Kenley Shaw
CPSC 2150
Project Report

Requirements Analysis

Functional:

1. As a user, I can see whose turn it is in order to play my turn.
2. As a user, I can choose a location so that it can be placed on the board.
3. As a user, I can view the game board so that I can see the spaces that are available.
4. As a user, I can get notified if there are wins or draws so that I know why the game is over.
5. As a user, I can re-enter a location if it is unavailable so that I can take my turn correctly.
6. As a user, I can enter if I would like to play again so that the game can either start over or end.
7. As a user, I can enter locations that line up in a column so that I can win the game vertically.
8. As a user, I can enter locations that line up in a row so that I can win the game horizontally.
9. As a user, I can enter locations that line up diagonally in both different rows and columns so that I can win the game diagonally.
10. As a user, I can enter the last location on the board so that the game can end in a tie.
11. As a user, I can enter a row coordinate and a column coordinate so that the program receives my location.
12. As a user, I can see whose turn it is so that I am correctly taking my turn.
13. As a user, I can choose how I want to implement my board so that the game can either run fast or be memory efficient.
14. As a user, I can choose the dimensions of the board so that the array can vary in size.
15. As a user, I can choose how many players the game will have so that multiple people can play.
16. As a user, I can choose my character for my player so that each player has different characters.

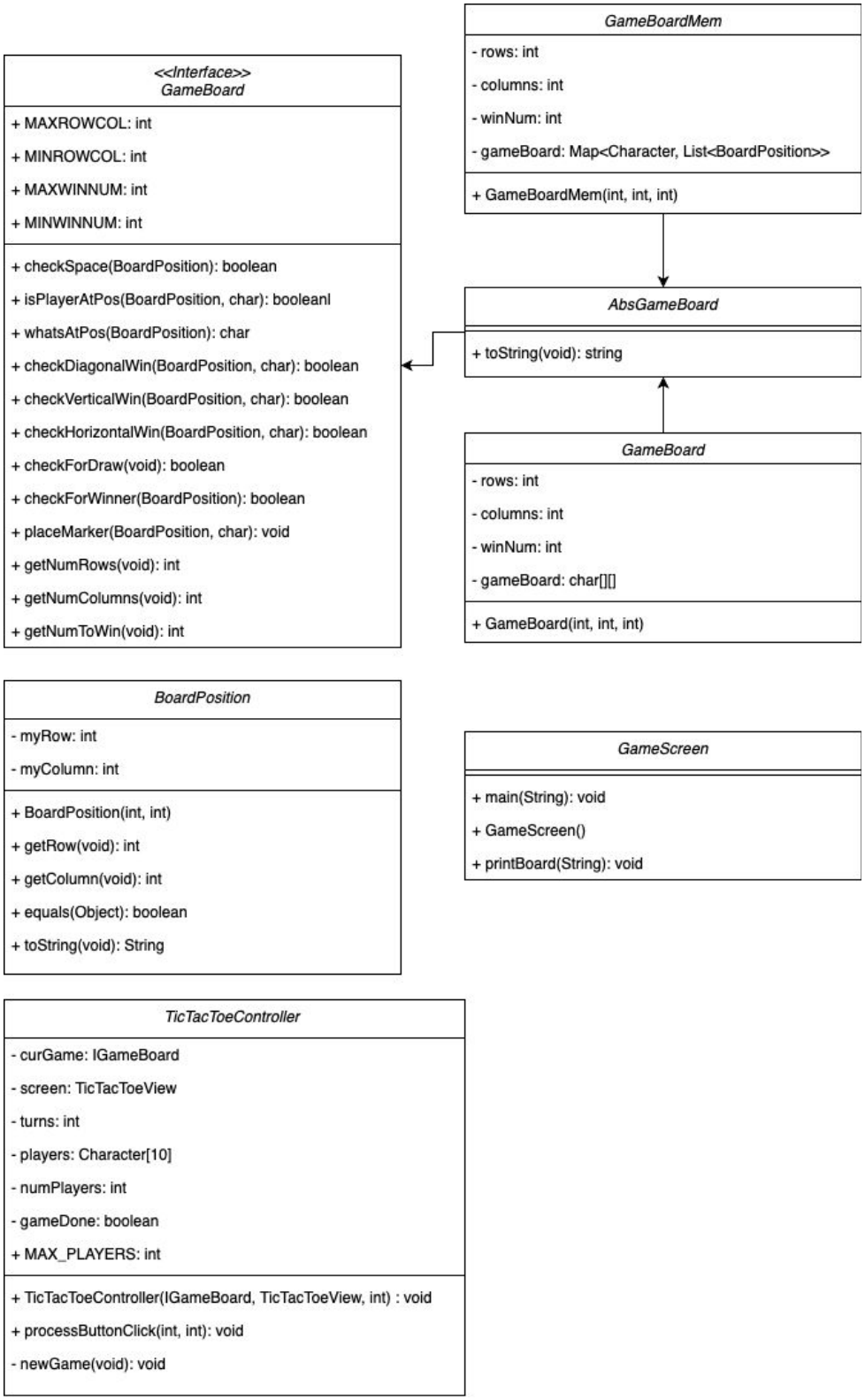
Non-Functional:

1. The game must run as a Graphical User Interface.
2. The game must be programmed using Java.
3. The system is clear in it’s prompts to the user.
4. The system uses the GUI to receive user input.
5. The system uses the GUI to output manipulations of data.
6. The system was created using the application IntelliJ Idea.
7. The system keeps track of all user interactions.
8. The system runs with no errors.
9. The board can not be greater than 20 rows by 20 columns.

10. The board can not be less than 3 rows by 3 columns.
11. The winning number can not be greater than 20.
12. The winning number can not be less than 3.
13. The number of players can not be greater than 10.
14. The number of players can not be less than 10.

