Kellen Haas

CPSC 2150

Project 4

11/18/2020

Requirements Analysis

Functional Requirements

1. As a user, I can choose which row I want to place my marker in.
2. As a user, I can choose which column I want to place my marker in.
3. As a user, I can view the game board before my turn and after my turn with my updated marker that I just placed.
4. As a user, after the first user takes their turn, I can then choose my row.
5. As a user, after the first user takes their turn, I can choose my column.
6. As a user, I can expect that the system will notify me and my opponent if someone has won horizontally.
7. As a user, I can expect that the system will notify me and my opponent if someone has won vertically.
8. As a user, I can expect that the system will notify me and my opponent if someone has won diagonally.
9. As a user, I want to be notified by the system if there is a draw.
10. As a user, if I choose a position where a marker already has been placed, the system will tell me that I cannot place a marker there.

11. As a user, if I choose a position that is out of the bounds of the board, the system will tell me that it is not a valid position.
12. As a user, I want to be able to view both mine and my opponents placed markers after every turn.
13. As a user, I expect the top of the board to be the index 0, 0.
14. As a user, I want to be asked after the game has ended if I want to play again.
15. As a user, if I choose to play again, then the program should start over from the beginning and clear the game board.

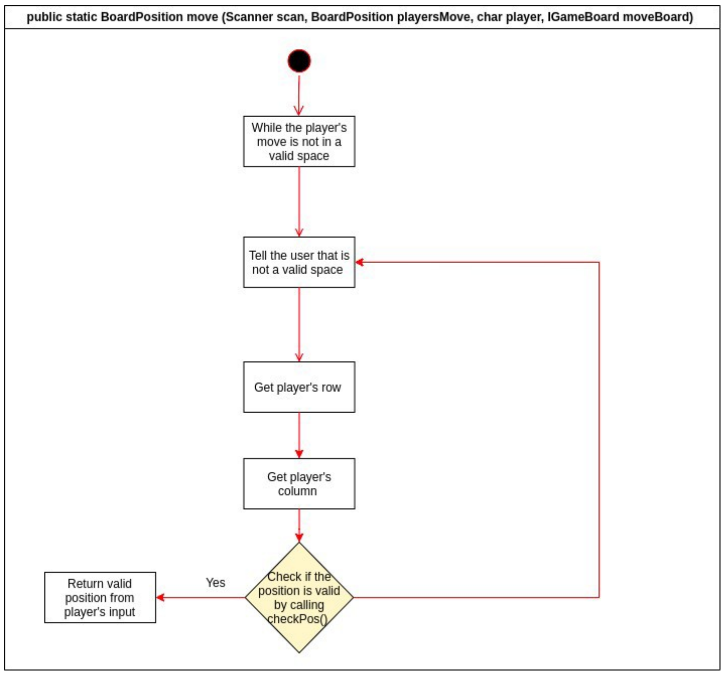
Non-Functional Requirements

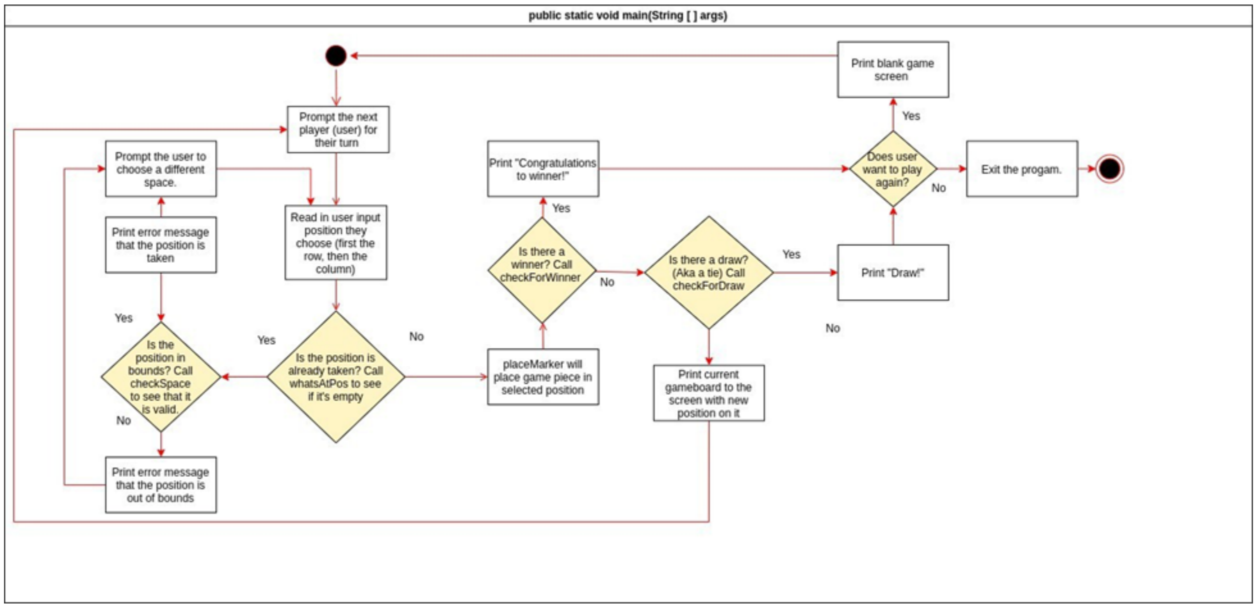
1. The system must be coded in Java programming language.
2. The system must be able to run on Unix/Linux.
3. Program must be able to compile and run quickly and efficiently.
4. The system must be written in IntelliJ IDE for debugging purposes in the future.
5. The top left corner of the game board must be 0, 0.
6. The game board is currently 8x8.
7. Currently, player X goes first and then player O.

