = gb.toString()	This test case is unique and distinct because it tests placing a marker in the top right corner or a normal sized board.
		
		testPlaceMarker4

char[][] a = new char[8][8];
for (int i = 0; i<8; i++) {
for (int j = 0; j < 8; j++) {
a[i][j] = ' ';
}
}
a[3][3] = 'X';
 	
BoardPosition pos = new BoardPosition(7, 0);
gb.placeMarker(pos, 'X');
	printBoard(a) = gb.toString()	This test case is unique and distinct because it tests placing a marker somewhere away from the board boundaries.
		
		testPlaceMarker5

Requirements Analysis:

Functional Requirements (User Stories):

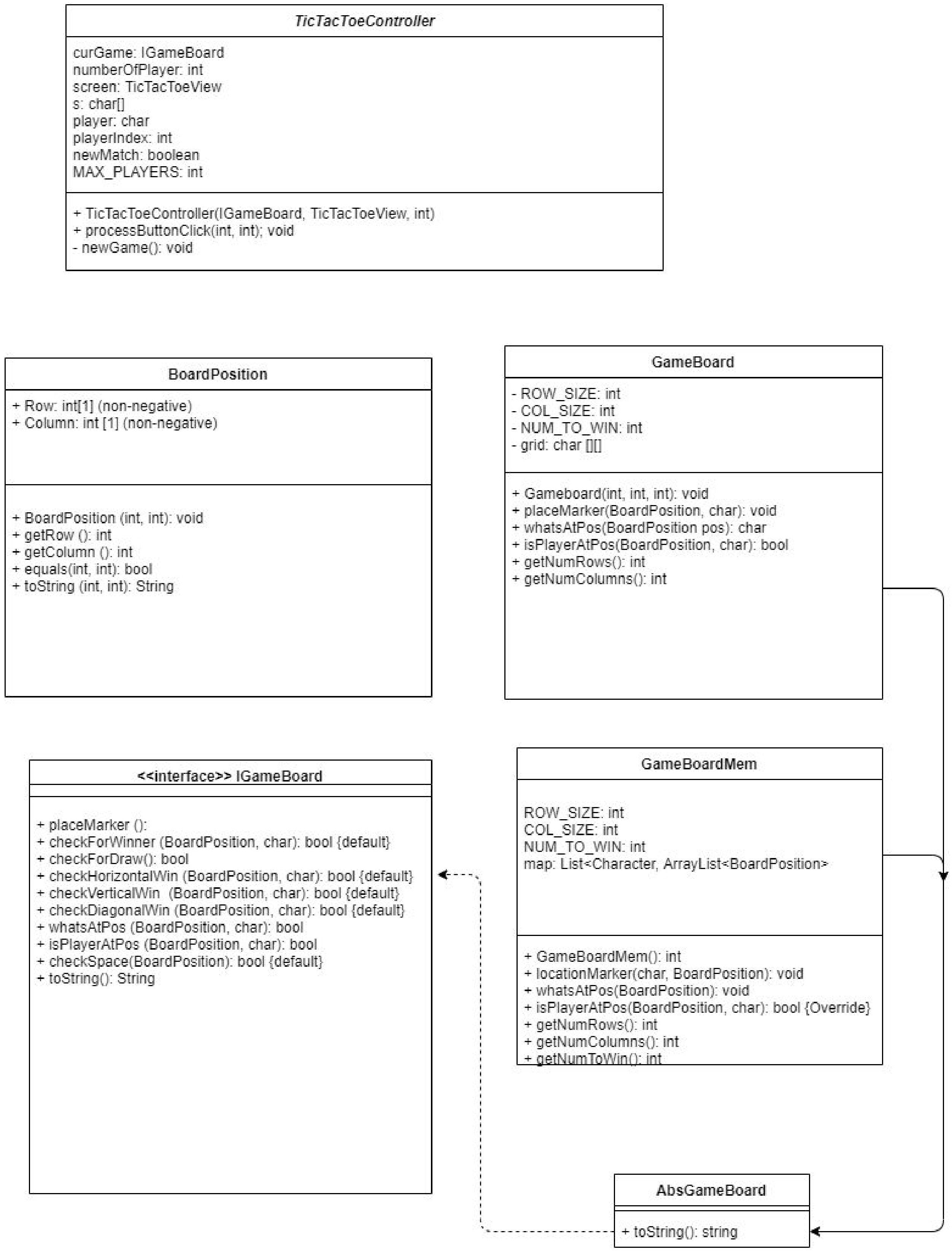
1. As the client, I can click on the gameboard where I want my character to be.
2. As the client, I can enter a number from 2 to 10 to set up the quantity of players.
3. As the client, I can click what position that I want my character inside the game board.
4. As a user, I can have upto 10 players to play with.
5. As a client, I can have row size and column size upto 20 and at least 3.
6. As a user, I can replay the game after draw or win.