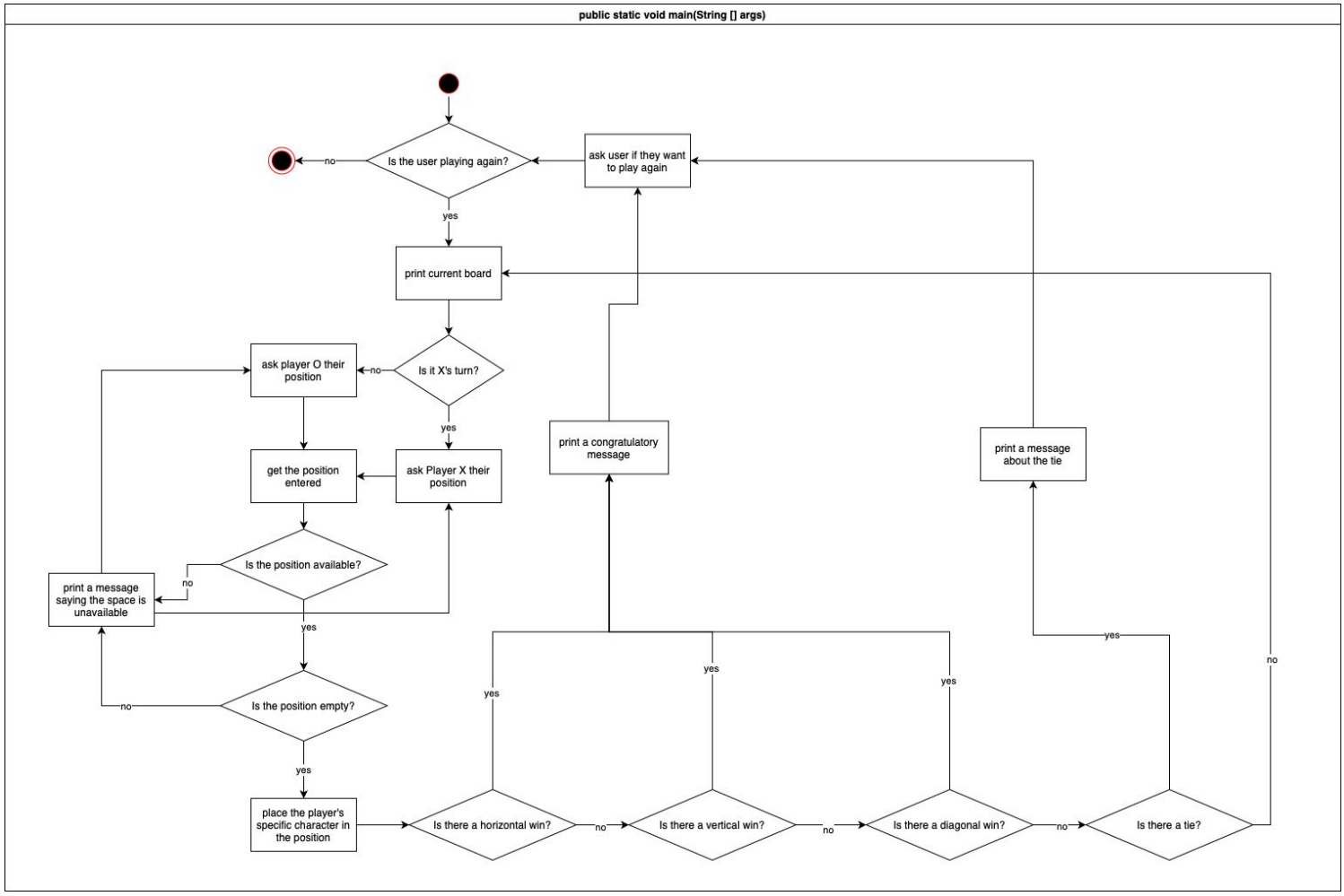
Design

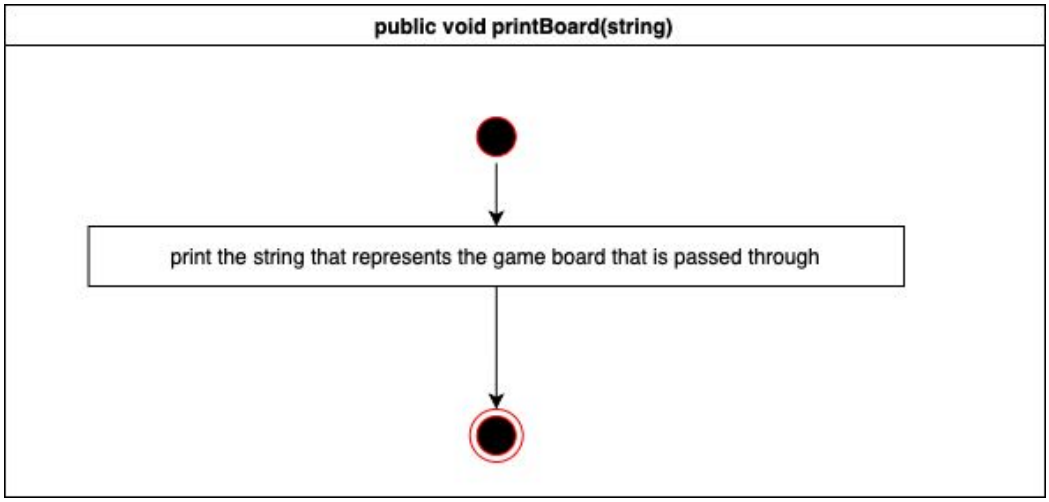
1. UML class diagrams

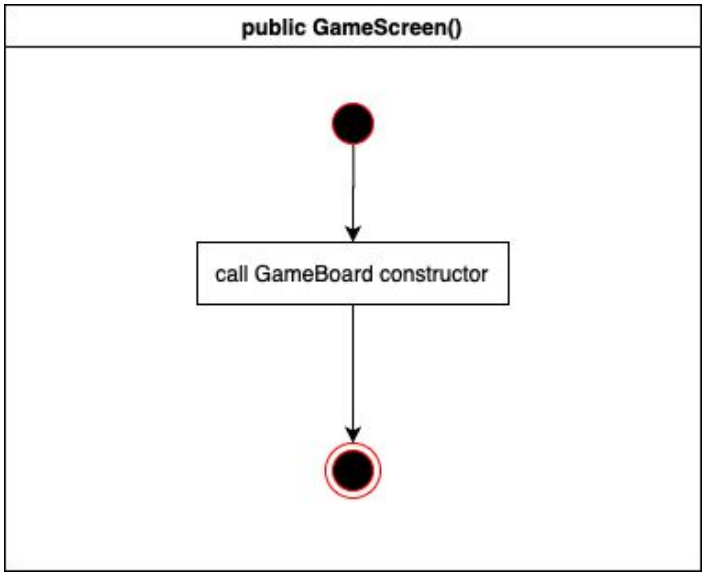


2. UML Activity Diagrams

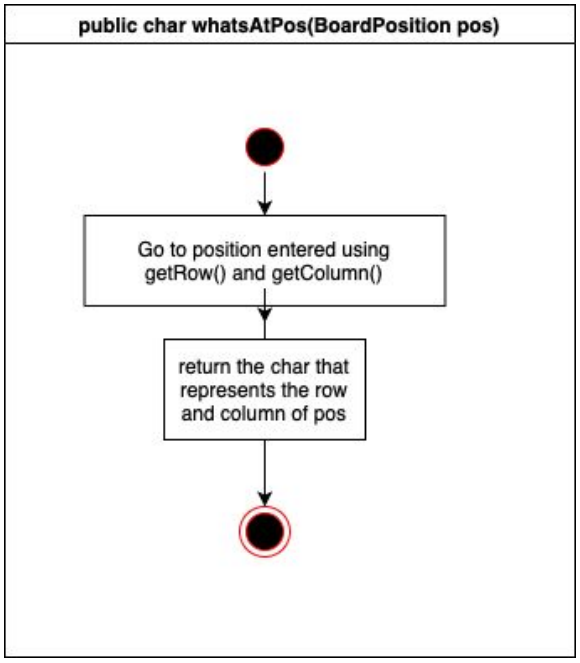
Activity diagrams for methods in GameScreen