Design

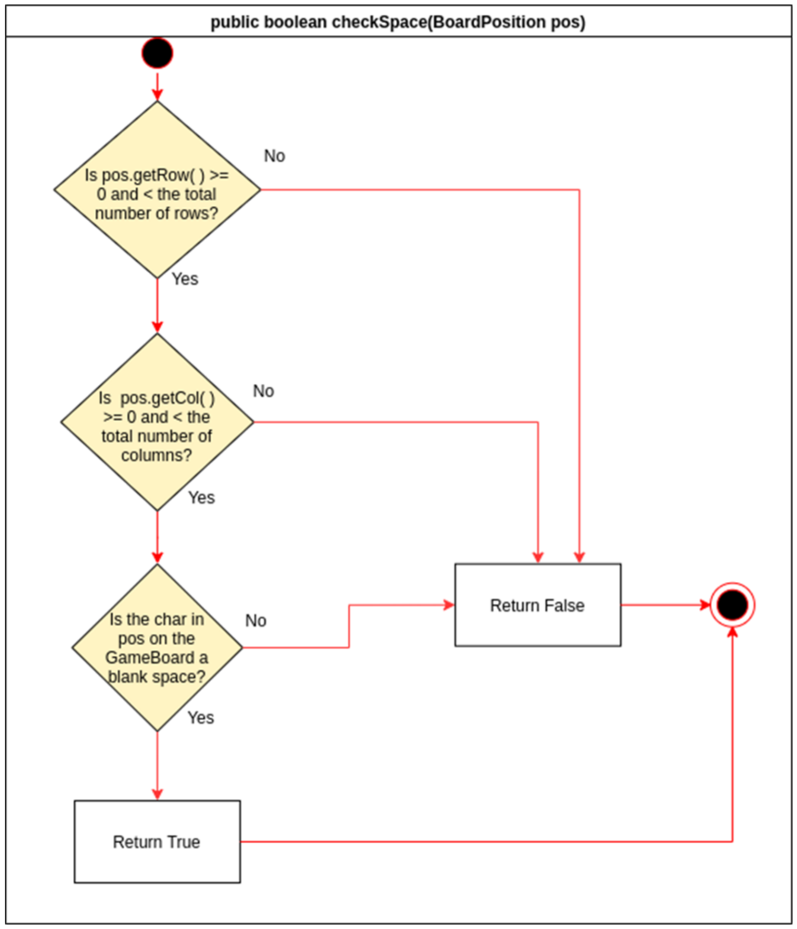
Activity Diagrams

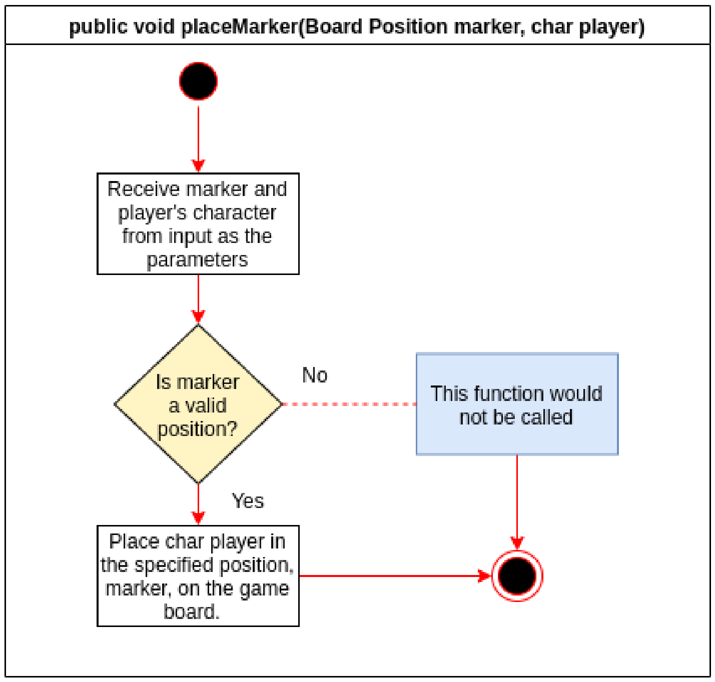
GameScreen.java

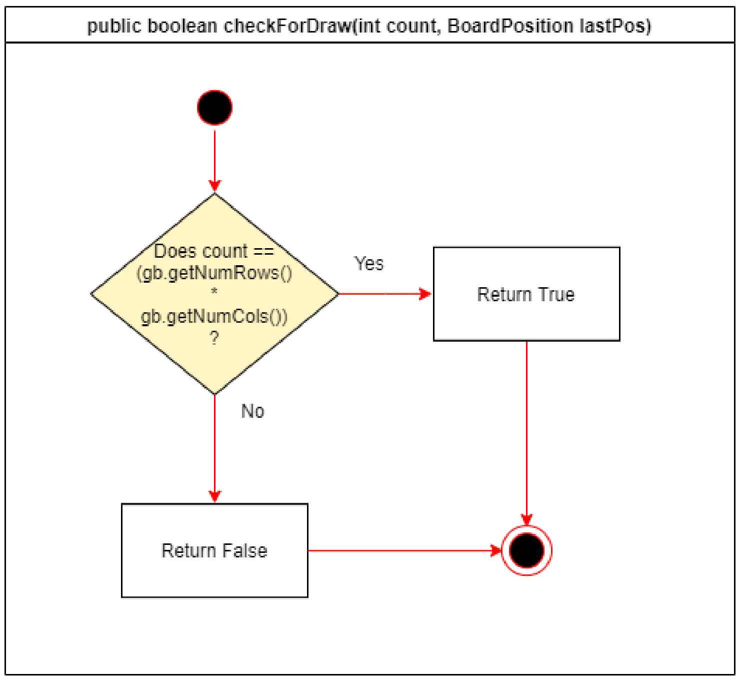

public static BoardPosition move (Scanner scan, BoardPosition playersMove, char player, IGameBoard moveBoard)
While the player's move is not in a valid space
Tell the user that is not a valid space
Get player's row
Get player's column
Check if the position is valid by calling checkPos()
Yes
Return valid position from player's input

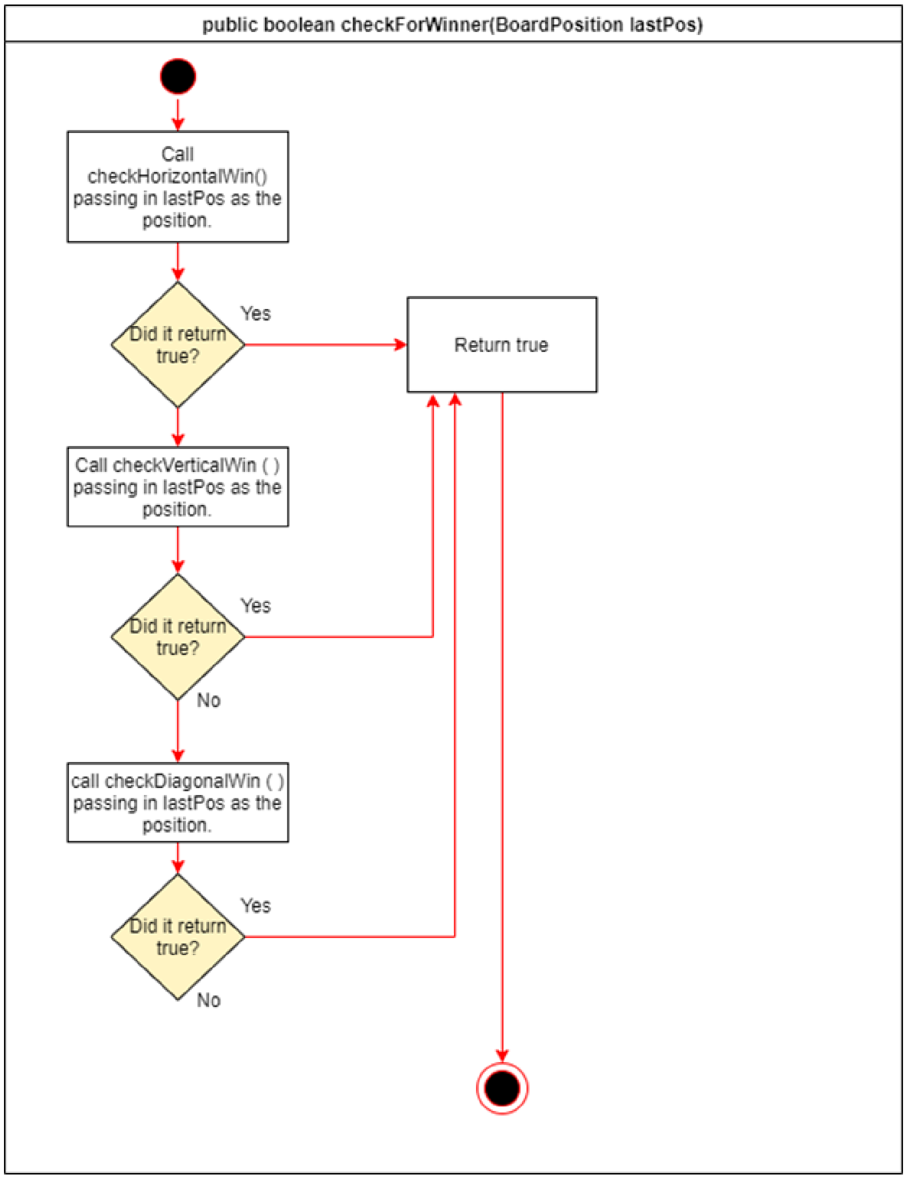

public static void main(String [ ] args)
Print blank game screen
Prompt the next player (user) for their turn
Prompt the user to choose a different space.
Print "Congratulations to winner!"
Does user want to play again?
Yes
No
Exit the progam.
Read in user input position they choose (first the row, then the column)
Print error message that the position is taken
Yes
Is there a winner? Call checkForWinner
No
Is there a draw? (Aka a tie) Call checkForDraw
Yes
Print "Draw!"
No
Yes
Is the position in bounds? Call checkSpace to see that it is valid.
No
Yes
Is the position is already taken? Call whatsAtPos to see if it's empty
No
placeMarker will place game piece in selected position
Print current gameboard to the screen with new position on it
Print error message that the position is out of bounds

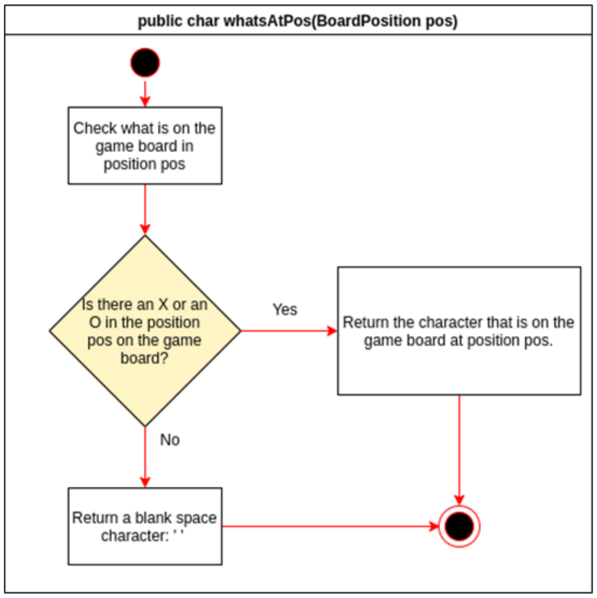
GameBoard.java

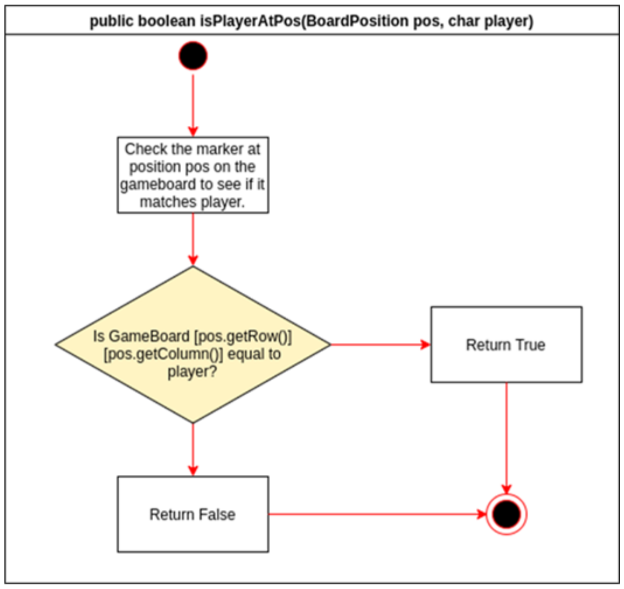

public boolean checkSpace(BoardPosition pos)
Is pos.getRow( ) >= 0 and < the total number of rows?
No
Yes
Is pos.getCol( ) >= 0 and < the total number of columns?
No
Yes
Is the char in pos on the GameBoard a blank space?
No
Yes
Return False
Return True