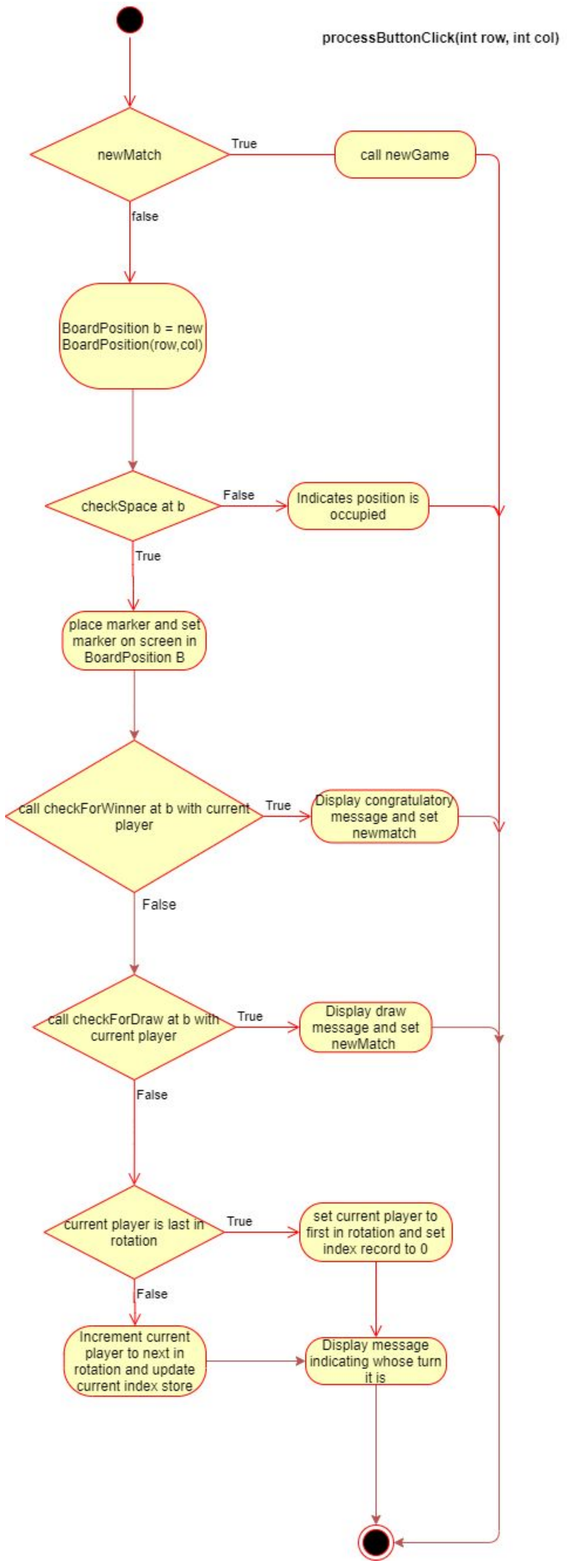
Non-Functional Requirements:

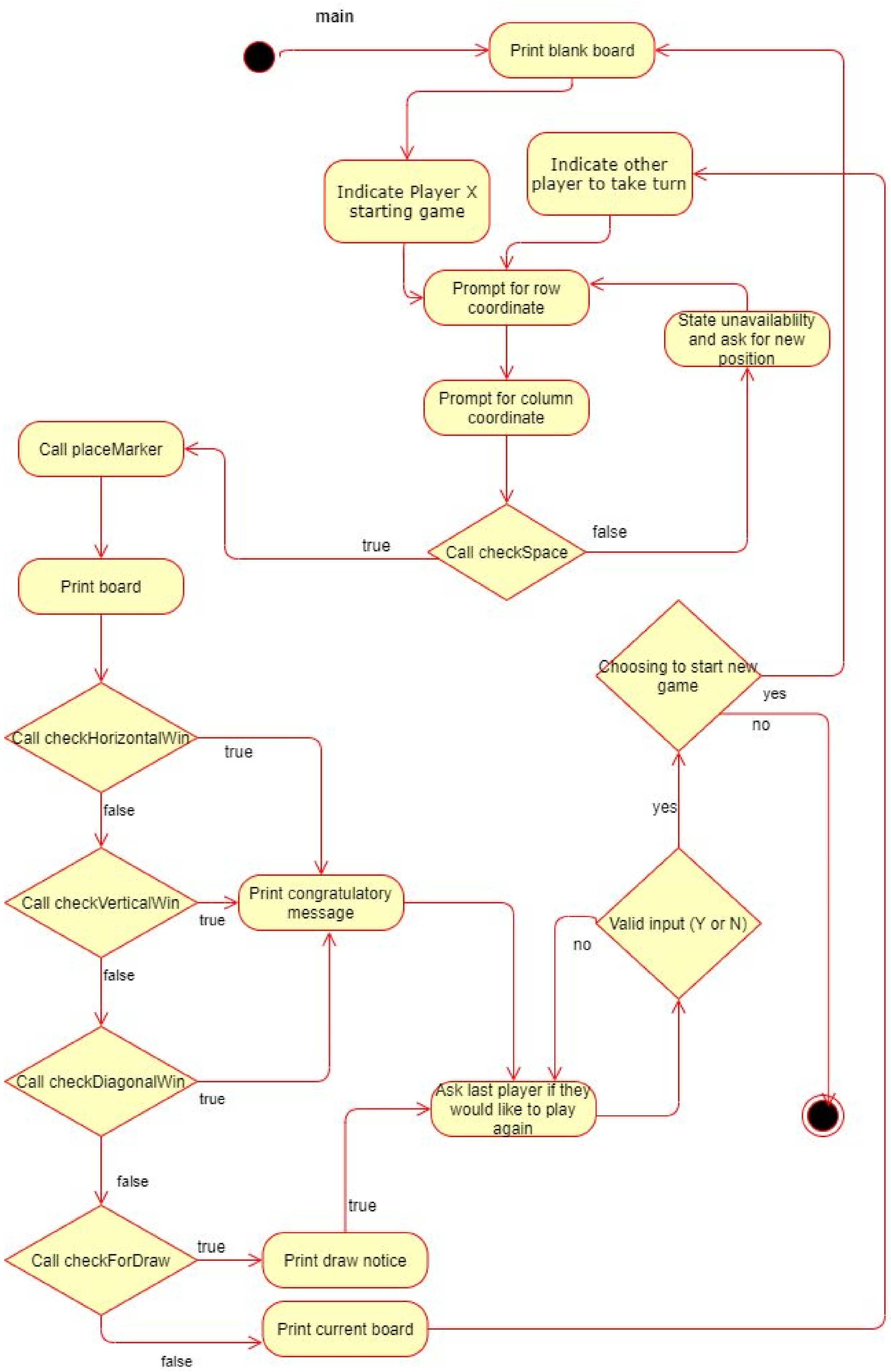
1. The framework must be coded in Java.
2. The framework must have the option to run on Unix/Linux, Windows, and MacOS.
3. Time for placing markers, changing turns, and detecting wins or draws should be quick.