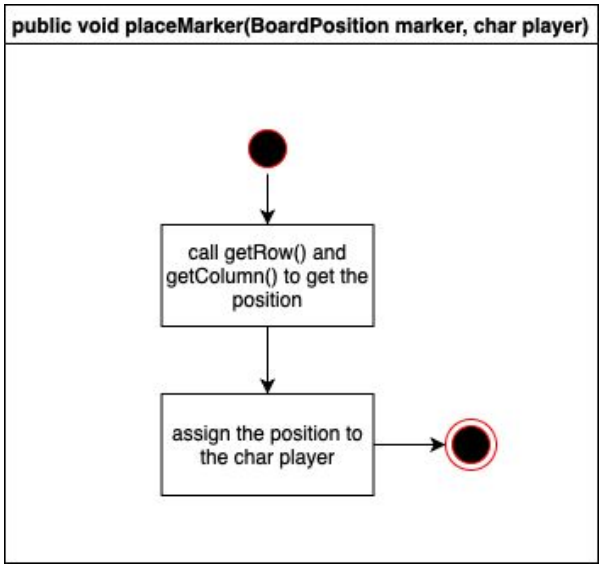


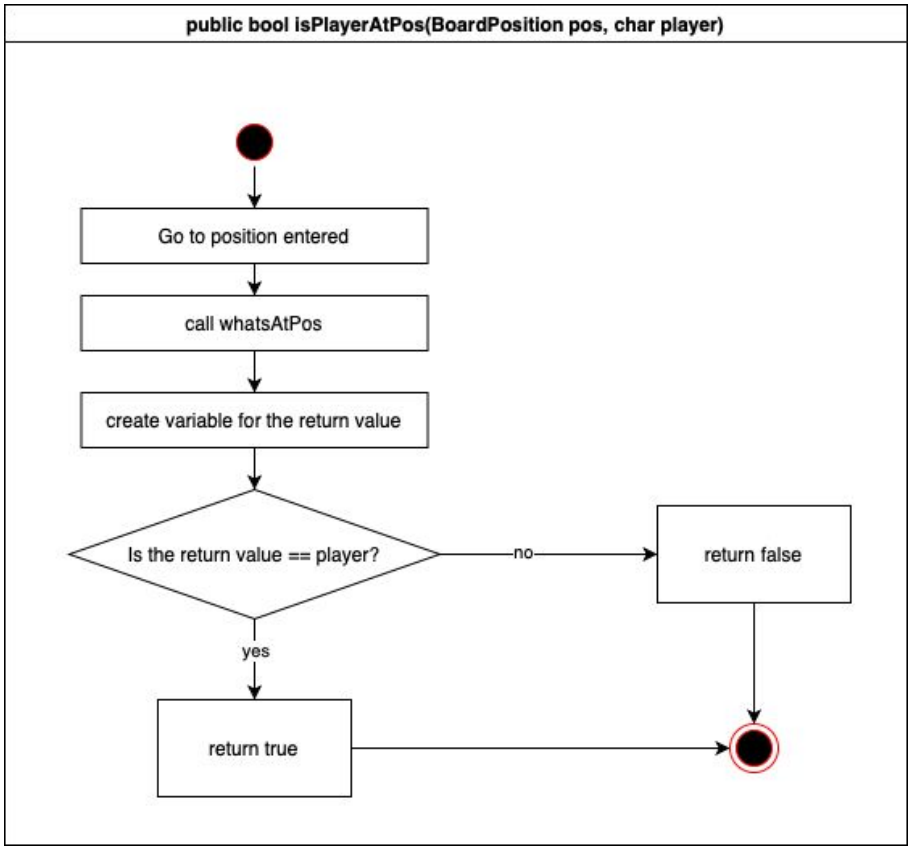


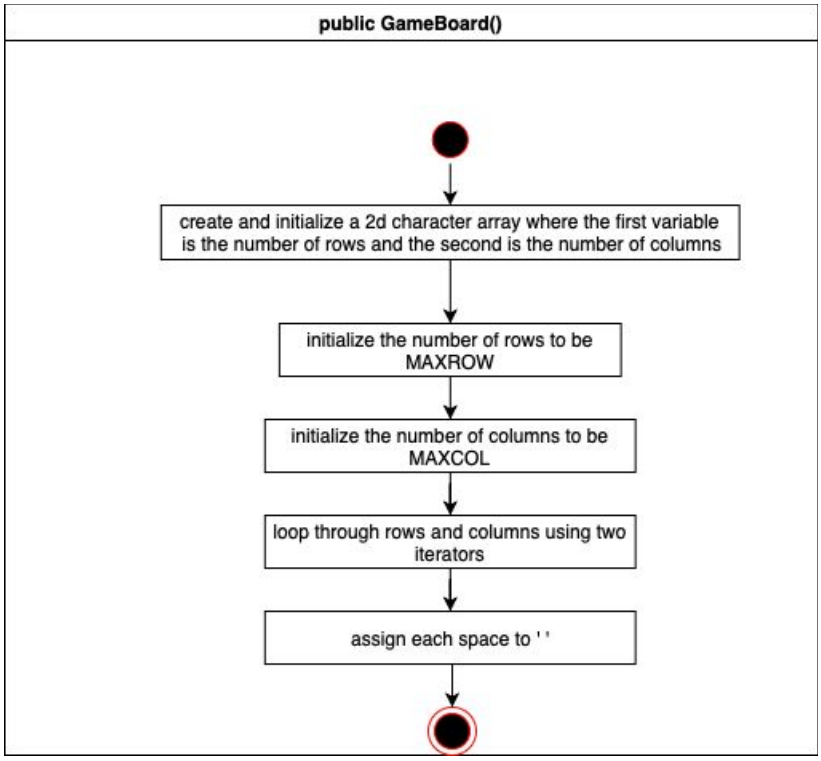


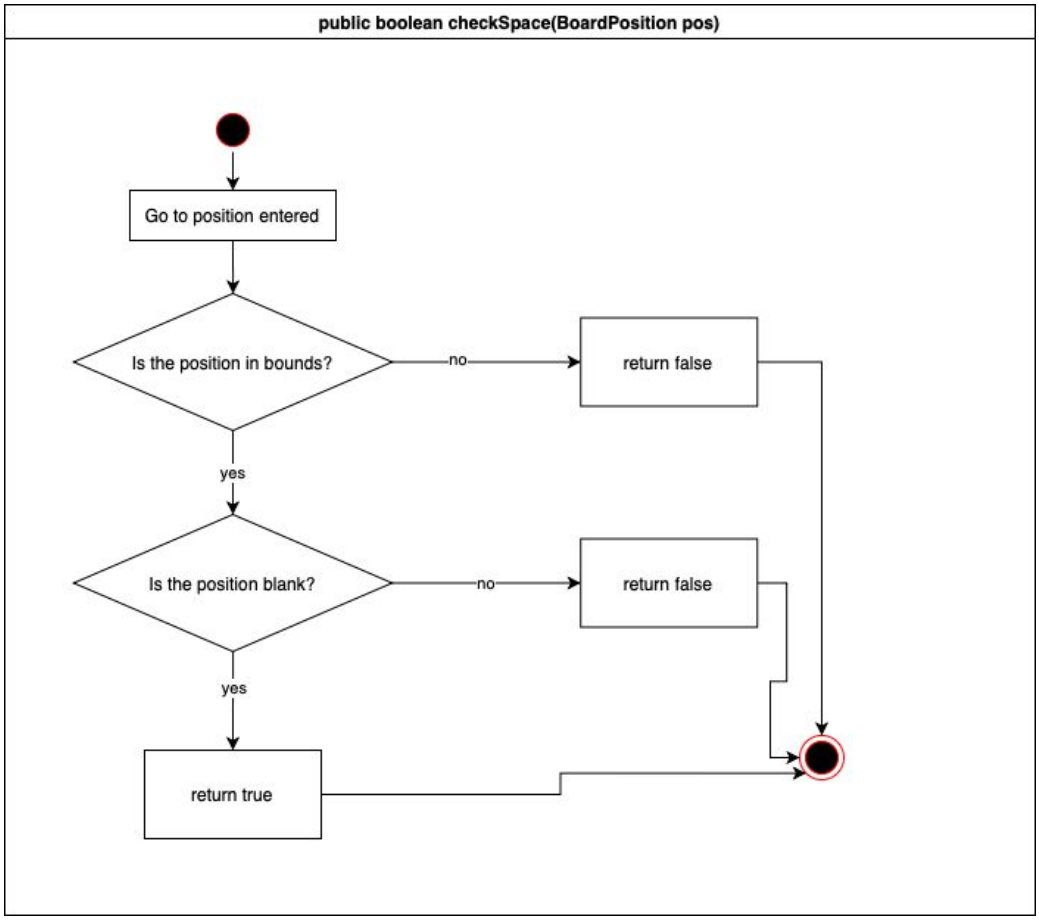
Activity diagrams for methods in GameBoard

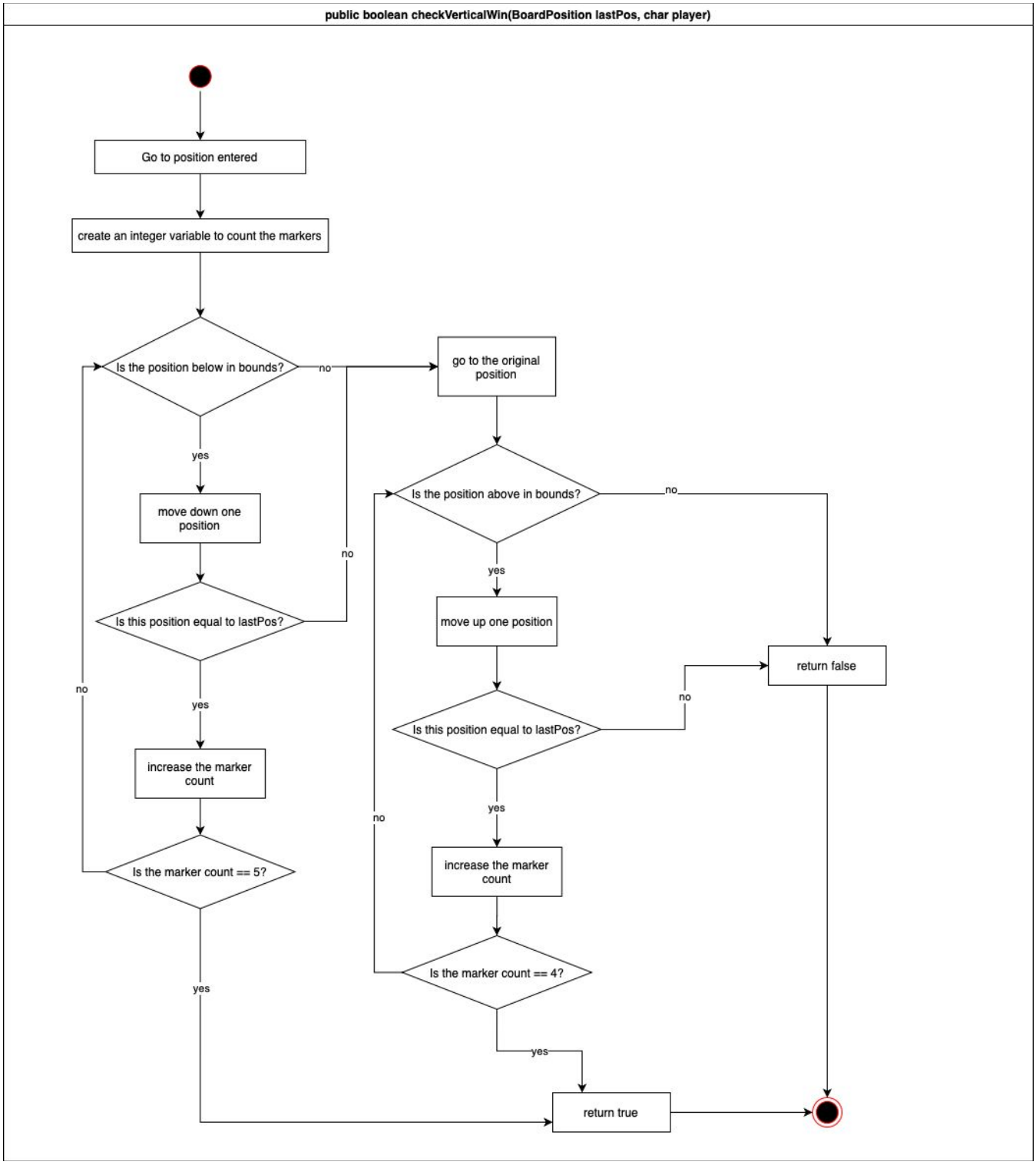


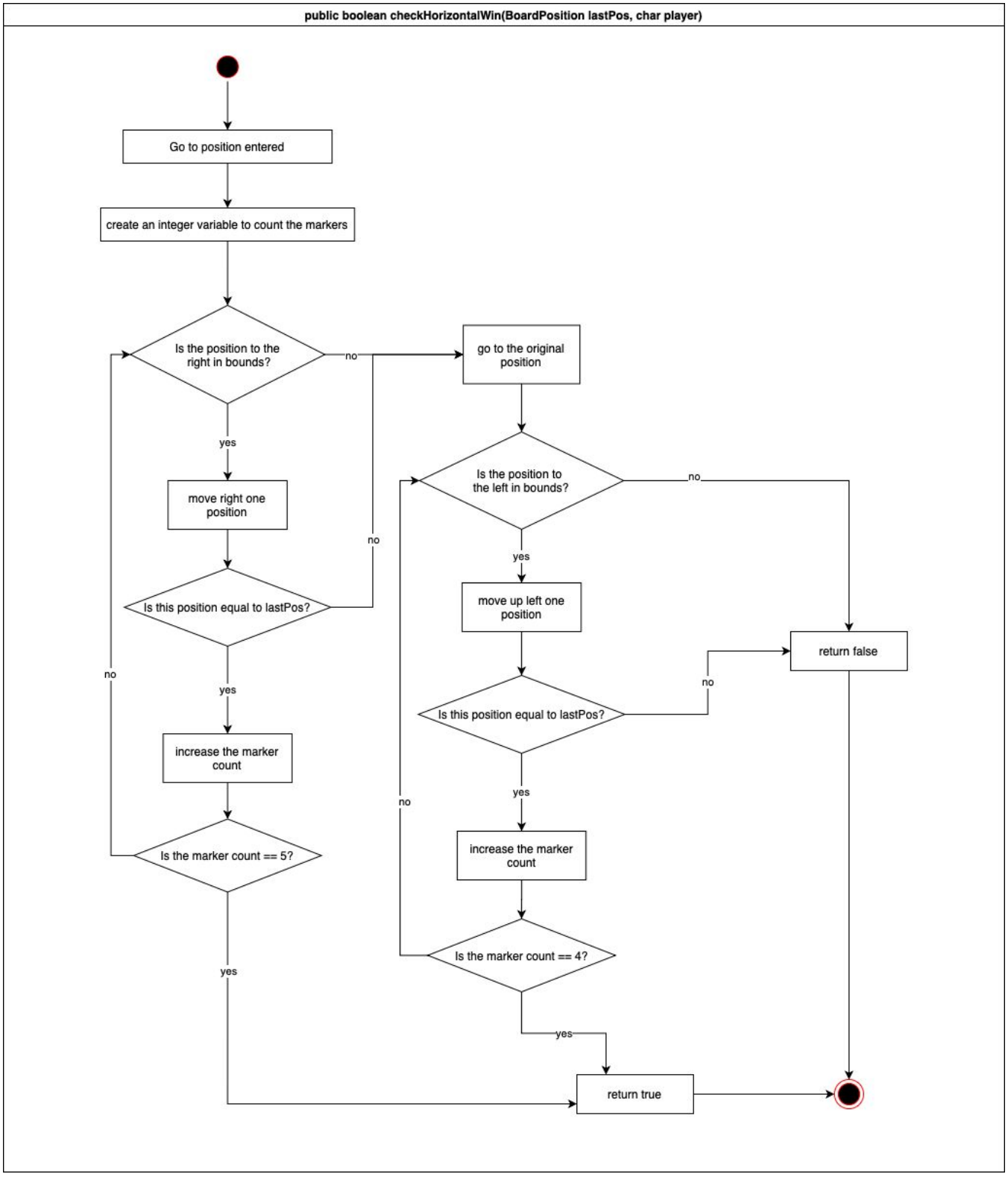

public void placeMarker(BoardPosition marker, char player)
call getRow() and getColumn() to get the position
assign the position to the char player