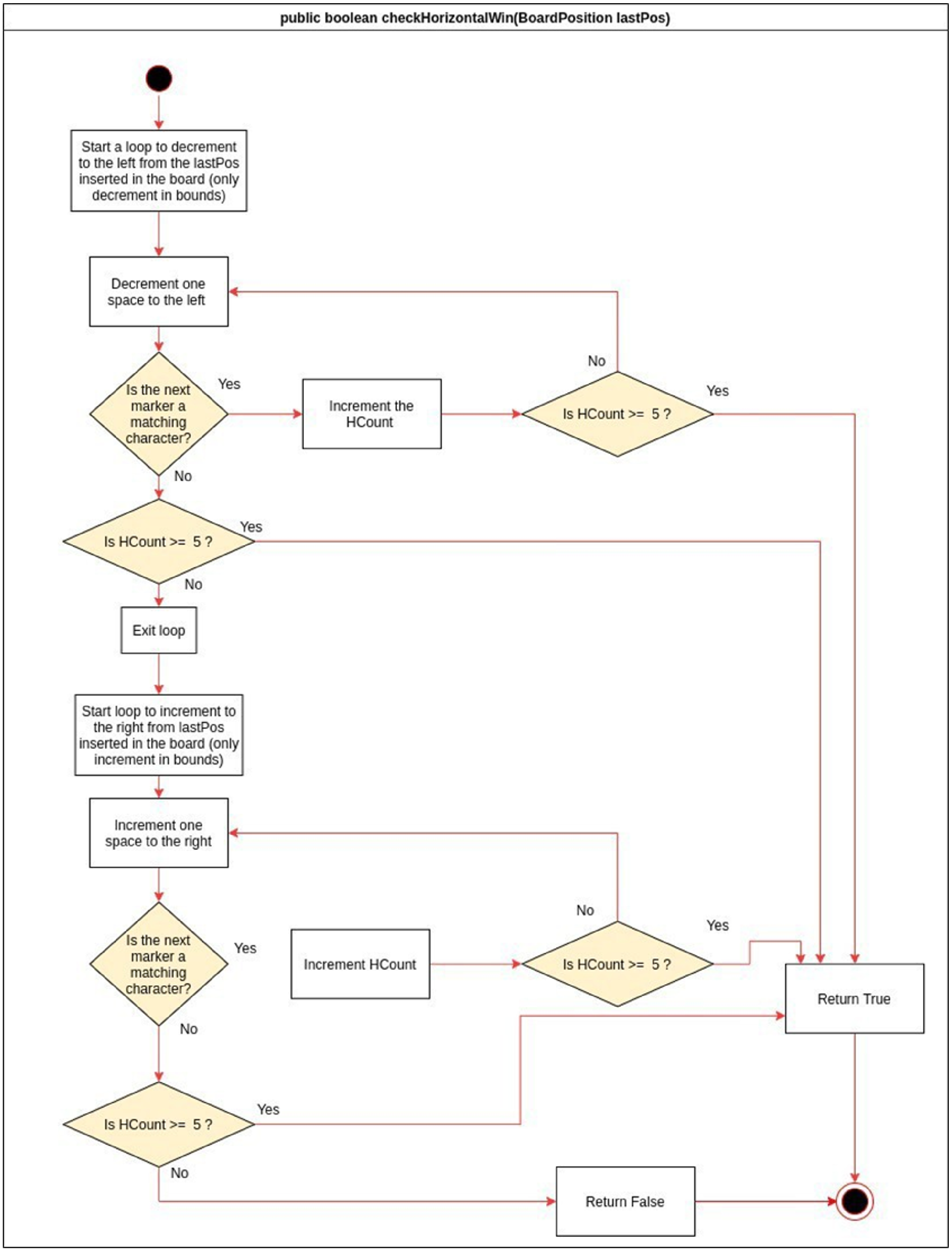

public void placeMarker(Board Position marker, char player)
Receive marker and player's character from input as the parameters
Is marker a valid position?
No
This function would not be called
Yes
Place char player in the specified position, marker, on the game board.

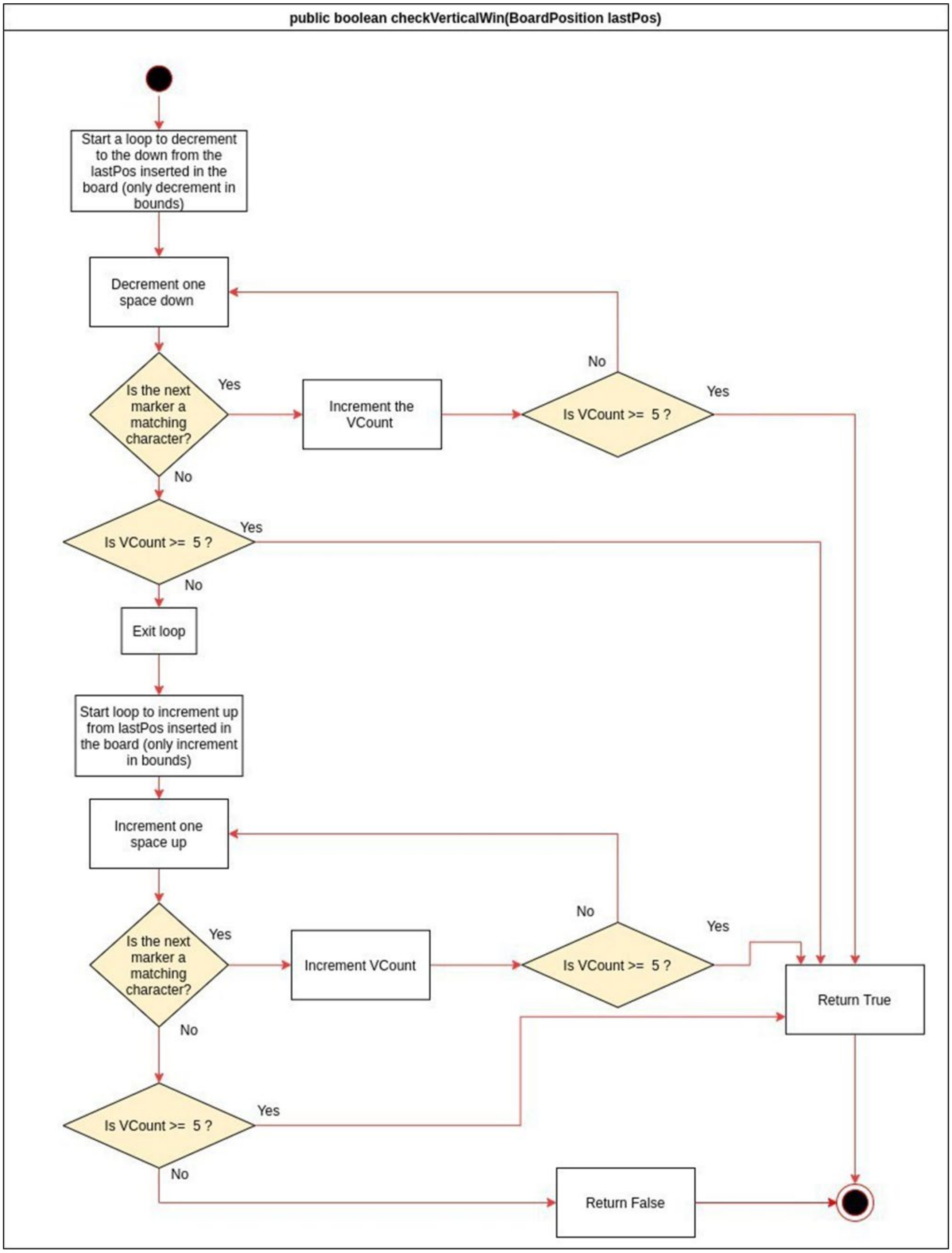

public boolean checkForDraw(int count, BoardPosition lastPos)
Does count == (gb.getNumRows() * gb.getNumCols()) ?
Yes
Return True
No
Return False

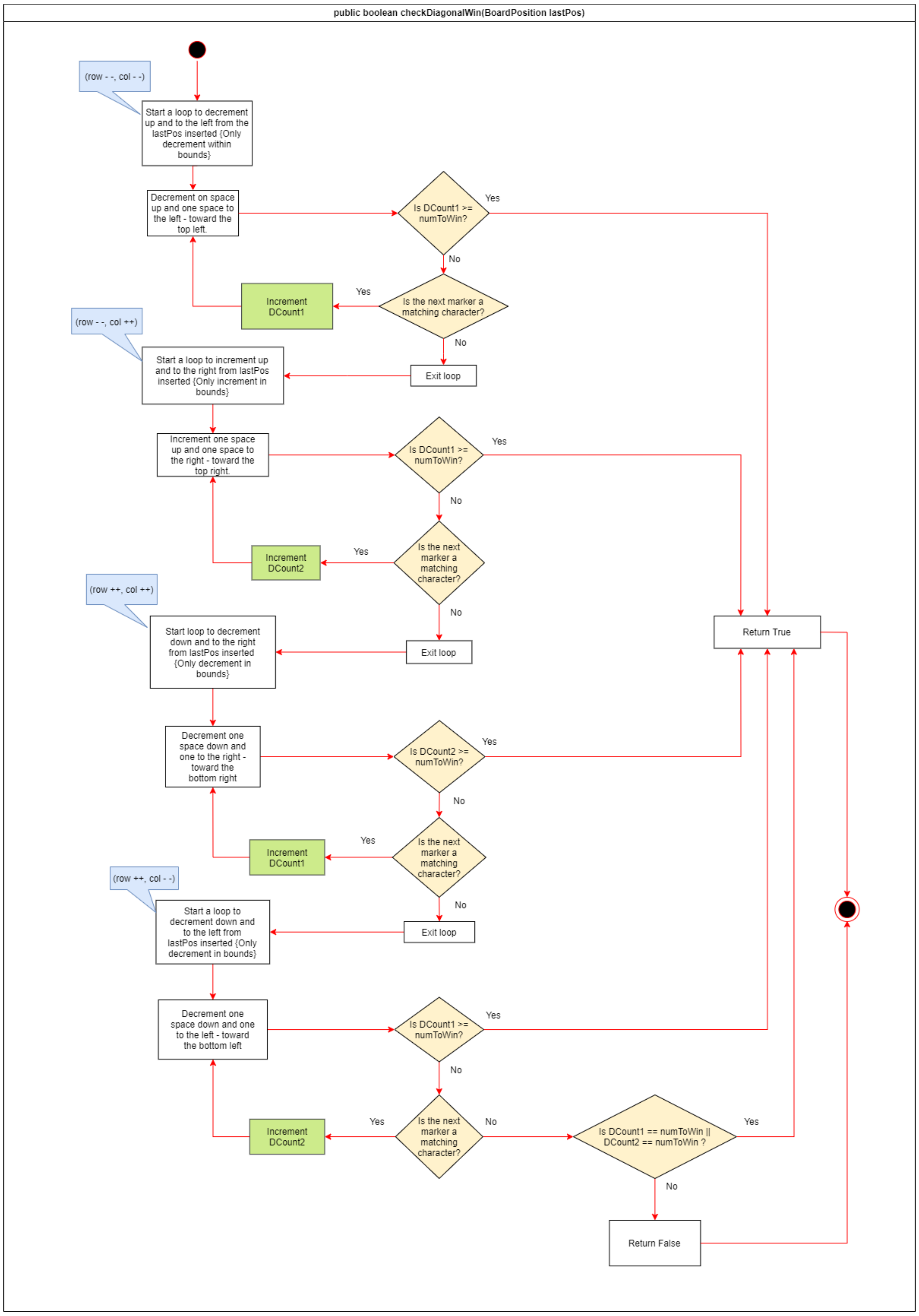

public boolean checkForWinner(BoardPosition lastPos)
Call checkHorizontalWin() passing in lastPos as the position.
Did it return true?
Yes
Return true
Call checkVerticalWin ( ) passing in lastPos as the position.
Did it return true?
Yes
No
call checkDiagonalWin ( ) passing in lastPos as the position.
Did it return true?
Yes
No