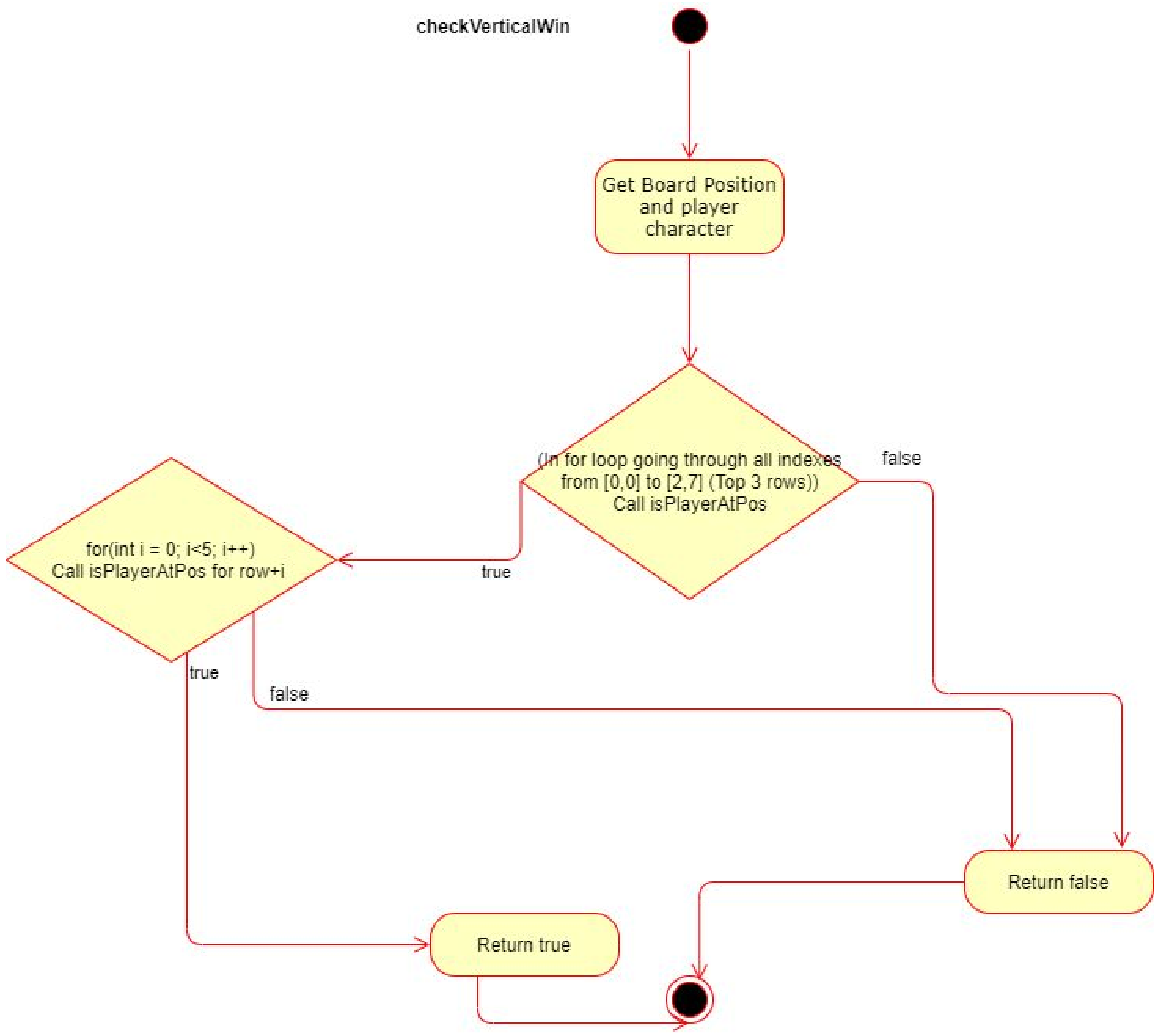
Design:

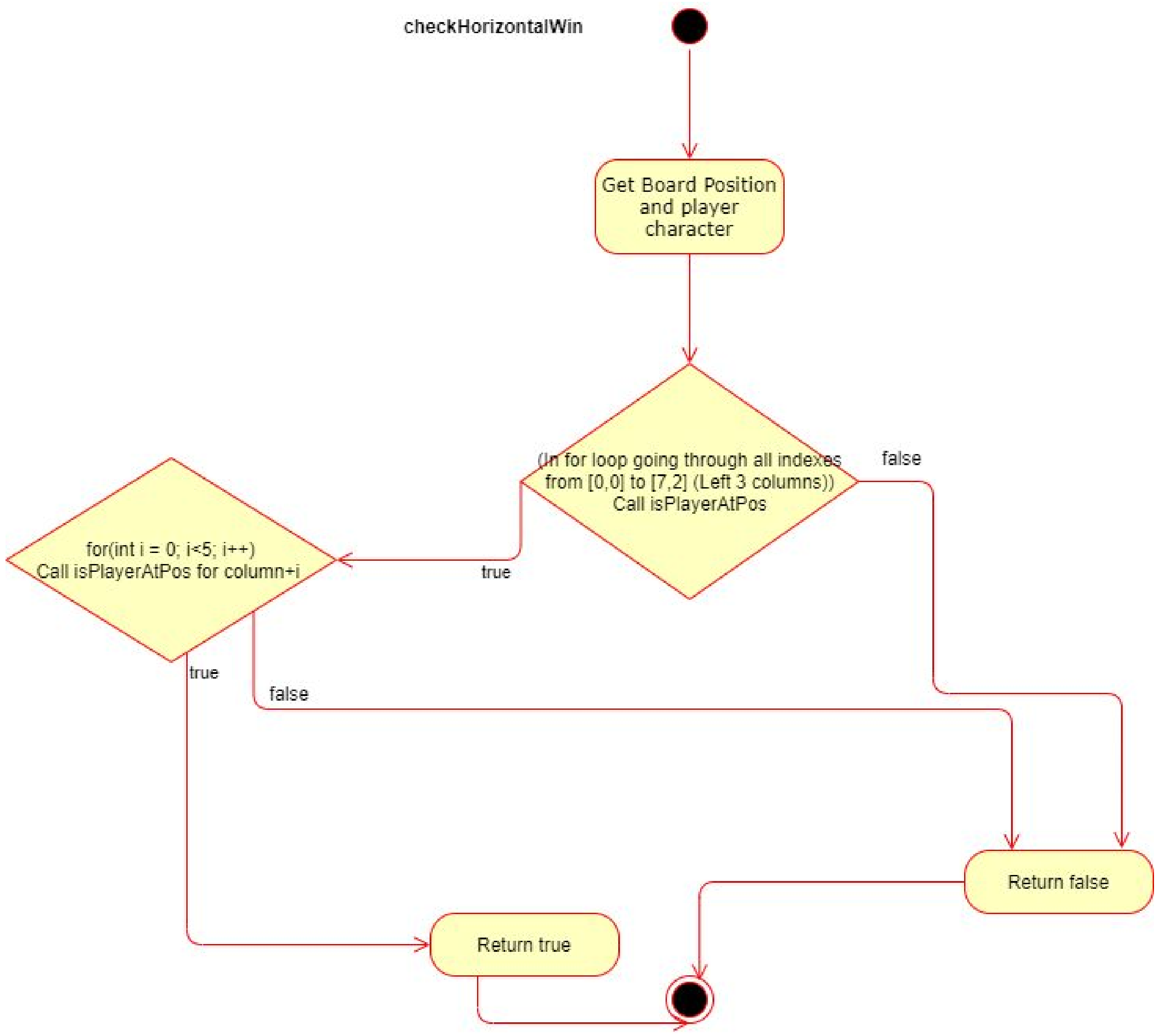
Project 5 classes

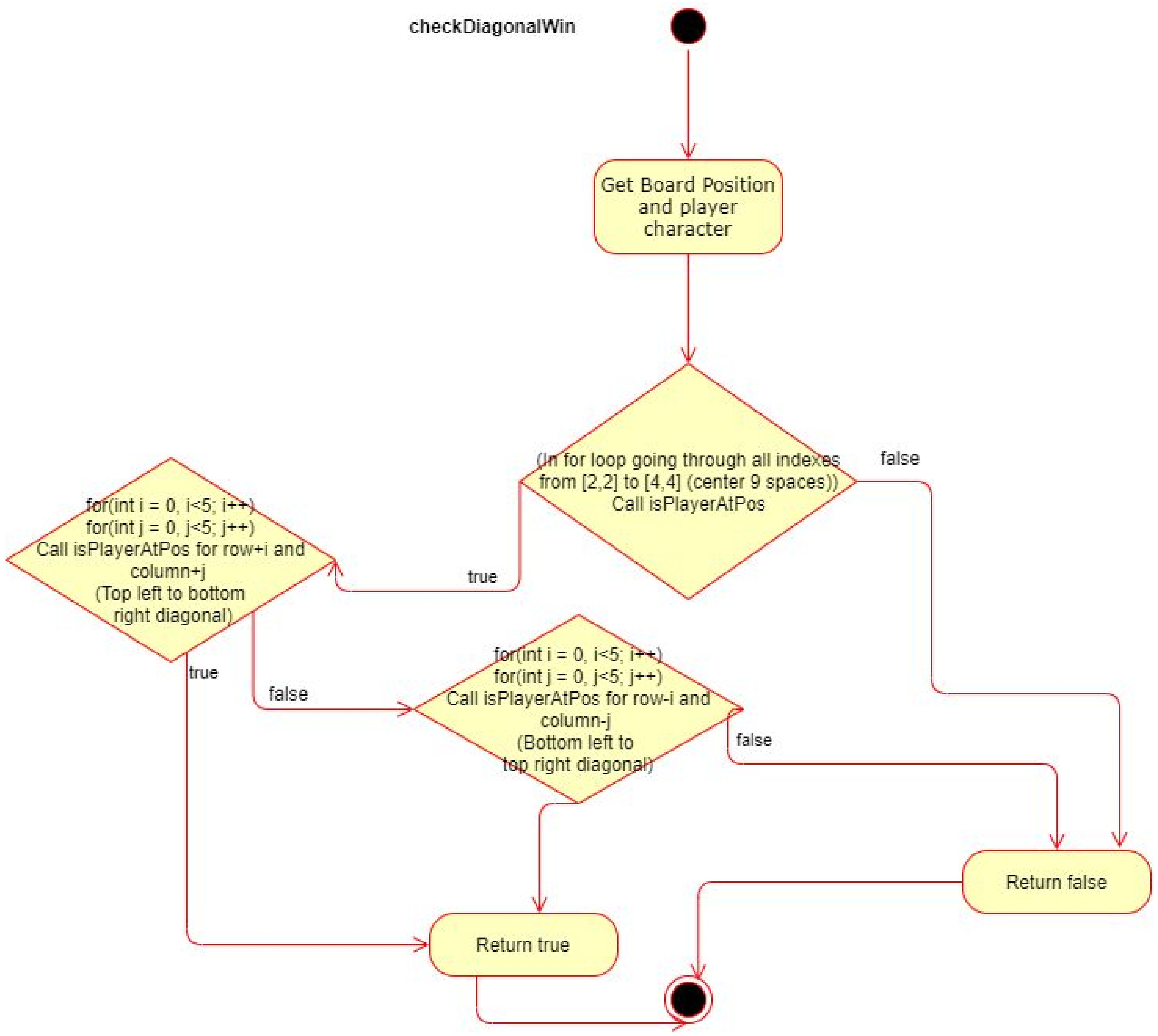

TicTacToeController
curGame: IGameBoard
numberOfPlayer: int
screen: TicTacToeView
s: char[]
player: char
playerIndex: int
newMatch: boolean
MAX_PLAYERS: int
+ TicTacToeController(IGameBoard, TicTacToeView, int)
+ processButtonClick(int, int); void
- newGame(): void
BoardPosition
+ Row: int[1] (non-negative)
+ Column: int [1] (non-negative)
+ BoardPosition (int, int): void
+ getRow (): int
+ getColumn (): int
+ equals(int, int): bool
+ toString (int, int): String
GameBoard
- ROW_SIZE: int
- COL_SIZE: int
- NUM_TO_WIN: int
- grid: char [][]
+ Gameboard(int, int, int): void
+ placeMarker(BoardPosition, char): void
+ whatsAtPos(BoardPosition pos): char
+ isPlayerAtPos(BoardPosition, char): bool
+ getNumRows(): int
+ getNumColumns(): int
<<interface>> IGameBoard
+ placeMarker ():
+ checkForWinner (BoardPosition, char): bool {default}
+ checkForDraw(): bool
+ checkHorizontalWin (BoardPosition, char): bool {default}
+ checkVerticalWin (BoardPosition, char): bool {default}
+ checkDiagonalWin (BoardPosition, char): bool {default}
+ whatsAtPos (BoardPosition, char): bool
+ isPlayerAtPos (BoardPosition, char): bool
+ checkSpace(BoardPosition): bool {default}
+ toString(): String
GameBoardMem
ROW_SIZE: int
COL_SIZE: int
NUM_TO_WIN: int
map: List<Character, ArrayList<BoardPosition>
+ GameBoardMem(): int
+ locationMarker(char, BoardPosition): void
+ whatsAtPos(BoardPosition): void
+ isPlayerAtPos(BoardPosition, char): bool {Override}
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
AbsGameBoard
+ toString(): string


processButtonClick(int row, int col)
newMatch
True
call newGame
false
BoardPosition b = new BoardPosition(row,col)
checkSpace at b
False
Indicates position is occupied
True
place marker and set marker on screen in BoardPosition B
call checkForWinner at b with current player
True
Display congratulatory message and set newmatch
False
call checkForDraw at b with current player
True
Display draw message and set newMatch
False
current player is last in rotation
True
set current player to first in rotation and set index record to 0
False
Increment current player to next in rotation and update current index store
Display message indicating whose turn it is