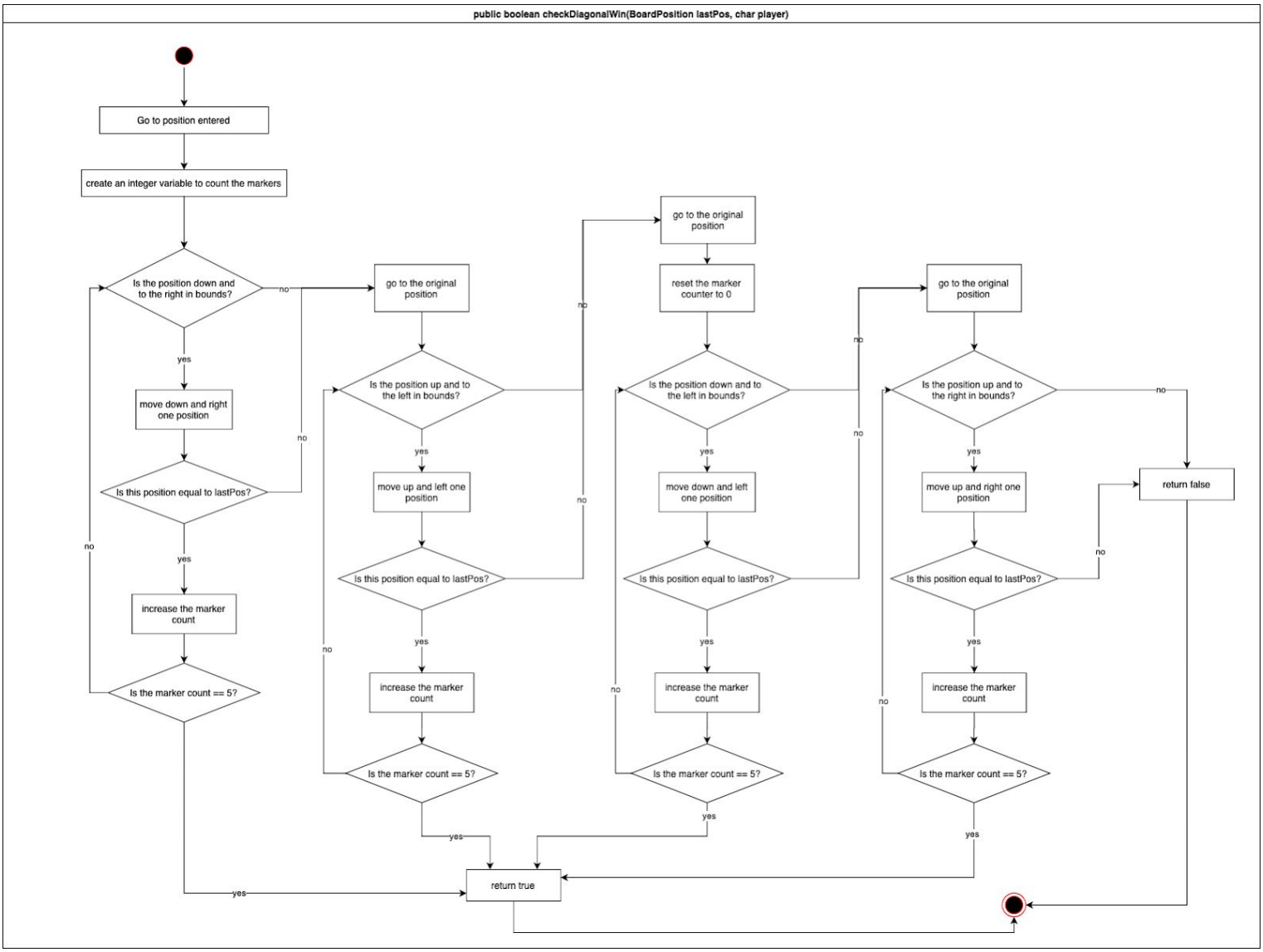

public bool isPlayerAtPos(BoardPosition pos, char player)
Go to position entered
call whatsAtPos
create variable for the return value
Is the return value == player?
no
return false
yes
return true

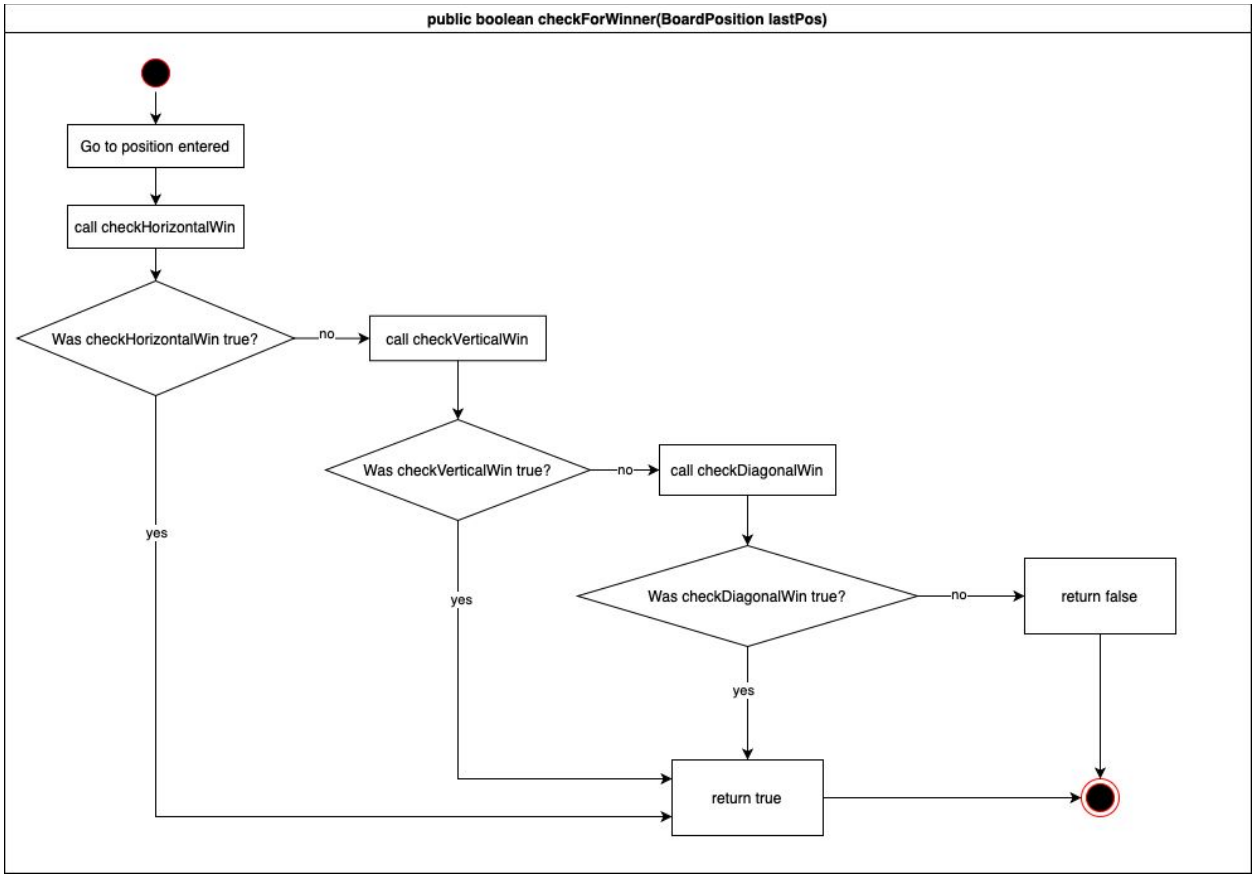

public GameBoard()
create and initialize a 2d character array where the first variable is the number of rows and the second is the number of columns
initialize the number of rows to be MAXROW
initialize the number of columns to be MAXCOL
loop through rows and columns using two iterators
assign each space to ' '