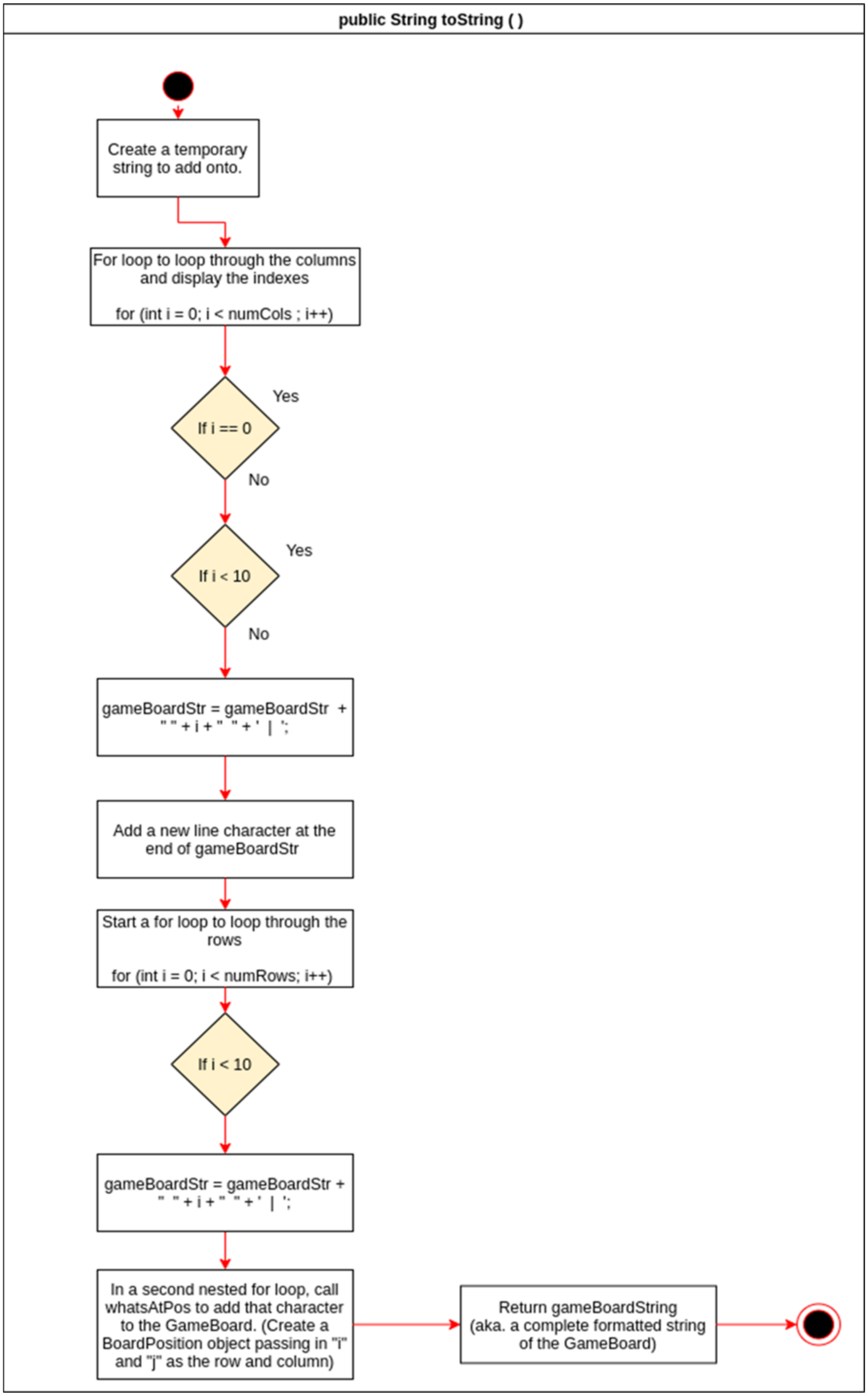

public char whatsAtPos(BoardPosition pos)
Check what is on the game board in position pos
Is there an X or an O in the position pos on the game board?
Yes
Return the character that is on the game board at position pos.
No
Return a blank space character: ' '


public boolean isPlayerAtPos(BoardPosition pos, char player)
Check the marker at position pos on the gameboard to see if it matches player.
Is GameBoard [pos.getRow()] [pos.getColumn()] equal to player?
Return True
Return False

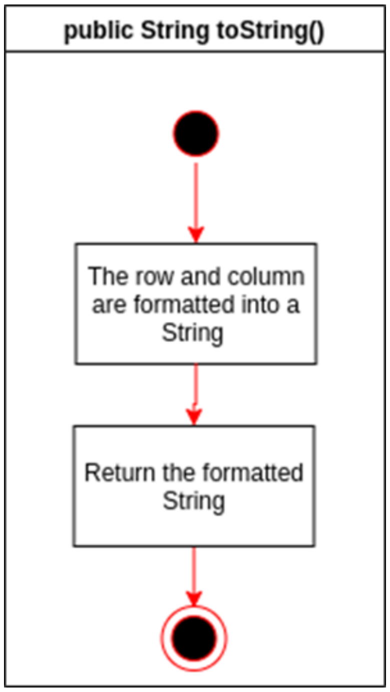

public boolean checkHorizontalWin(BoardPosition lastPos)
Start a loop to decrement to the left from the lastPos inserted in the board (only decrement in bounds)
Decrement one space to the left
Is the next marker a matching character?
Yes
Increment the HCount
Is HCount >= 5 ?
No
Yes
No
Is HCount >= 5 ?
Yes
No
Exit loop
Start loop to increment to the right from lastPos inserted in the board (only increment in bounds)
Increment one space to the right
Is the next marker a matching character?
Yes
Increment HCount
Is HCount >= 5 ?
No
Yes
Return True
No
Is HCount >= 5 ?
Yes
No
Return False

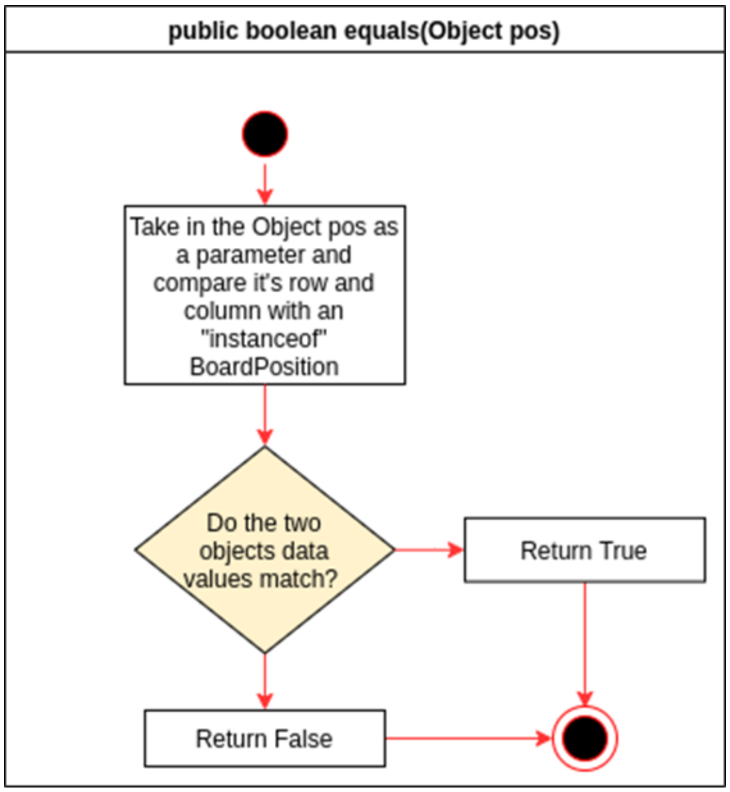

public boolean checkVerticalWin(BoardPosition lastPos)
Start a loop to decrement to the down from the lastPos inserted in the board (only decrement in bounds)
Decrement one space down
Is the next marker a matching character?
Yes
Increment the VCount
Is VCount >= 5 ?
No
Yes
No
Is VCount >= 5 ?
Yes
No
Exit loop
Start loop to increment up from lastPos inserted in the board (only increment in bounds)
Increment one space up
Is the next marker a matching character?
Yes
Increment VCount
Is VCount >= 5 ?
No
Yes
Return True
No
Is VCount >= 5 ?
Yes
No
Return False