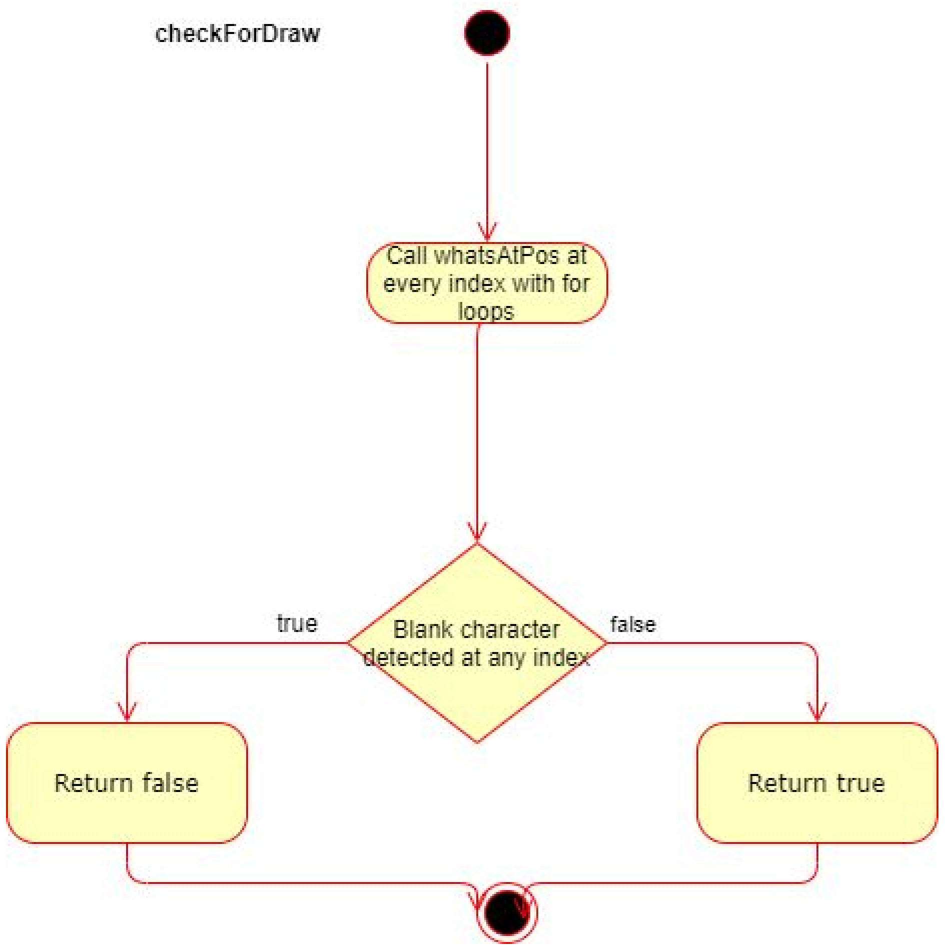

main
Print blank board
Indicate Player X starting game
Indicate other player to take turn
Prompt for row coordinate
State unavailablilty and ask for new position
Prompt for column coordinate
Call placeMarker
Call checkSpace
true
false
Print board
Choosing to start new game
yes
no
Call checkHorizontalWin
true
false
Call checkVerticalWin
true
Print congratulatory message
Valid input (Y or N)
yes
no
false
Call checkDiagonalWin
true
Ask last player if they would like to play again
false
true
Call checkForDraw
true
Print draw notice
Print current board
false

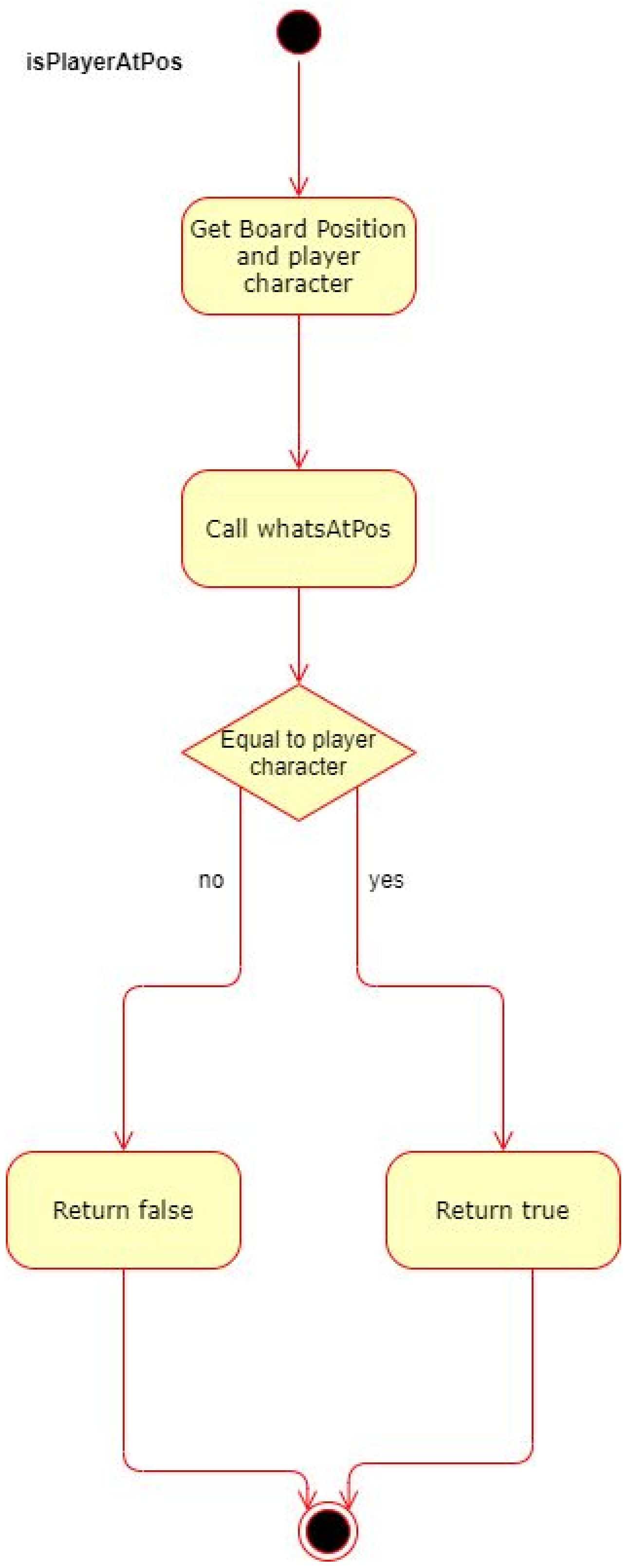

checkVerticalWin
Get Board Position and player character
(In for loop going through all indexes from [0,0] to [2,7] (Top 3 rows)) Call isPlayerAtPos
false
true
for(int i = 0; i<5; i++) Call isPlayerAtPos for row+i
true
false
Return false
Return true