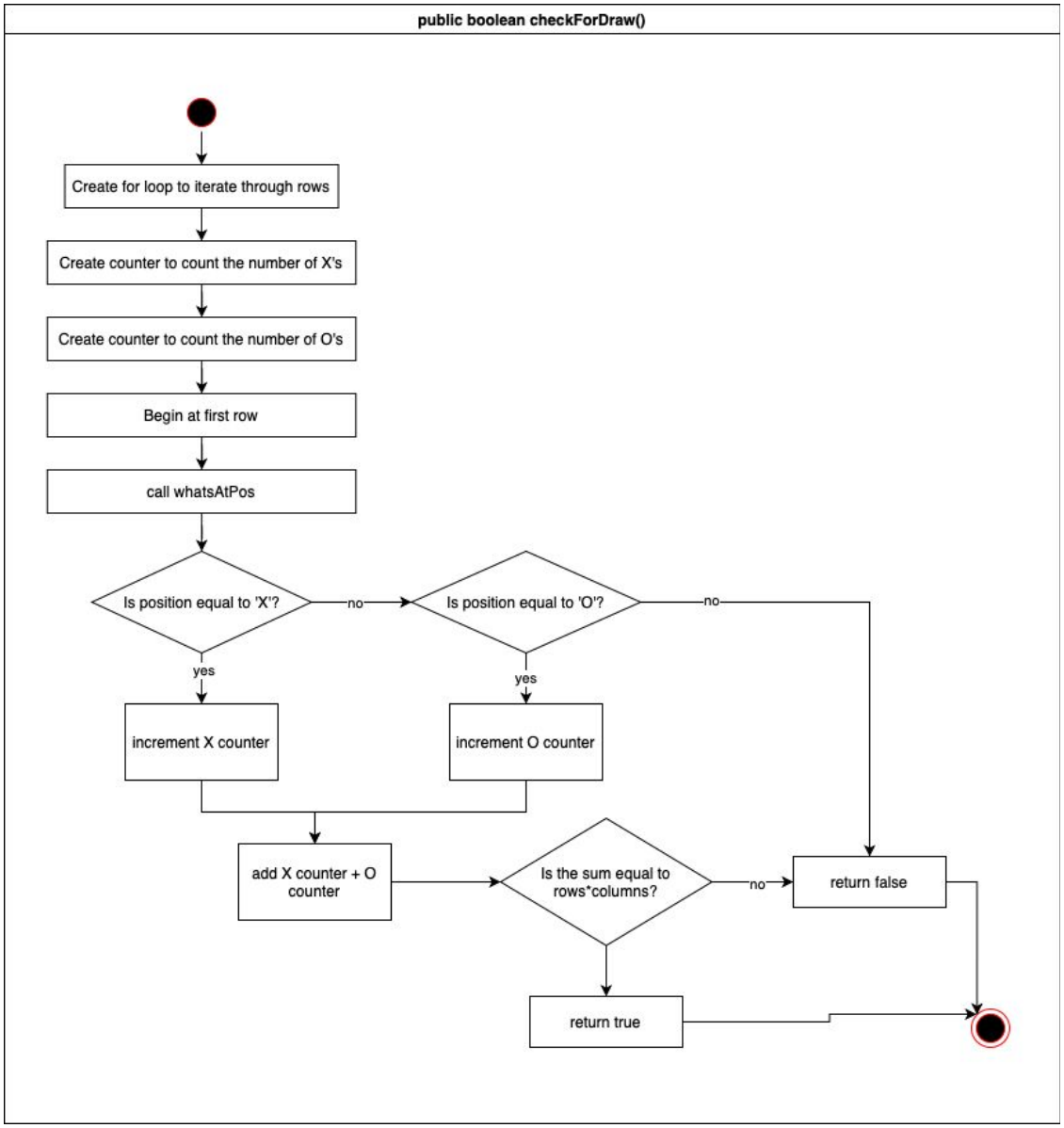

public boolean checkSpace(BoardPosition pos)
Go to position entered
Is the position in bounds?
no
return false
yes
Is the position blank?
no
return false
yes
return true

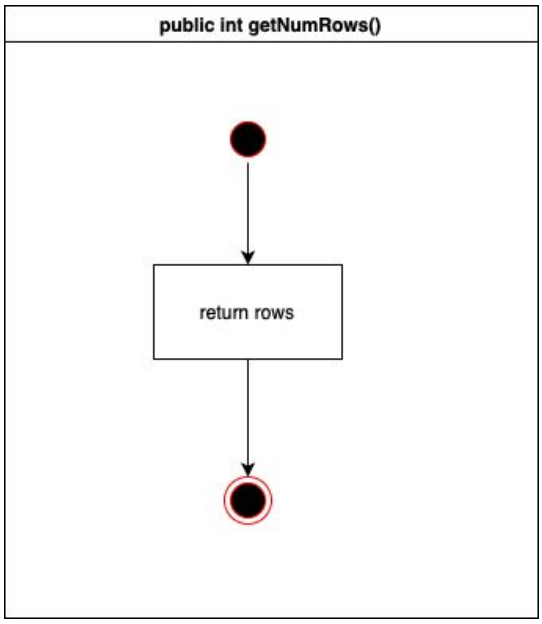

public boolean checkVerticalWin(BoardPosition lastPos, char player)
Go to position entered
create an integer variable to count the markers
Is the position below in bounds?
no
yes
move down one position
Is this position equal to lastPos?
no
yes
increase the marker count
Is the marker count == 5?
no
yes
go to the original position
Is the position above in bounds?
no
yes
move up one position
Is this position equal to lastPos?
no
yes
increase the marker count
Is the marker count == 4?
no
yes
return false
return true

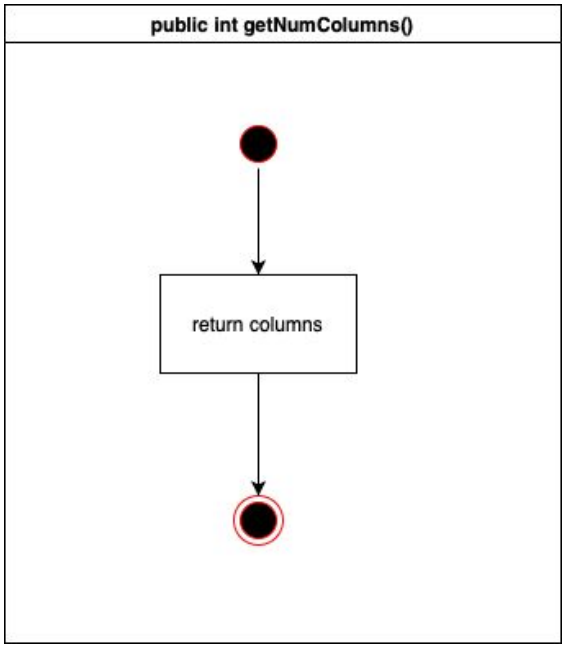

public boolean checkHorizontalWin(BoardPosition lastPos, char player)
Go to position entered
create an integer variable to count the markers
Is the position to the right in bounds?
no
yes
move right one position
Is this position equal to lastPos?
no
yes
increase the marker count
Is the marker count == 5?
no
yes
go to the original position
Is the position to the left in bounds?
no
yes
move up left one position
Is this position equal to lastPos?
no
yes
increase the marker count
Is the marker count == 4?
no
yes
return false
return true

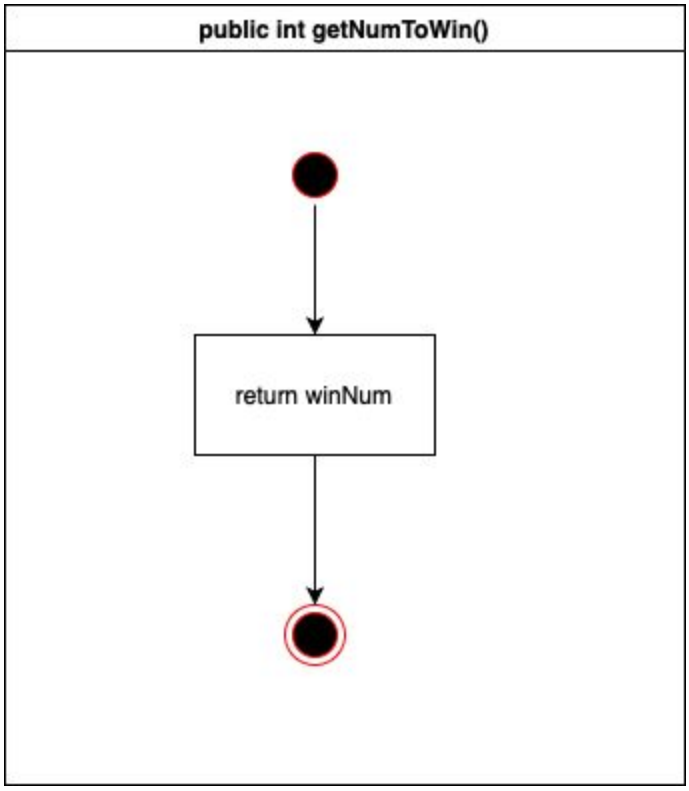

public boolean checkDiagonalWin(BoardPosition lastPos, char player)
Go to position entered
create an integer variable to count the markers
Is the position down and to the right in bounds?
no
yes
move down and right one position
Is this position equal to lastPos?
no
yes
increase the marker count
Is the marker count == 5?
no
yes
go to the original position
Is the position up and to the left in bounds?
no
yes
move up and left one position
Is this position equal to lastPos?
no
yes
increase the marker count
Is the marker count == 5?
no
yes
go to the original position
reset the marker counter to 0
Is the position down and to the left in bounds?
no
yes
move down and left one position
Is this position equal to lastPos?
no
yes
increase the marker count
Is the marker count == 5?
no
yes
go to the original position
Is the position up and to the right in bounds?
no
yes
move up and right one position
Is this position equal to lastPos?
no
yes
increase the marker count
Is the marker count == 5?
no
yes
return false
return true


public boolean checkForWinner(BoardPosition lastPos)
Go to position entered
call checkHorizontalWin
Was checkHorizontalWin true?
no
call checkVerticalWin
yes
Was checkVerticalWin true?
no
call checkDiagonalWin
yes
Was checkDiagonalWin true?
no
return false
yes
return true