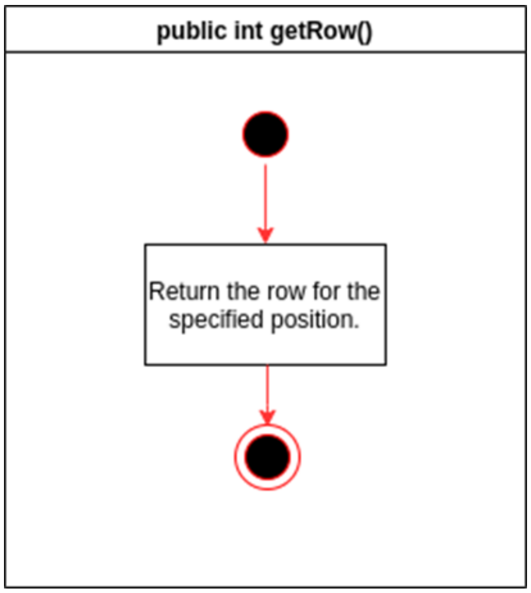

public boolean checkDiagonalWin(BoardPosition lastPos)
(row - -, col - -)
Start a loop to decrement up and to the left from the lastPos inserted {Only decrement within bounds}
Decrement on space up and one space to the left - toward the top left.
Is DCount1 >= numToWin?
Yes
No
Is the next marker a matching character?
Yes
Increment DCount1
No
Exit loop
(row - -, col ++)
Start a loop to increment up and to the right from lastPos inserted {Only increment in bounds}
Increment one space up and one space to the right - toward the top right.
Is DCount1 >= numToWin?
Yes
No
Is the next marker a matching character?
Yes
Increment DCount2
No
Exit loop
(row ++, col ++)
Start loop to decrement down and to the right from lastPos inserted {Only decrement in bounds}
Decrement one space down and one to the right - toward the bottom right
Is DCount2 >= numToWin?
Yes
No
Is the next marker a matching character?
Yes
Increment DCount1
No
Exit loop
(row ++, col - -)
Start a loop to decrement down and to the left from lastPos inserted {Only decrement in bounds}
Decrement one space down and one to the left - toward the bottom left
Is DCount1 >= numToWin?
Yes
No
Is the next marker a matching character?
Yes
Increment DCount2
No
Is DCount1 == numToWin || DCount2 == numToWin ?
Yes
No
Return True
Return False

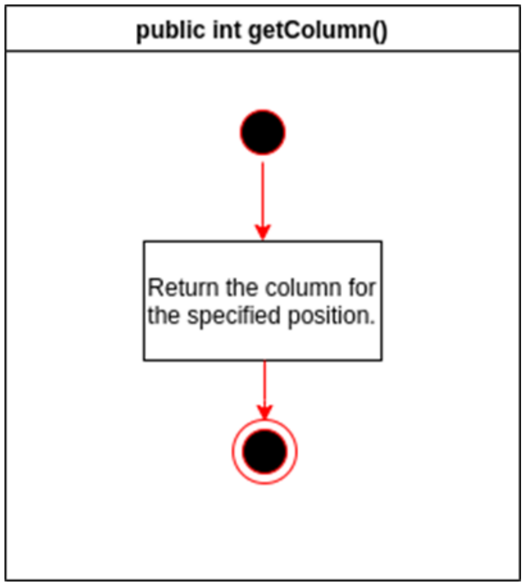

public String toString ( )
Create a temporary string to add onto.
For loop to loop through the columns and display the indexes
for (int i = 0; i < numCols ; i++)
If i == 0
Yes
No
If i < 10
Yes
No
gameBoardStr = gameBoardStr + " " + i + " " + ' | ';
Add a new line character at the end of gameBoardStr
Start a for loop to loop through the rows
for (int i = 0; i < numRows; i++)
If i < 10
gameBoardStr = gameBoardStr + " " + i + " " + ' | ';
In a second nested for loop, call whatsAtPos to add that character to the GameBoard. (Create a BoardPosition object passing in "i" and "j" as the row and column)
Return gameBoardString (aka. a complete formatted string of the GameBoard)

BoardPosition.java

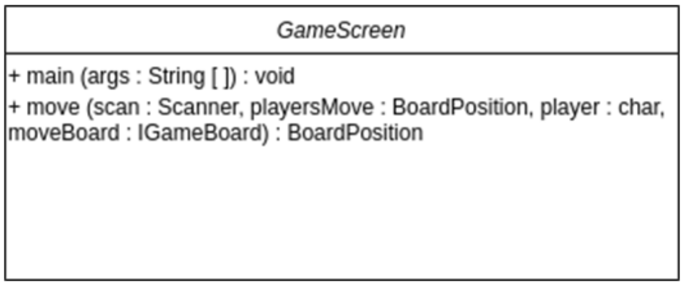

public String toString()
The row and column are formatted into a String
Return the formatted String


public boolean equals(Object pos)
Take in the Object pos as a parameter and compare it's row and column with an "instanceof" BoardPosition
Do the two objects data values match?
Return True
Return False



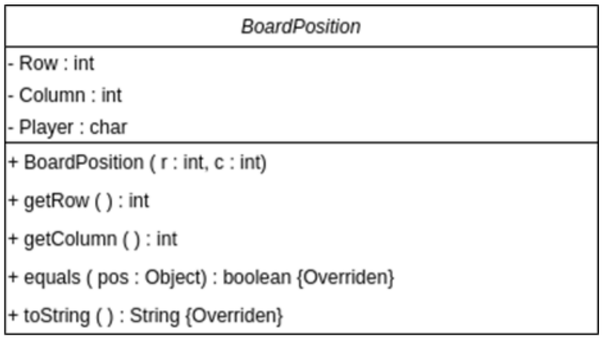


UML Class Diagrams



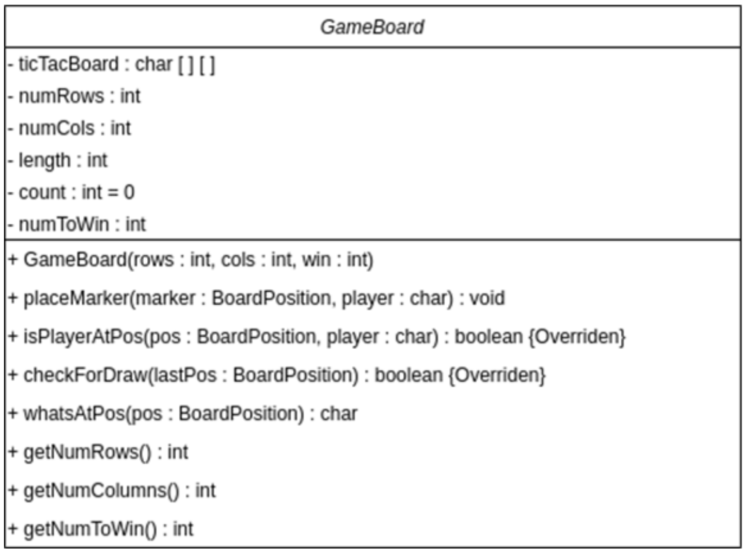
GameScreen

```
+ main (args : String [ ]) : void
+ move (scan : Scanner, playersMove : BoardPosition, player : char,
moveBoard : IGameBoard) : BoardPosition
```



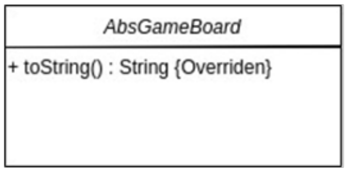
BoardPosition

```
- Row : int
- Column : int
- Player : char
---
+ BoardPosition ( r : int, c : int)
+ getRow ( ) : int
+ getColumn ( ) : int
+ equals ( pos : Object) : boolean {Overriden}
+ toString ( ) : String {Overriden}
```



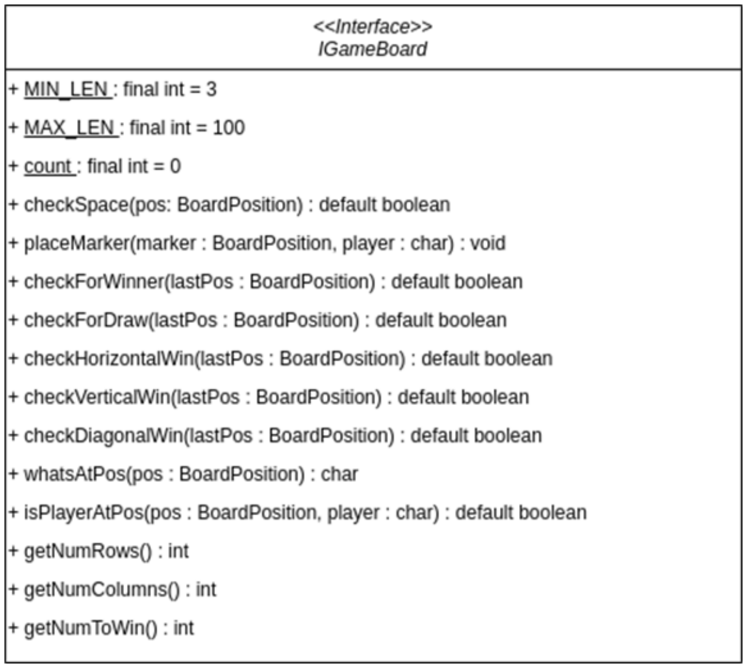
GameBoard

```
- ticTacBoard : char [ ] [ ]
- numRows : int
- numCols : int
- length : int
- count : int = 0
- numToWin : int
---
+ GameBoard(rows : int, cols : int, win : int)
+ placeMarker(marker : BoardPosition, player : char) : void
+ isPlayerAtPos(pos : BoardPosition, player : char) : boolean {Overriden}
+ checkForDraw(lastPos : BoardPosition) : boolean {Overriden}
+ whatsAtPos(pos : BoardPosition) : char
+ getNumRows() : int
+ getNumColumns() : int
+ getNumToWin() : int
```