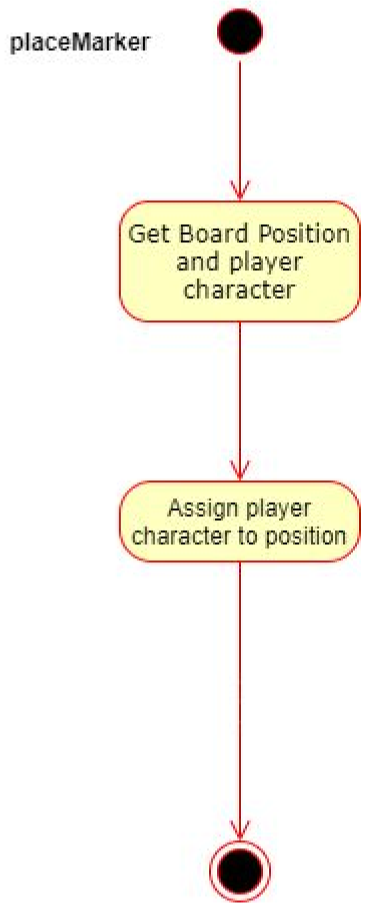

checkHorizontalWin
Get Board Position and player character
(In for loop going through all indexes from [0,0] to [7,2] (Left 3 columns)) Call isPlayerAtPos
false
true
for(int i = 0; i<5; i++) Call isPlayerAtPos for column+i
true
false
Return false
Return true