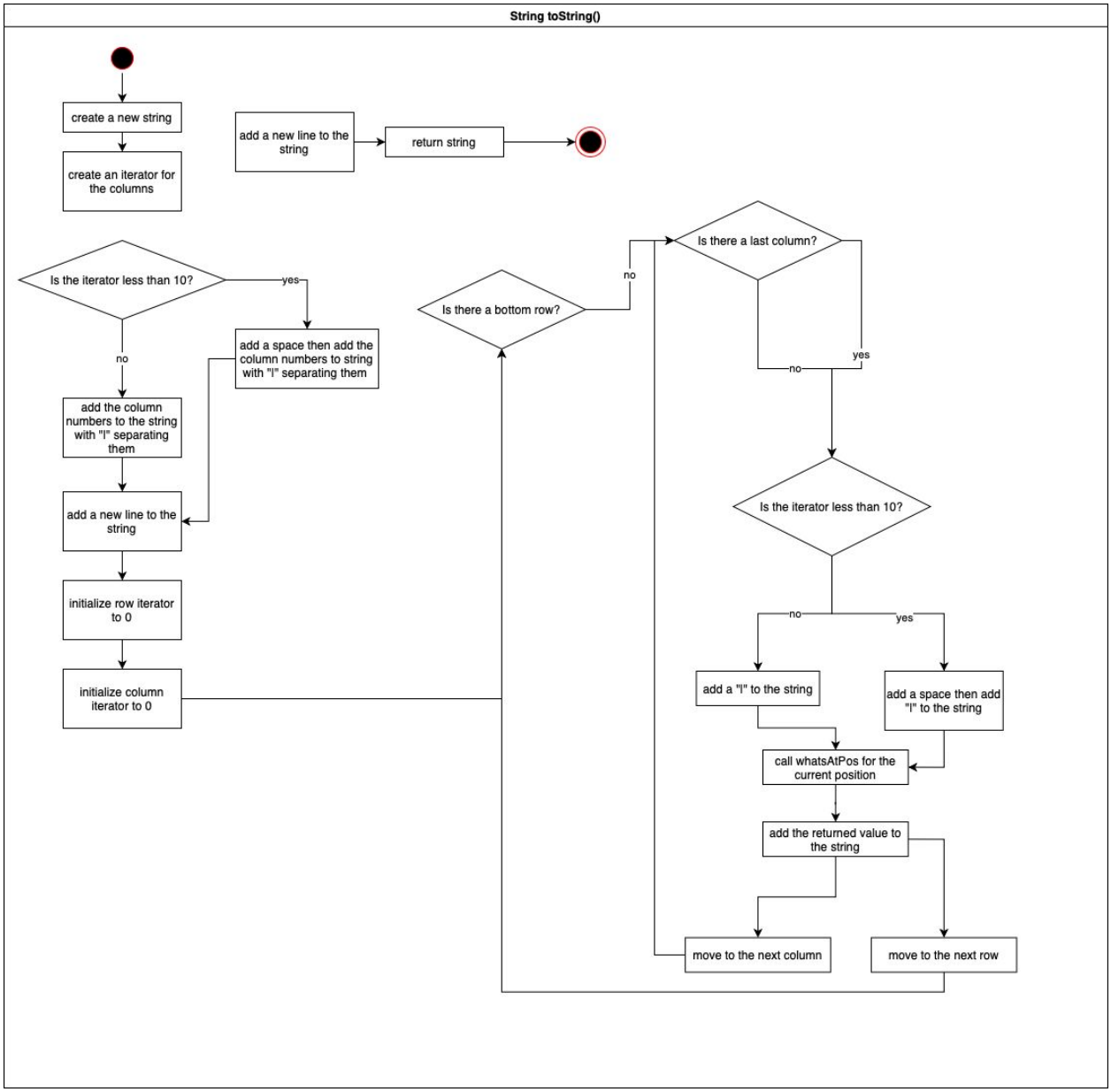

public boolean checkForDraw()
Create for loop to iterate through rows
Create counter to count the number of X's
Create counter to count the number of O's
Begin at first row
call whatsAtPos
Is position equal to 'X'?
no
Is position equal to 'O'?
no
yes
yes
increment X counter
increment O counter
add X counter + O counter
Is the sum equal to rows*columns?
no
return false
return true


public int getNumRows()
return rows

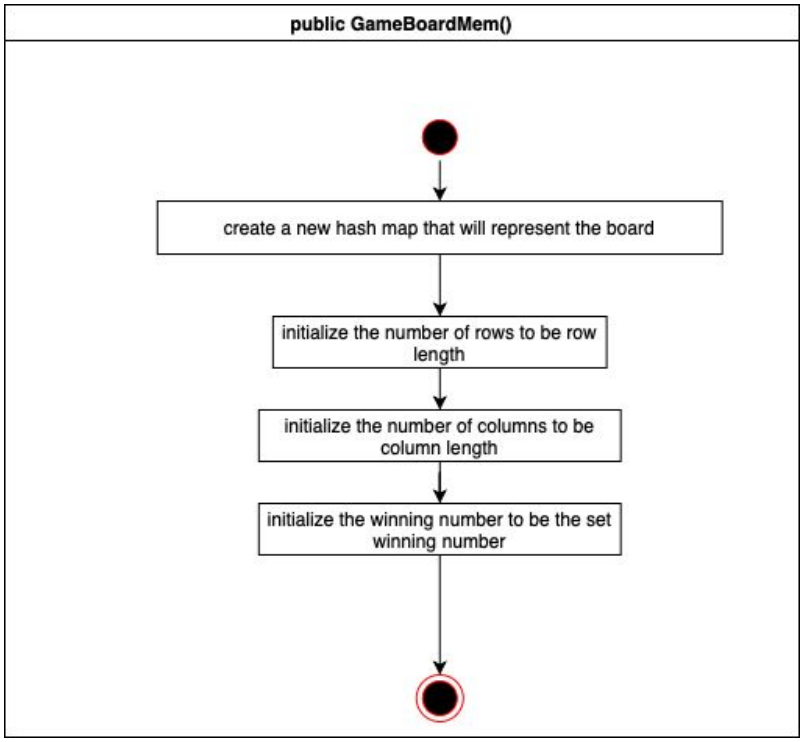

public int getNumColumns()
return columns

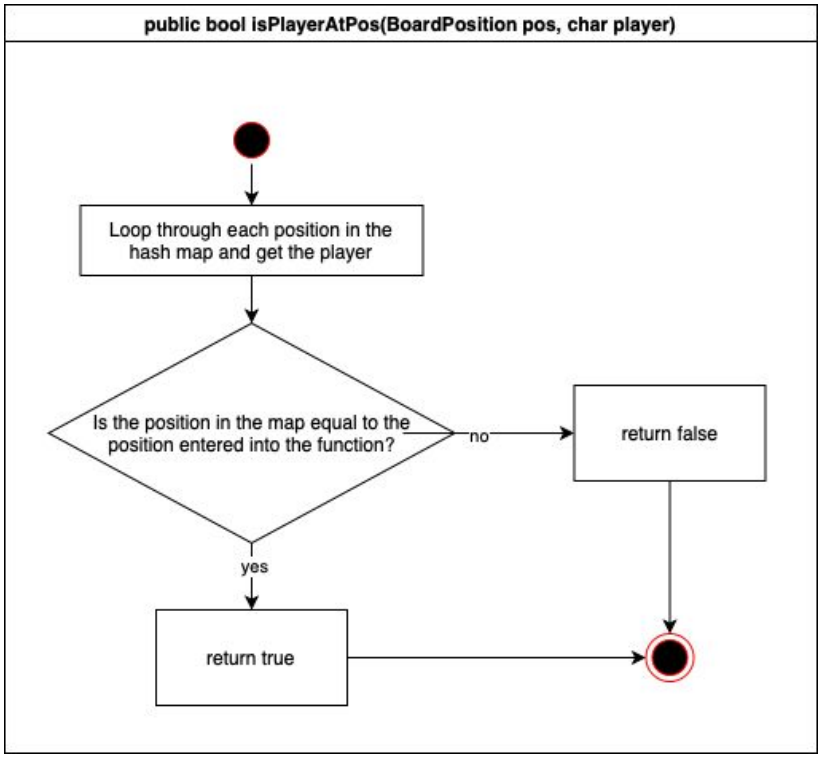

public int getNumToWin()
return winNum

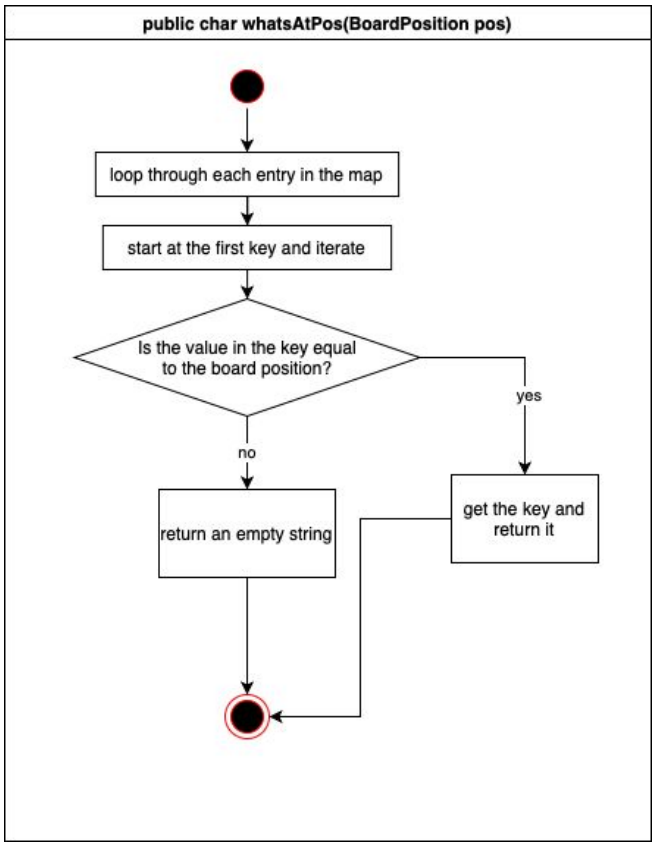
Activity diagram for method in AbsGameBoard