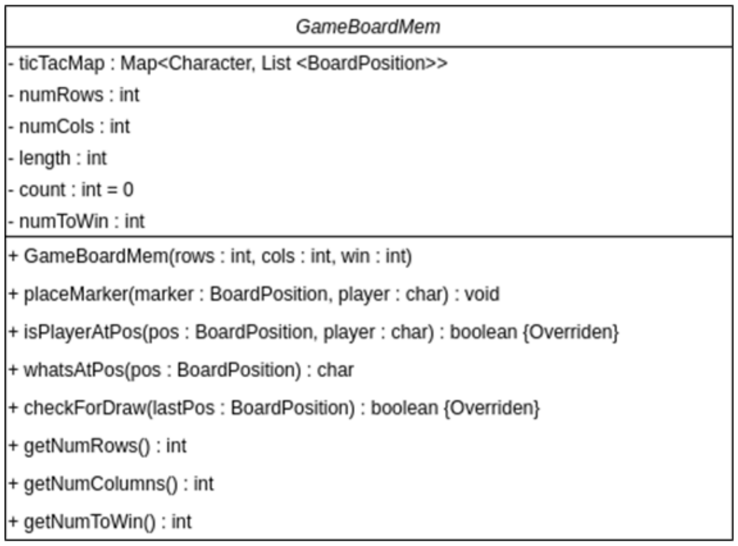
AbsGameBoard

```
+ toString() : String {Overriden}
```



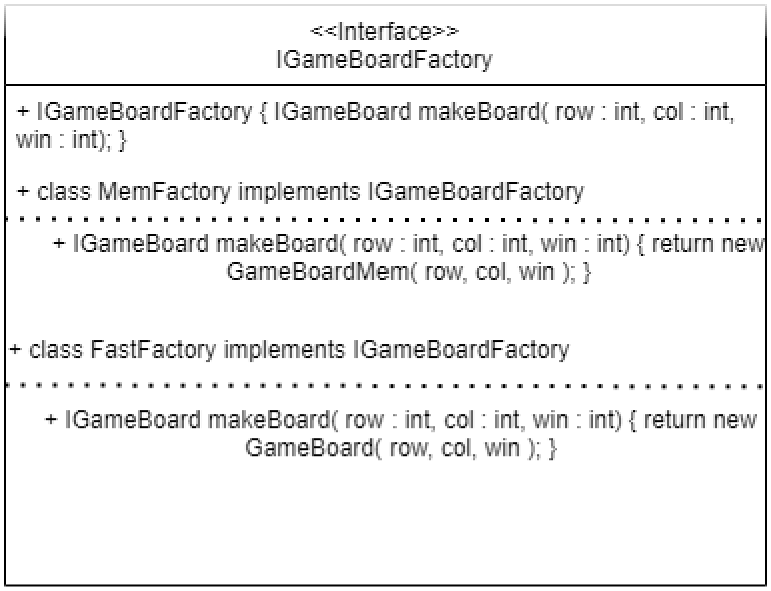
<<Interface>>
IGameBoard

```
+ MIN_LEN : final int = 3
+ MAX_LEN : final int = 100
+ count : final int = 0
+ checkSpace(pos: BoardPosition) : default boolean
+ placeMarker(marker : BoardPosition, player : char) : void
+ checkForWinner(lastPos : BoardPosition) : default boolean
+ checkForDraw(lastPos : BoardPosition) : default boolean
+ checkHorizontalWin(lastPos : BoardPosition) : default boolean
+ checkVerticalWin(lastPos : BoardPosition) : default boolean
+ checkDiagonalWin(lastPos : BoardPosition) : default boolean
+ whatsAtPos(pos : BoardPosition) : char
+ isPlayerAtPos(pos : BoardPosition, player : char) : default boolean
+ getNumRows() : int
+ getNumColumns() : int
+ getNumToWin() : int
```



GameBoardMem

```
- ticTacMap : Map<Character, List <BoardPosition>>
- numRows : int
- numCols : int
- length : int
- count : int = 0
- numToWin : int
---
+ GameBoardMem(rows : int, cols : int, win : int)
+ placeMarker(marker : BoardPosition, player : char) : void
+ isPlayerAtPos(pos : BoardPosition, player : char) : boolean {Overriden}
+ whatsAtPos(pos : BoardPosition) : char
+ checkForDraw(lastPos : BoardPosition) : boolean {Overriden}
+ getNumRows() : int
+ getNumColumns() : int
+ getNumToWin() : int
```


<<Interface>>
IGameBoardFactory

```
+ IGameBoardFactory { IGameBoard makeBoard( row : int, col : int,
win : int); }

+ class MemFactory implements IGameBoardFactory
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
    + IGameBoard makeBoard( row : int, col : int, win : int) { return new
                GameBoardMem( row, col, win ); }

+ class FastFactory implements IGameBoardFactory
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
    + IGameBoard makeBoard( row : int, col : int, win : int) { return new
                GameBoard( row, col, win ); }
```

Testing

`public void GameBoard()`

Input:	Output:	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | | | | IGameBoard gb = GameBoardFactory(3, 3, 3)	GameBoard constructor getNumRows() = 3 getNumCols() = 3 getNumToWin() = 3 State: unchanged	This test case is unique and distinct because it tests that the constructor successfully creates a game board using the minimum values for row, column, and number in a row to win. **Function Name:** `TestGameBoard_Minimums`

`public void GameBoard()`

Input:	Output:	Reason:
State: (number to win = 25) | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | | | | IGameBoard gb = GameBoardFactory(100, 100, 25)	GameBoard constructor getNumRows() = 100 getNumCols() = 100 getNumToWin() = 25 State: unchanged	This test case is unique and distinct because it tests that the constructor successfully creates a game board using the maximum values for row, column, and number in a row to win. **Function Name:** `TestGameBoard_Maximums`

```
public void GameBoard()
```

Input:

State: (number to win = 3)

	0	1	2	3	4	5
0						
1						
2						
3						

IGameBoard gb =
GameBoardFactory(4, 6, 3)

Output:

GameBoard constructor
getNumRows() = 4
getNumCols() = 6
getNumToWin() = 3

State: unchanged

Reason:
This test case is unique and distinct because it tests that the constructor successfully creates a game board using different values for row and column.

Function Name:
`TestGameBoard_Diff_RowsAndCols`

```
public void placeMarker(BoardPosition testPos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb =
GameBoardFactory(3, 3, 3)
player = ‘X’

testPos = new BoardPosition (2, 2)
gb.placeMarker (testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = ‘X’

State:

	0	1	2
0			
1			
2			X

Reason:
This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in the bottom right corner on the game board (2, 2).

Function Name:
`Test_PlaceMarker_in_Bottom_Right_Corner()`

```
public void placeMarker(BoardPosition testPos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb =
GameBoardFactory(3, 3, 3)
player = ‘X’

testPos = new BoardPosition (0, 2)
gb.placeMarker (testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = ‘X’

State:

	0	1	2
0			X
1			
2			

Reason:
This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in top right corner on the game board (0, 2).

Function Name:
`Test_PlaceMarker_in_Top_Right_Corner()`

```
public void placeMarker(BoardPosition testPos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb =
GameBoardFactory(3, 3, 3)
player = ‘X’

testPos = new BoardPosition (2, 0)
gb.placeMarker (testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = ‘X’

State:

	0	1	2
0			
1			
2	X		

Reason:
This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in the bottom left corner on the game board (2, 0).

Function Name:
`Test_PlaceMarker_in_Bottom_Left_Corner()`

```
public void placeMarker(BoardPosition testPos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb =
GameBoardFactory(3, 3, 3)
player = ‘X’

testPos = new BoardPosition (0, 0)
gb.placeMarker (testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = ‘X’

State:

	0	1	2
0	X		
1			
2			

Reason:
This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in top left corner on the game board (0, 0).

Function Name:
`Test_PlaceMarker_in_Top_Left_Corner()`

```
public void placeMarker(BoardPosition testPos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb =
GameBoardFactory(3, 3, 3)
player = ‘X’

testPos = new BoardPosition (1, 1)
gb.placeMarker (testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
use whatsAtPos(testPos) for comparing the results of a cell in the board after a marker is placed.

State:

	0	1	2
0	X	O	X
1	O	X	O
2	X	O	X

Reason:
This test case is unique and distinct because it tests that the placeMarker function can successfully place a player’s character in the last available position on the game board (1, 1).

Function Name:
`Test_PlaceMarker_Last_Marker_on_Board()`

```
public char whatsAtPos(BoardPosition testPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
player = 'X'

testPos = new BoardPosition (2, 2)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = ' '

State:

	0	1	2
0			
1			
2			🟨

Reason:
This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the specified position.

Function Name:
`Test_WhatsAtPos_Blank_Board()`

```
public char whatsAtPos(BoardPosition testPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
player = 'X'

testPos = new BoardPosition (0, 0)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = 'X'

State:

	0	1	2
0	X		
1			
2			

Reason:
This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the specified position even if it is the only marker on the board.

Function Name:
`Test_WhatsAtPos_One_Marker_Placed()`

```
public char whatsAtPos(BoardPosition testPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
player = ‘X’

testPos = new BoardPosition (2, 2)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = ‘X’

State:

	0	1	2
0			
1			
2			X

Reason:
This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the bottom right corner of the game board even if it is the only marker on the board.

Function Name:
`Test_WhatsAtPos_Bottom_Right_Corner()`

```
public char whatsAtPos(BoardPosition testPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
player = ‘X’

testPos = new BoardPosition (0, 2)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = ‘X’

State:

	0	1	2
0			X
1			
2			

Reason:
This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the top right corner of the game board even if it is the only marker on the board.

Function Name:
`Test_WhatsAtPos_Top_Right_Corner()`

```
public char whatsAtPos(BoardPosition testPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
player = 'X'

testPos = new BoardPosition (2, 0)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = 'X'

State:

	0	1	2
0			
1			
2	X		

Reason:
This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the bottom left corner of the game board even if it is the only marker on the board.