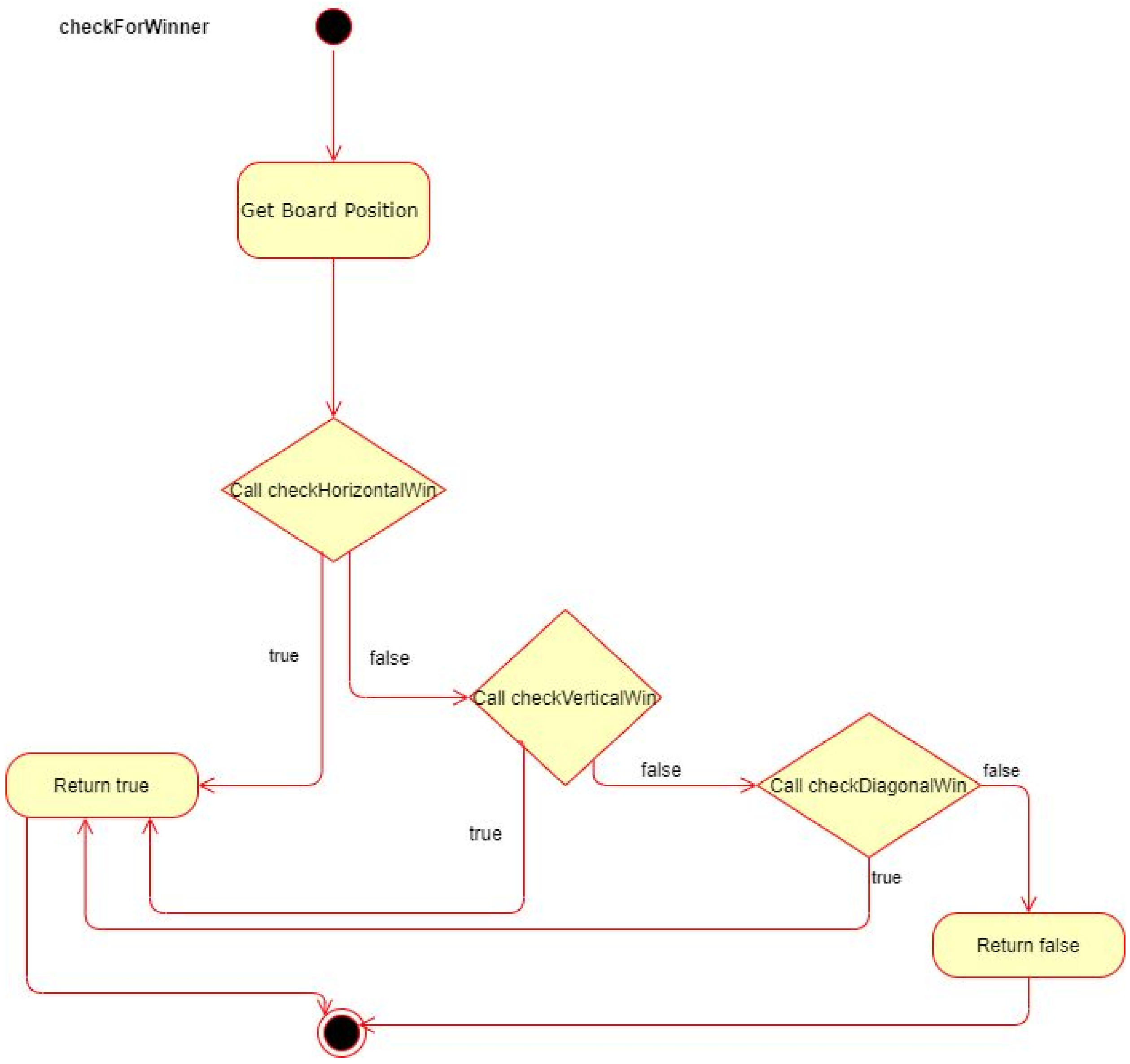

checkDiagonalWin
Get Board Position and player character
(In for loop going through all indexes from [2,2] to [4,4] (center 9 spaces)) Call isPlayerAtPos
false
true
for(int i = 0, i<5; i++)
for(int j = 0, j<5; j++)
Call isPlayerAtPos for row+i and column+j
(Top left to bottom right diagonal)
true
false
for(int i = 0, i<5; i++)
for(int j = 0, j<5; j++)
Call isPlayerAtPos for row-i and column-j
(Bottom left to top right diagonal)
false
Return false
Return true

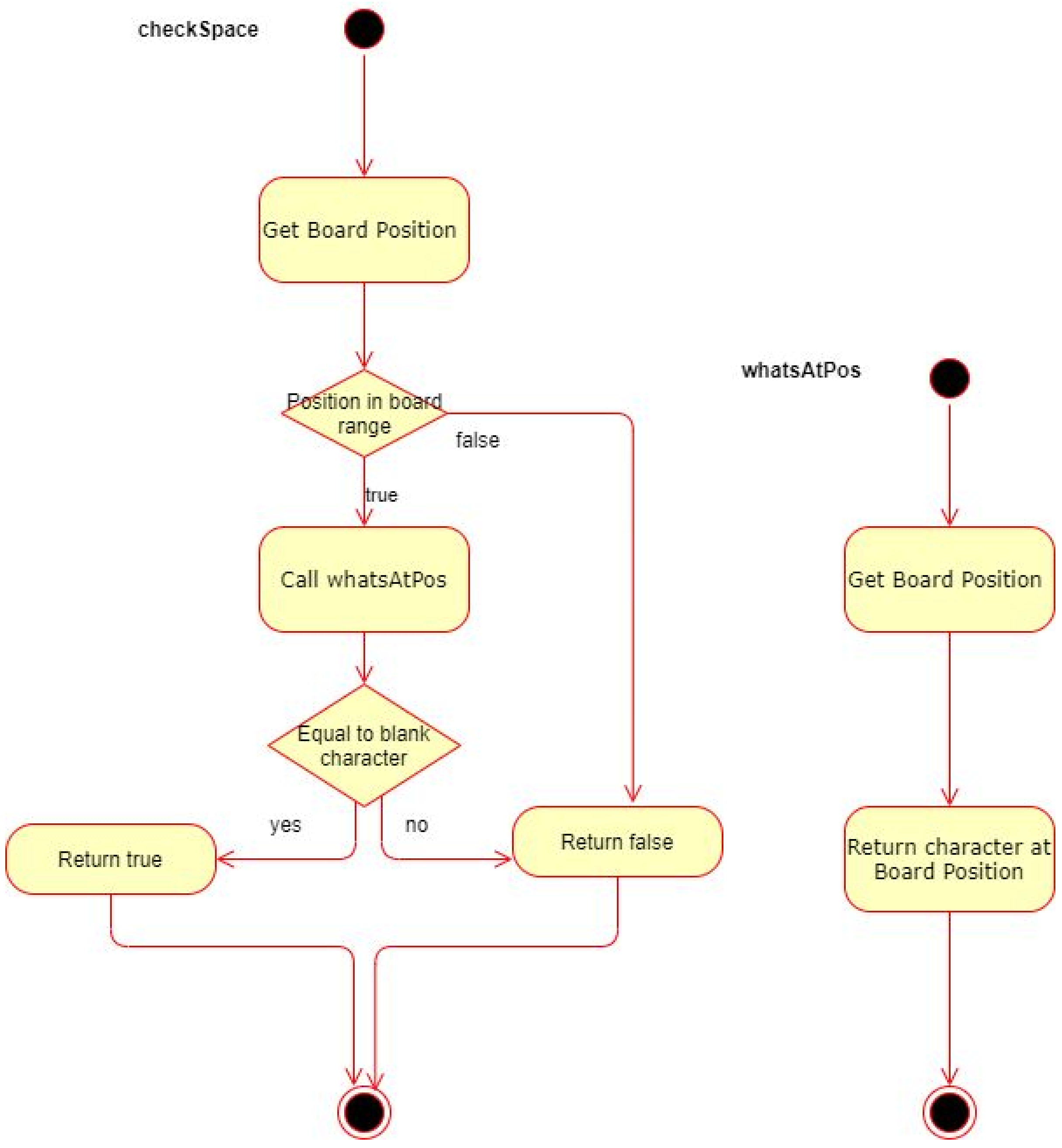
checkForDraw


Call whatsAtPos at every index with for loops
Blank character detected at any index
true
false
Return false
Return true


isPlayerAtPos
Get Board Position and player character
Call whatsAtPos
Equal to player character
no
yes
Return false
Return true


placeMarker
Get Board Position and player character
Assign player character to position


checkForWinner
Get Board Position
Call checkHorizontalWin
true
false
Call checkVerticalWin
false
true
Call checkDiagonalWin
false
true
Return true
Return false


checkSpace
Get Board Position
Position in board range
false
true
Call whatsAtPos
Equal to blank character
yes
no
Return true
Return false
whatsAtPos
Get Board Position
Return character at Board Position