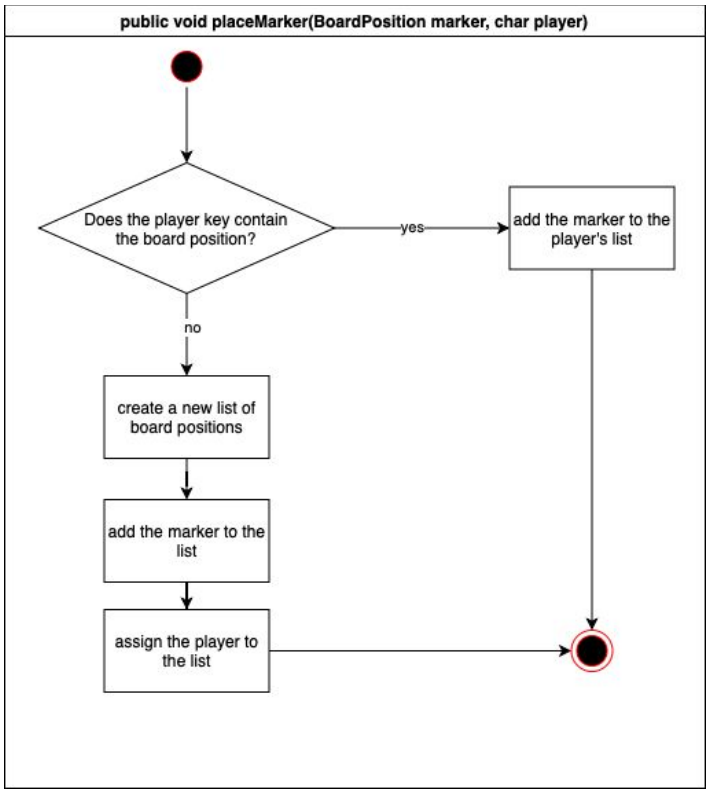


Activity diagrams for methods in GameBoardMem

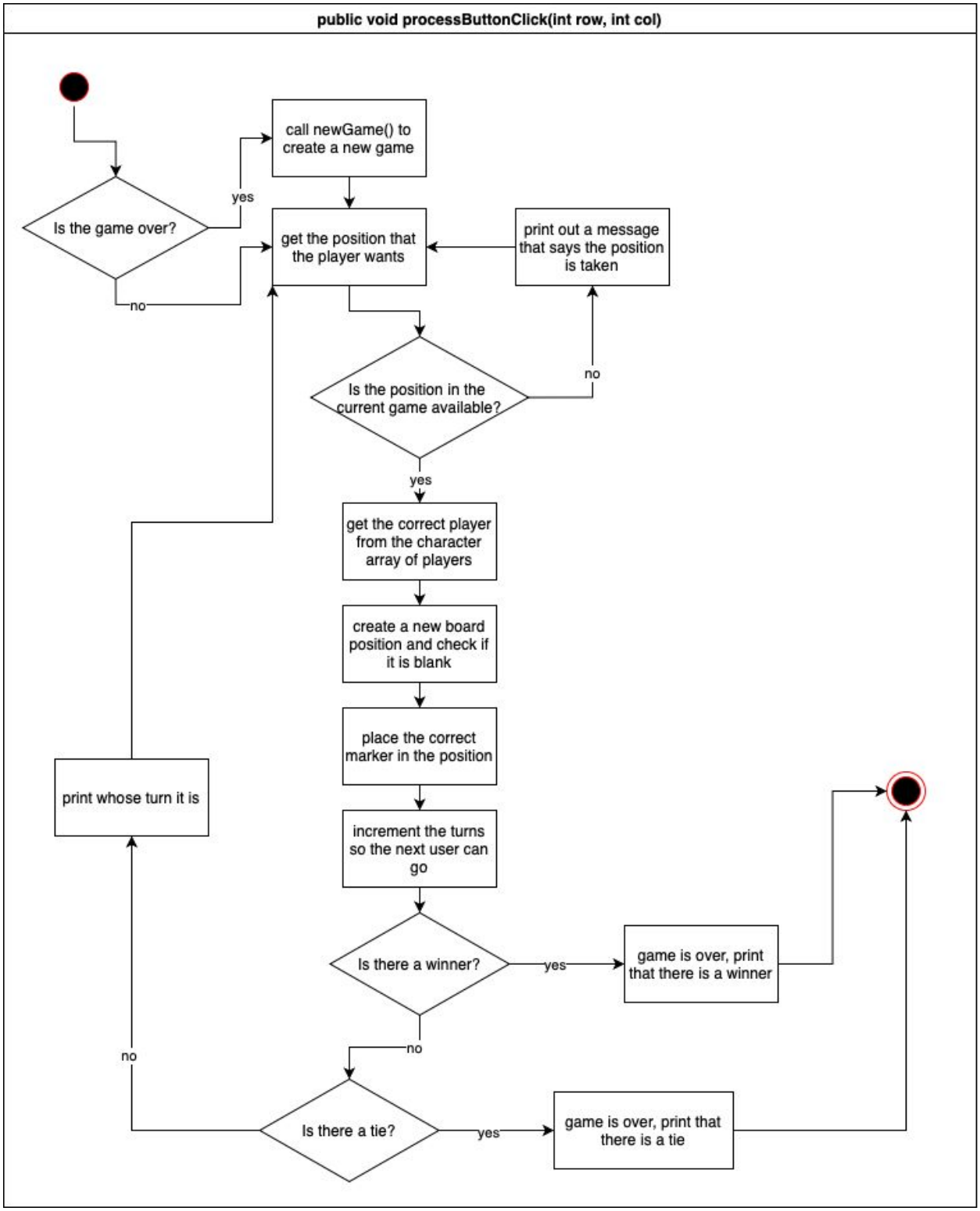





public char whatsAtPos(BoardPosition pos)
loop through each entry in the map
start at the first key and iterate
Is the value in the key equal to the board position?
yes
no
return an empty string
get the key and return it


public void placeMarker(BoardPosition marker, char player)
Does the player key contain the board position?
yes
add the marker to the player's list
no
create a new list of board positions
add the marker to the list
assign the player to the list

Activity Diagrams for methods in TicTacToeController class



Testing

public GameBoard(int rowLength, int colLength, int winnerNum)

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>State: uninitialized

Row = 3
Column = 3
winNum = 3</td>
<td>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>This function is distinct because it tests that the board can be sized to the minimum dimensions and the number to win can be the minimum number.
<table>
<tr><td>Function Name</td></tr>
<tr><td>testConstructorMin</td></tr>
</table>
</td>
</tr>
</table>

Input	Output	Reason
State: uninitialized Row = 100 Column = 100 winNum = 25	State: too large to show, but it results in a 100x100 board that takes 25 numbers in a row to win	This function is distinct because it tests that the board can be sized to the maximum dimensions and the number to win can be the maximum number. Function Name testConstructorMax

Input	Output	Reason
State: uninitialized Row = 30 Column = 40 winNum = 10	State: too large to show, but it results in a 30x40 board that takes 10 numbers in a row to win	This function is distinct because it tests that the board can be sized to any random dimensions within bounds and the number to win can be any number within bounds.

		Function Name testConstructorMiddle

default boolean checkSpace(BoardPosition pos)

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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></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>
pos.getRow() = 0
pos.getColumn() = 0</td>
<td>State:
checkSpace = false,
State of board is unchanged</td>
<td>This function is distinct because it tests that the space is already occupied by a character.

<table><tr><td>Function Name</td></tr><tr><td>testCheckSpaceTaken</td></tr></table></td>
</tr>
</table>

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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></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>
pos.getRow() = 2
pos.getColumn() = 0</td>
<td>State:
checkSpace = true,
State of board is unchanged</td>
<td>This function is distinct because it tests that the space is empty when it is in the maximum row.

<table><tr><td>Function Name</td></tr><tr><td>testCheckSpaceEmptyRow</td></tr></table></td>
</tr>
</table>

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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></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>
pos.getRow() = 0
pos.getColumn() = 2</td>
<td>State:
checkSpace = true,
State of board is unchanged</td>
<td>This function is distinct because it tests that the space is empty when it is in the maximum col.

<table><tr><td>Function Name</td></tr><tr><td>testCheckSpaceEmptyCol</td></tr></table></td>
</tr>
</table>

default boolean checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>O</td><td></td><td></td></tr><tr><td>2</td><td>O</td><td></td><td></td></tr></table>
p = ‘X’
pos.getRow() = 0
pos.getColumn() = 0</td>
<td>State:
checkHorizontalWin = true,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is a horizontal win to the right of the last character placed the farthest left.
<table><tr><td>Function Name</td></tr><tr><td>testCheckHorizontalWinLeft</td></tr></table></td>
</tr>
</table>

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>O</td><td></td><td></td></tr><tr><td>2</td><td>O</td><td></td><td></td></tr></table>
p = ‘X’
pos.getRow() = 0
pos.getColumn() = 2</td>
<td>State:
checkHorizontalWin = true,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is a horizontal win to the left of the last character placed the farthest right.
<table><tr><td>Function Name</td></tr><tr><td>testCheckHorizontalWinRight</td></tr></table></td>
</tr>
</table>

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>O</td><td></td><td></td></tr><tr><td>2</td><td>O</td><td></td><td></td></tr></table>
p = ‘X’</td>
<td>State:
checkHorizontalWin = true,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is a horizontal win to both the left and right of the last character placed in the middle of a row.
<table><tr><td>Function Name</td></tr></table></td>
</tr>
</table>

pos.getRow() = 0 pos.getColumn() = 1		testCheckHorizontalWinMiddle

Input	Output	Reason
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>O</td></tr><tr><td>1</td><td>O</td><td></td><td></td></tr><tr><td>2</td><td>X</td><td></td><td></td></tr></table> p = ‘X’ pos.getRow() = 0 pos.getColumn() = 0	State: checkHorizontalWin = false, State of board is unchanged	This function is distinct because it tests that there is not a horizontal win to both the left or right of the last character placed. Function Name testCheckHorizontalWinFalse

default boolean checkVerticalWin(BoardPosition lastPos, char player)

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | X | O | O | | 1 | X | | | | 2 | X | | | p = 'X' pos.getRow() = 0 pos.getColumn() = 0	State: checkHorizontalWin = true, State of board is unchanged	This function is distinct because it tests that there is a vertical win below the last character placed in the top position in the column. | Function Name | | testCheckVerticalWinDown |

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | X | O | O | | 1 | X | | | | 2 | X | | | p = 'X' pos.getRow() = 2 pos.getColumn() = 0	State: checkHorizontalWin = true, State of board is unchanged	This function is distinct because it tests that there is a vertical win above the last character placed in the bottom position in the column. | Function Name | | testCheckVerticalWinUp |

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | X | O | O | | 1 | X | | | | 2 | X | | | p = 'X'	State: checkHorizontalWin = true, State of board is unchanged	This function is distinct because it tests that there is a vertical win below and above the last character placed in a middle position in the column. | Function Name |

pos.getRow() = 1 pos.getColumn() = 0		testCheckVerticalWinMiddle

Input	Output	Reason
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>O</td></tr><tr><td>1</td><td>O</td><td></td><td></td></tr><tr><td>2</td><td>X</td><td></td><td></td></tr></table> p = ‘X’ pos.getRow() = 0 pos.getColumn() = 0	State: checkHorizontalWin = false, State of board is unchanged	This function is distinct because it tests that there is not a vertical win below or above the last character placed in the column. Function Name testCheckVerticalWinFalse

default boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | X | | O | | 1 | O | X | | | 2 | | | X | p = 'X' pos.getRow() = 0 pos.getColumn() = 0	State: checkDiagonalWin = true, State of board is unchanged	This function is distinct because it tests that there is a diagonal win beginning from the last character placed in the top left corner so that the only characters that have to be checked are in positions below and to the right of it. | Function Name | | testCheckDiagonalWinLeftTopLeft |

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | X | | O | | 1 | O | X | | | 2 | | | X | p = 'X' pos.getRow() = 2 pos.getColumn() = 2	State: checkDiagonalWin = true, State of board is unchanged	This function is distinct because it tests that there is a diagonal win beginning from the last character placed in the bottom right corner so that the only characters that have to be checked are in positions above and to the left of it. | Function Name | | testCheckDiagonalWinLeftBottomRight |

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | X | | O |	State: checkDiagonalWin = true, State of board is unchanged	This function is distinct because it tests that there is a diagonal win beginning from the last character placed

1	O	X	
2			X

p = ‘X’
pos.getRow() = 1
pos.getColumn() = 1

in the middle of the board so that both characters above and to the left and below and to the right have to be checked.