Function Name:
`Test_WhatsAtPos_Bottom_Left_Corner()`

```
public char whatsAtPos(BoardPosition testPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
player = 'X'

testPos = new BoardPosition (0, 0)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
whatsAtPos(testPos) = 'X'

State:

	0	1	2
0	X		
1			
2			

Reason:
This test case is unique and distinct because it tests that the whatsAtPos function can successfully determine and then return the character that is in the top left corner of the game board even if it is the only marker on the board.

Function Name:
`Test_WhatsAtPos_Top_Left_Corner()`

`default boolean checkSpace(BoardPosition pos)`

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)

testPos = new BoardPosition (0, 0)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkSpace(pos) = true

State is unchanged

Reason:
This test case is unique and distinct because it tests that the checkSpace function can successfully determine whether the position the user wants to place their marker is empty.

Function Name:
`Test_CheckSpace_Empty_Space()`

`default boolean checkSpace(BoardPosition pos)`

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)

testPos = new BoardPosition (2, 2)

gb.placeMarker(testPos)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkSpace(testPos) = false

State:

	0	1	2
0			
1			
2			X

Reason:
This test case is unique and distinct because it tests that the checkSpace function can successfully determine whether the position the user wants to place their marker is already taken (aka not empty).

Function Name:
`Test_CheckSpace_Unavailable_Space()`

```
default boolean checkSpace(BoardPosition pos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)

testPos = new BoardPosition (3, 3)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkSpace(pos) = false

State is unchanged

Reason:
This test case is unique and distinct because it tests that the checkSpace function can successfully determine whether the position is out of the bounds of the array.

Function Name:
`Test_CheckSpace_Out_of_Bounds()`

```
public boolean isPlayerAtPos(BoardPosition pos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)

testPos = new BoardPosition (1, 1)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.isPlayerAtPos(testPos, player) = false

State is unchanged

Reason:
This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player's char passed into the function matches the char in position pos.

Function Name:
`Test_IsPlayerAtPos_Empty_Board()`

```
public boolean isPlayerAtPos(BoardPosition pos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
//Assume player = ‘X’

testPos = new BoardPosition (1, 1)

gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.isPlayerAtPos(testPos, player) = true

State:

	0	1	2
0			
1		X	
2			

Reason:
This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player’s char passed into the function matches the char in position pos. In this case, they should match.

Function Name:
`Test_IsPlayerAtPos_with_Player_in_Pos()`

```
public boolean isPlayerAtPos(BoardPosition pos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
//Assume player = ‘X’

gb.placeMarker((0, 0), ‘A’)
gb.placeMarker((0, 1), ‘B’)
testPos = new BoardPosition (0, 2)
gb.placeMarker(testPos, ‘C’)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.isPlayerAtPos(testPos, player) = false

State:

	0	1	2
0	A		
1	B		
2	C		

Reason:
This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player’s char passed into the function matches the char in position pos. In this case, they should not match.

Function Name:
`Test_IsPlayerAtPos_with_Wrong_Player_in_Pos()`

```
public boolean isPlayerAtPos(BoardPosition pos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb =
GameBoardFactory(3, 3, 3)
//Assume player = ‘X’

gb.placeMarker((0, 0), ‘X’)
gb.placeMarker((0, 1), ‘X’)
testPos = new BoardPosition (0, 2)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.isPlayerAtPos(testPos, player) = true

State:

	0	1	2
0	X		
1	X		
2	X		

Reason:
This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player’s char passed into the function matches the char in position pos, even in the case where the entire row is marked by the same char.

Function Name:
`Test_IsPlayerAtPos_True_Row()`

```
public boolean isPlayerAtPos(BoardPosition pos, char player)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb =
GameBoardFactory(3, 3, 3)
//Assume player = ‘X’

testPos = new BoardPosition (0, 2)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.isPlayerAtPos(testPos, player) = true

State:

	0	1	2
0			
1		X	
2			

Reason:
This test case is unique and distinct because it tests that the isPlayerAtPos function can successfully determine if the player’s char passed into the function matches the char in position pos, even in the case where the char to match is in the middle of the board.

Function Name:
`Test_IsPlayerAtPos_Middle()`

```
public boolean checkForDraw(BoardPosition lastPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
//Assume player = ‘O’

gb.placeMarker((0, 0), ‘X’)
gb.placeMarker((0, 1), ‘O’)
gb.placeMarker((0, 2), ‘X’)
gb.placeMarker((2, 0), ‘O’)
gb.placeMarker((2, 1), ‘X’)
testPos = new BoardPosition (2, 2)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkForDraw(testPos) = false

State:

	0	1	2
0	X	O	X
1			
2	O	X	O

Reason:
This test case is unique and distinct because it tests that the checkForDraw function can successfully determine if the board is full. In this case, checkForDraw should be false because there is an empty row.

Function Name:
`Test_CheckForDraw_One_Empty_Row()`

```
public boolean checkForDraw(BoardPosition lastPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
//Assume player = ‘O’

gb.placeMarker((0, 0), ‘X’)
gb.placeMarker((1, 0), ‘O’)
gb.placeMarker((2, 0), ‘X’)
gb.placeMarker((0, 1), ‘O’)
gb.placeMarker((1, 1), ‘X’)
testPos = new BoardPosition (2, 1)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkForDraw(testPos) = false

State:

	0	1	2
0	X	O	
1	O	X	
2	X	O	

Reason:
This test case is unique and distinct because it tests that the checkForDraw function can successfully determine if the board is full. In this case, checkForDraw should be false because there is an empty column.

Function Name:
`Test_CheckForDraw_One_Empty_Col()`

```
public boolean checkForDraw(BoardPosition lastPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb =
GameBoardFactory(3, 3, 3)
//Assume player = ‘O’
gb.placeMarker((0, 0), ‘X’)
gb.placeMarker((0, 1), ‘O’)
gb.placeMarker((0, 2), ‘X’)
gb.placeMarker((1, 0), ‘O’)
gb.placeMarker((1, 1), ‘X’)
gb.placeMarker((1, 2), ‘O’)
gb.placeMarker((2, 0), ‘X’)
testPos = new BoardPosition (2, 1)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkForDraw(testPos) = false

State:

	0	1	2
0	X	O	
1	O	X	
2	X	O	

Reason:
This test case is unique and distinct because it tests that the checkForDraw function can successfully determine if the board is full. In this case, checkForDraw should be false because there is an empty space left.

Function Name:
`Test_CheckForDraw_One_Empty_Space()`

```
public boolean checkForDraw(BoardPosition lastPos)
```

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb =
GameBoardFactory(3, 3, 3)
//Assume player = ‘O’

gb.placeMarker((0, 0), ‘X’)
gb.placeMarker((0, 1), ‘X’)
gb.placeMarker((0, 2), ‘O’)
gb.placeMarker((1, 0), ‘O’)
gb.placeMarker((1, 1), ‘O’)
gb.placeMarker((1, 2), ‘X’)
gb.placeMarker((2, 0), ‘X’)
gb.placeMarker((2, 1), ‘X’)
testPos = new BoardPosition (2, 2)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkForDraw(testPos) = true

State:

	0	1	2
0	X	X	O
1	O	O	X
2	X	X	O

Reason:
This test case is unique and distinct because it tests that the checkForDraw function can successfully determine if the board is full. In this case, checkForDraw should be true because every space on the board is full and there was no winner.

Function Name:
`Test_CheckForDraw_Full_Board()`

`default boolean checkDiagonalWin(BoardPosition lastPos)`

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
//Assume player = ‘X’
gb.placeMarker((0, 0), ‘X’)
gb.placeMarker((0, 1), ‘O’)
gb.placeMarker((0, 2), ‘X’)
gb.placeMarker((1, 0), ‘O’)
gb.placeMarker((1, 1), ‘X’)
gb.placeMarker((1, 2), ‘O’)
gb.placeMarker((2, 0), ‘X’)
gb.placeMarker((2, 1), ‘O’)
testPos = new BoardPosition (2, 2)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkDiagonalWin(testPos) = true

State:

	0	1	2
0	X	O	X
1	O	X	O
2	X	O	X

Reason:
This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from bottom right corner up to the top left corner.

Function Name:
`Test_Diagonal_Win_Start_at_Bottom_Right()`

`default boolean checkDiagonalWin(BoardPosition lastPos)`

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
//Assume player = ‘X’

gb.placeMarker((0, 0), ‘X’)
gb.placeMarker((0, 1), ‘O’)
gb.placeMarker((0, 2), ‘X’)
gb.placeMarker((1, 0), ‘O’)
gb.placeMarker((1, 1), ‘X’)
gb.placeMarker((1, 2), ‘O’)
gb.placeMarker((2, 0), ‘X’)
gb.placeMarker((2, 1), ‘O’)
testPos = new BoardPosition (2, 2)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkDiagonalWin(testPos) = true

State:

	0	1	2
0			X
1		X	
2	X		

Reason:
This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top right corner down to bottom left corner.

Function Name:
`Test_Diagonal_Win_Start_at_Top_Right()`

`default boolean checkDiagonalWin(BoardPosition lastPos)`

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
//Assume player = ‘X’
gb.placeMarker((0, 0), ‘X’)
gb.placeMarker((0, 1), ‘O’)
gb.placeMarker((0, 2), ‘X’)
gb.placeMarker((1, 0), ‘O’)
gb.placeMarker((1, 1), ‘X’)
gb.placeMarker((1, 2), ‘O’)
gb.placeMarker((2, 0), ‘X’)
gb.placeMarker((2, 1), ‘O’)
testPos = new BoardPosition (2, 2)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkDiagonalWin(testPos) = true

State:

	0	1	2
0			X
1		X	
2	X		

Reason:
This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from bottom left corner up to top right corner.

Function Name:
`Test_Diagonal_Win_Start_at_Bottom_Left()`

`default boolean checkDiagonalWin(BoardPosition lastPos)`

Input:

State: (number to win = 3)

	0	1	2
0			
1			
2			

IGameBoard gb = GameBoardFactory(3, 3, 3)
//Assume player = ‘X’

gb.placeMarker((0, 0), ‘X’)
gb.placeMarker((0, 1), ‘O’)
gb.placeMarker((0, 2), ‘X’)
gb.placeMarker((1, 0), ‘O’)
gb.placeMarker((1, 1), ‘X’)
gb.placeMarker((1, 2), ‘O’)
gb.placeMarker((2, 0), ‘X’)
gb.placeMarker((2, 1), ‘O’)
testPos = new BoardPosition (2, 2)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 3
getNumCols() = 3
getNumToWin() = 3
gb.checkDiagonalWin(testPos) = true

State:

	0	1	2
0	X		
1		X	
2			X

Reason:
This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top left corner down to bottom right corner.