Function Name
testCheckDiagonalWinLeftMiddle

Input	Output	Reason
State: p = ‘X’ pos.getRow() = 0 pos.getColumn() = 2	State: checkDiagonalWin = true, State of board is unchanged	This function is distinct because it tests that there is a diagonal win beginning from the last character placed in the top right corner so that the only characters that have to be checked are in positions below and to the left of it.

	0	1	2
0	O		X
1	O	X	
2	X		

Function Name
testCheckDiagonalWinRightTopRight

Input	Output	Reason
State: p = ‘X’ pos.getRow() = 2 pos.getColumn() = 0	State: checkDiagonalWin = true, State of board is unchanged	This function is distinct because it tests that there is a diagonal win beginning from the last character placed in the bottom left corner so that the only characters that have to be checked are in positions above and to the right of it.

	0	1	2
0	O		X
1	O	X	
2	X		

Function Name

		testCheckDiagonalWinRighttBottomLeft

Input	Output	Reason
State: p = ‘X’ pos.getRow() = 1 pos.getColumn() = 1	State: checkDiagonalWin = true, State of board is unchanged	This function is distinct because it tests that there is a diagonal win beginning from the last character placed in the middle of the board so that both characters above and to the right and below and to the left have to be checked. Function Name: testCheckDiagonalWinRighttMiddle

	0	1	2
0	O		X
1	O	X	
2	X		

Input	Output	Reason
State: p = ‘X’ pos.getRow() = 0 pos.getColumn() = 0	State: checkDiagonalWin = false, State of board is unchanged	This function is distinct because it tests that there is a not a diagonal win beginning from the last character placed in the top left corner after checking all characters below and to the right of it. Function Name: testCheckDiagonalWinRighttFalse

	0	1	2
0	X	O	
1	O	X	
2	X		

default boolean checkForDraw()

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>State:
checkForDraw = false,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is a not a draw when the board contains no characters and is all empty spaces, and there is no win.
<table><tr><td>Function Name</td></tr><tr><td>testCheckForDrawFalse</td></tr></table></td>
</tr>
</table>

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>X</td></tr><tr><td>2</td><td>O</td><td>X</td><td>O</td></tr></table></td>
<td>State:
checkForDraw = true,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is a draw when all of the spaces on the board contain characters so every space is checked, and there is no win.
<table><tr><td>Function Name</td></tr><tr><td>testCheckForDrawFull</td></tr></table></td>
</tr>
</table>

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>X</td></tr><tr><td>2</td><td>O</td><td>X</td><td>O</td></tr><tr><td>3</td><td></td><td></td><td></td></tr></table></td>
<td>State:
checkForDraw = false,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is a not a draw when all of the board contains characters except for the bottom row that signifies there are empty spaces in a row being checked.
<table><tr><td>Function Name</td></tr></table></td>
</tr>
</table>

		testCheckForDrawEmptyMaxRow

Input	Output	Reason
State: | | 0 | 1 | 2 | 3 | | 0 | X | O | X | | | 1 | O | X | X | | | 2 | O | X | O | |	State: checkForDraw = false, State of board is unchanged	This function is distinct because it tests that there is a not a draw when all of the board contains characters except for the rightmost column that signifies there are empty spaces in a column being checked. Function Name: testCheckForDrawEmptyMaxCol

public char whatsAtPos(BoardPosition pos)

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | | O | X | | 1 | O | X | X | | 2 | O | X | O | pos.getRow() = 0 pos.getColumn() = 0	State: whatsAtPos == ' ', State of board is unchanged	This function is distinct because it tests that there is a space contained at a position in the board that is not initialized with a marker. | Function Name | | testWhatsAtPosEmpty |

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | X | O | X | | 1 | O | X | X | | 2 | X | O | O | pos.getRow() = 2 post.getColumn() = 0	State: whatsAtPos == 'O', State of board is unchanged	This function is distinct because it tests that there is a marker contained at a position in the maximum row of the board. | Function Name | | testWhatsAtPosMaxRow |

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | X | O | X | | 1 | O | X | X | | 2 | X | O | O | pos.getRow() = 0 post.getColumn() = 2	State: whatsAtPos == 'X', State of board is unchanged	This function is distinct because it tests that there is a marker contained at a position in the maximum column of the board. | Function Name | | testWhatsAtPosMaxCol |

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>X</td></tr><tr><td>2</td><td>X</td><td>O</td><td>O</td></tr></table>
pos.getRow() = 1
post.getColumn() = 1</td>
<td>State:
whatsAtPos == ‘X’,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is a marker contained at a position that can be anywhere in the middle of the board.

<table><tr><td>Function Name</td></tr><tr><td>testWhatsAtPosMiddle</td></tr></table></td>
</tr>
</table>

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>X</td></tr><tr><td>2</td><td>X</td><td>O</td><td>O</td></tr></table>
pos.getRow() = 2
post.getColumn() = 2</td>
<td>State:
whatsAtPos == ‘O’,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is a marker contained at the last position entered that can be anywhere on the board and this correlates with how the keys are stored in maps.

<table><tr><td>Function Name</td></tr><tr><td>testWhatsAtPosMaps</td></tr></table></td>
</tr>
</table>

default boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>X</td></tr><tr><td>2</td><td>X</td><td>O</td><td>O</td></tr></table>
pos.getRow() = 1
post.getColumn() = 1
P = ‘X’</td>
<td>State:
isPlayerAtPos = true,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is a marker relating to the player contained in a position of the board.
<table><tr><td>Function Name</td></tr><tr><td>testIsPlayerAtPosTrue</td></tr></table></td>
</tr>
</table>

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>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>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>X</td></tr><tr><td>2</td><td>X</td><td>O</td><td>O</td></tr></table>
pos.getRow() = 2
post.getColumn() = 2
P = ‘X’</td>
<td>State:
isPlayerAtPos = false,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is not a marker relating to the player contained in a position of the board.
<table><tr><td>Function Name</td></tr><tr><td>testIsPlayerAtPosFalse</td></tr></table></td>
</tr>
</table>

<table>
<tr><th>Input</th><th>Output</th><th>Reason</th></tr>
<tr>
<td>State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td>O</td><td>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>X</td></tr><tr><td>2</td><td>X</td><td>O</td><td>O</td></tr></table>
pos.getRow() = 0</td>
<td>State:
isPlayerAtPos = false,
State of board is unchanged</td>
<td>This function is distinct because it tests that there is no marker relating to the player if there is an empty space contained in a position of the board.
<table><tr><td>Function Name</td></tr></table></td>
</tr>
</table>

post.getColumn() = 0 P = ‘X’		testIsPlayerAtPosEmptyFalse

Input	Output	Reason
State: pos.getRow() = 2 post.getColumn() = 0 P = ‘O’	State: isPlayerAtPos = true, State of board is unchanged	This function is distinct because it tests that there is a marker relating to the player contained in a position of the board that is within the maximum row of the bounds. Function Name: testIsPlayerAtPosMaxRow

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

Input	Output	Reason
State: pos.getRow() = 0 post.getColumn() = 2 P = ‘X’	State: isPlayerAtPos = true, State of board is unchanged	This function is distinct because it tests that there is a marker relating to the player contained in a position of the board that is within the maximum column of the bounds. Function Name: testIsPlayerAtPosMaxCol

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

public void placeMarker(BoardPosition marker, char player)

Input	Output	Reason
State: pos.getRow() = 2 post.getColumn() = 0 P = 'X'	State:	This function is distinct because it tests that a marker relating to the player can be placed in an empty position in the maximum row that is within the bounds of the board. Function Name: testPlaceMarkerMaxRow

	0	1	2
0			
1			
2			

	0	1	2
0			
1			
2	X		

Input	Output	Reason
State: pos.getRow() = 0 post.getColumn() = 2 P = 'X'	State:	This function is distinct because it tests that a marker relating to the player can be placed in an empty position in the maximum column that is within the bounds of the board. Function Name: testPlaceMarkerMaxCol

	0	1	2
0			
1			
2			

	0	1	2
0			X
1			
2			

Input	Output	Reason
State:	State:	This function is distinct because it tests that a marker relating to the player can be placed in an empty position in the middle of the board that is within the bounds. Function Name

	0	1	2
0			
1			
2			

	0	1	2
0			
1		X	
2			

pos.getRow() = 1 post.getColumn() = 1 P = ‘X’		testPlaceMarkerMiddle

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | | | | pos.getRow() = 1 post.getColumn() = 1 P = ‘X’ pos.getRow() = 2 post.getColumn() = 2 P = ‘O’	State: | | 0 | 1 | 2 | | 0 | | | | | 1 | | X | | | 2 | | | O |	This function is distinct because it tests that multiple markers relating to multiple players can be placed in empty positions that are within the bounds of the board. Function Name: testPlaceMarkerMultiPlayer

Input	Output	Reason
State: | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | | | | pos.getRow() = 0 post.getColumn() = 0 P = ‘A’ pos.getRow() = 1 post.getColumn() = 1 P = ‘B’ pos.getRow() = 2 post.getColumn() = 1 P = ‘C’ pos.getRow() = 0 post.getColumn() = 2	State: | | 0 | 1 | 2 | | 0 | A | F | D | | 1 | G | B | E | | 2 | H | C | I |	This function is distinct because it tests that a board be filled with as many as 25 players so in this case, 9 players can hold one position each on the smallest board size. Function Name: testPlaceMarkerFullPlayers

P = ‘D’ pos.getRow() = 1 post.getColumn() = 2 P = ‘E’ pos.getRow() = 0 post.getColumn() = 1 P = ‘F’ pos.getRow() = 1 post.getColumn() = 0 P = ‘G’ pos.getRow() = 2 post.getColumn() = 0 P = ‘H’ pos.getRow() = 2 post.getColumn() = 2 P = ‘I’		