Function Name:
`Test_Diagonal_Win_Start_at_Top_Left()`

```
default boolean checkDiagonalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

IGameBoard gb = GameBoardFactory(8, 8, 3)
//Assume player = 'X'

gb.placeMarker((1, 2), 'X')
gb.placeMarker((2, 3), 'X')
gb.placeMarker((3, 4), 'X')
gb.placeMarker((4, 5), 'X')
testPos = new BoardPosition (5, 6)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 8
getNumCols() = 8
getNumToWin() = 5
gb.checkDiagonalWin(testPos) = true

State:

	0	1	2	3	4	5	6	7
0								
1			X					
2				X				
3					X			
4						X		
5							X	
6								
7								

Reason:
This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top left corner down to bottom right corner.

Function Name:
`Test_Diagonal_Win_Middle_of_Larger_Board()`

```
default boolean checkDiagonalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb =
GameBoardFactory(10, 10, 5)
//Assume player = 'X'

gb.placeMarker((0, 0), 'X')
gb.placeMarker((1, 1), 'X')
gb.placeMarker((2, 2), 'X')
gb.placeMarker((3. 3), 'O')
testPos = new BoardPosition (4, 4)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkDiagonalWin(testPos) = false

State:

	0	1	2	3	4	5	6	7	8	9
0	X									
1		X								
2			X							
3				O						
4					X					
5										
6										
7										
8										
9										

Reason:
This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top left corner down to bottom right corner but is cut short because it encounters a different character before reaching 5 characters that matched 'X".

Function Name:
`Test_Diagonal_Win_Three_Out_of_Five_In_a_Row()`

```
default boolean checkDiagonalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb =
GameBoardFactory(10, 10, 5)
//Assume player = 'O'

gb.placeMarker((0, 0), 'X')
gb.placeMarker((1, 1), 'X')
gb.placeMarker((2, 2), 'X')
gb.placeMarker((3. 3), 'X')
testPos = new BoardPosition (4, 4)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkDiagonalWin(testPos) = false

State:

	0	1	2	3	4	5	6	7	8	9
0	X									
1		X								
2			X							
3				X						
4					O					
5										
6										
7										
8										
9										

Reason:
This test case is unique and distinct because it tests that the checkDiagonalWin function is scanning along a diagonal on the game board and returning true if it encounters >= numToWin, characters in a row. This case checks from top left corner down to bottom right corner but is cut short because it encounters a different character where the 5th matching character would have to be located.

Function Name:
`Test_Diagonal_Win_Four_Out_of_Five_In_a_Row()`

```
default boolean checkVerticalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb =
GameBoardFactory(10, 10, 5)
//Assume player = 'O'

gb.placeMarker((0, 0), 'X')
gb.placeMarker((1, 0), 'X')
gb.placeMarker((2, 0), 'X')
gb.placeMarker((3. 0), 'X')
testPos = new BoardPosition (4, 0)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkVerticalWin(testPos) = false

State:

	0	1	2	3	4	5	6	7	8	9
0	X									
1	X									
2	X									
3	X									
4	O									
5										
6										
7										
8										
9										

Reason:
This test case is unique and distinct because it tests that the checkVerticalWin function is accurately reading the first column on the board since it is a bound.

Function Name:
`Test_CheckVerticalWin_Far_Left_Column_One_Short()`

```
default boolean checkVerticalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb =
GameBoardFactory(10, 10, 5)
//Assume player = 'O'

gb.placeMarker((0, 9), 'X')
gb.placeMarker((1, 9), 'X')
gb.placeMarker((2, 9), 'X')
gb.placeMarker((3. 9), 'X')
testPos = new BoardPosition (4, 9)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkVerticalWin(testPos) = false

State:

	0	1	2	3	4	5	6	7	8	9
0										X
1										X
2										X
3										X
4										O
5										
6										
7										
8										
9										

Reason:
This test case is unique and distinct because it tests that the checkVerticalWin function is accurately reading the last column on the board since it is a bound.

Function Name:
`Test_CheckVerticalWin_Far_Right_Column_One_Short()`

```
default boolean checkVerticalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb =
GameBoardFactory(10, 10, 5)
//Assume player = 'X'

gb.placeMarker((0, 5), 'X')
gb.placeMarker((0, 0), 'O')
gb.placeMarker((1, 5), 'X')
gb.placeMarker((1, 1), 'O')
gb.placeMarker((2, 5), 'X')
gb.placeMarker((3, 4), 'O')
gb.placeMarker((3, 5), 'X')
gb.placeMarker((2, 7), 'O')
testPos = new BoardPosition (4, 5)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkVerticalWin(testPos) = true

State:

	0	1	2	3	4	5	6	7	8	9
0	O					X				
1		O				X				
2						X		O		
3					O	X				
4						X				
5										
6										
7										
8										
9										

Reason:
This test case is unique and distinct because it tests that the checkVerticalWin function is accurately detecting five characters in a row.

Function Name:
`Test_CheckVerticalWin_Middle_Column_Five_In_a_Row()`

```
default boolean checkVerticalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb =
GameBoardFactory(10, 10, 5)
//Assume player = 'X'

gb.placeMarker((0, 8), 'X')
gb.placeMarker((1, 8), 'X')
gb.placeMarker((2, 8), 'O')
gb.placeMarker((3, 8), 'X')
testPos = new BoardPosition (4, 8)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkVerticalWin(testPos) = false

State:

	0	1	2	3	4	5	6	7	8	9
0									X	
1									X	
2									O	
3									X	
4									X	
5										
6										
7										
8										
9										

Reason:
This test case is unique and distinct because it tests that the checkVerticalWin function stops scanning for matching characters when it encounters one that does not match.

Function Name:
`Test_CheckVerticalWin_Two_In_a_Row_Separated_by_One()`

```
default boolean checkHorizontalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb = GameBoardFactory(10, 10, 5)
//Assume player = 'O'

gb.placeMarker((0, 0), 'X')
gb.placeMarker((2, 2), 'O')
gb.placeMarker((0, 1), 'X')
gb.placeMarker((2, 4), 'O')
gb.placeMarker((0, 2), 'X')
gb.placeMarker((7, 7), 'O')
gb.placeMarker((0, 3), 'X')
gb.placeMarker((9, 5), 'O')
testPos = new BoardPosition (0, 4)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkHorizontalWin(testPos) = false

State:

	0	1	2	3	4	5	6	7	8	9
0	X	X	X	X	O					
1										
2			O		O					
3										
4										
5										
6										
7								O		
8										
9						O				

Reason:
This test case is unique and distinct because it tests that the checkHorizontalWin function is correctly detecting characters in the first row.

Function Name:
`Test_CheckHorizontalWin_Top_Row_One_Short()`

```
default boolean checkHorizontalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb =
GameBoardFactory(10, 10, 5)
//Assume player = ‘O’

gb.placeMarker((9, 0), ‘X’)
gb.placeMarker((2, 2), ‘O’)
gb.placeMarker((9, 1), ‘X’)
gb.placeMarker((2, 4), ‘O’)
gb.placeMarker((9, 2), ‘X’)
gb.placeMarker((7, 7), ‘O’)
gb.placeMarker((9, 3), ‘X’)
testPos = new BoardPosition (9, 4)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkHorizontalWin(testPos) = false

State:

	0	1	2	3	4	5	6	7	8	9
0										
1										
2			O		O					
3										
4										
5										
6										
7								O		
8										
9	X	X	X	X	O					

Reason:
This test case is unique and distinct because it tests that the checkHorizontalWin function is correctly detecting characters in the last row.

Function Name:
`Test_CheckHorizontalWin_Bottom_Row_One_Short()`

`default boolean checkHorizontalWin(BoardPosition lastPos)`

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb =
GameBoardFactory(10, 10, 5)
//Assume player = 'X'

gb.placeMarker((4, 3), X')
gb.placeMarker((4, 2), 'O')
gb.placeMarker((4, 4), 'X')
gb.placeMarker((5, 7), 'O')
gb.placeMarker((4. 5), 'X')
gb.placeMarker((7, 7), 'O')
gb.placeMarker((4, 6), 'X')
gb.placeMarker((9, 5), 'O')
testPos = new BoardPosition (4, 7)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkHorizontalWin(testPos) = true

State:

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4			O	X	X	X	X	X		
5								O		
6										
7								O		
8										
9						O				

Reason:
This test case is unique and distinct because it tests that the checkHorizontalWin function is correctly detecting characters in a row in the middle of the board.

Function Name:
`Test_CheckHorizontalWin_Middle_5_in_a_Row()`

```
default boolean checkHorizontalWin(BoardPosition lastPos)
```

Input:

State: (number to win = 5)

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										

IGameBoard gb =
GameBoardFactory(10, 10, 5)
//Assume player = ‘X’

gb.placeMarker((5, 0), X’)
gb.placeMarker((5, 1), ‘X’)
gb.placeMarker((5, 2), ‘O’)
gb.placeMarker((5, 3), ‘X’)
testPos = new BoardPosition (5, 4)
gb.placeMarker(testPos, player)

Output:

getNumRows() = 10
getNumCols() = 10
getNumToWin() = 5
gb.checkHorizontalWin(testPos) = false

State:

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5	X	X	O	X	X					
6										
7										
8										
9										

Reason:
This test case is unique and distinct because it tests that the checkHorizontalWin function stops scanning for matching characters when it encounters one that does not match.

Function Name:
`Test_CheckHorizontalWin_Two_In_a_Row_Separated_by_One()`