Project 3

04/11/21

CPSC 2150

Annie Hayes

ahayes5@g.clemson.edu

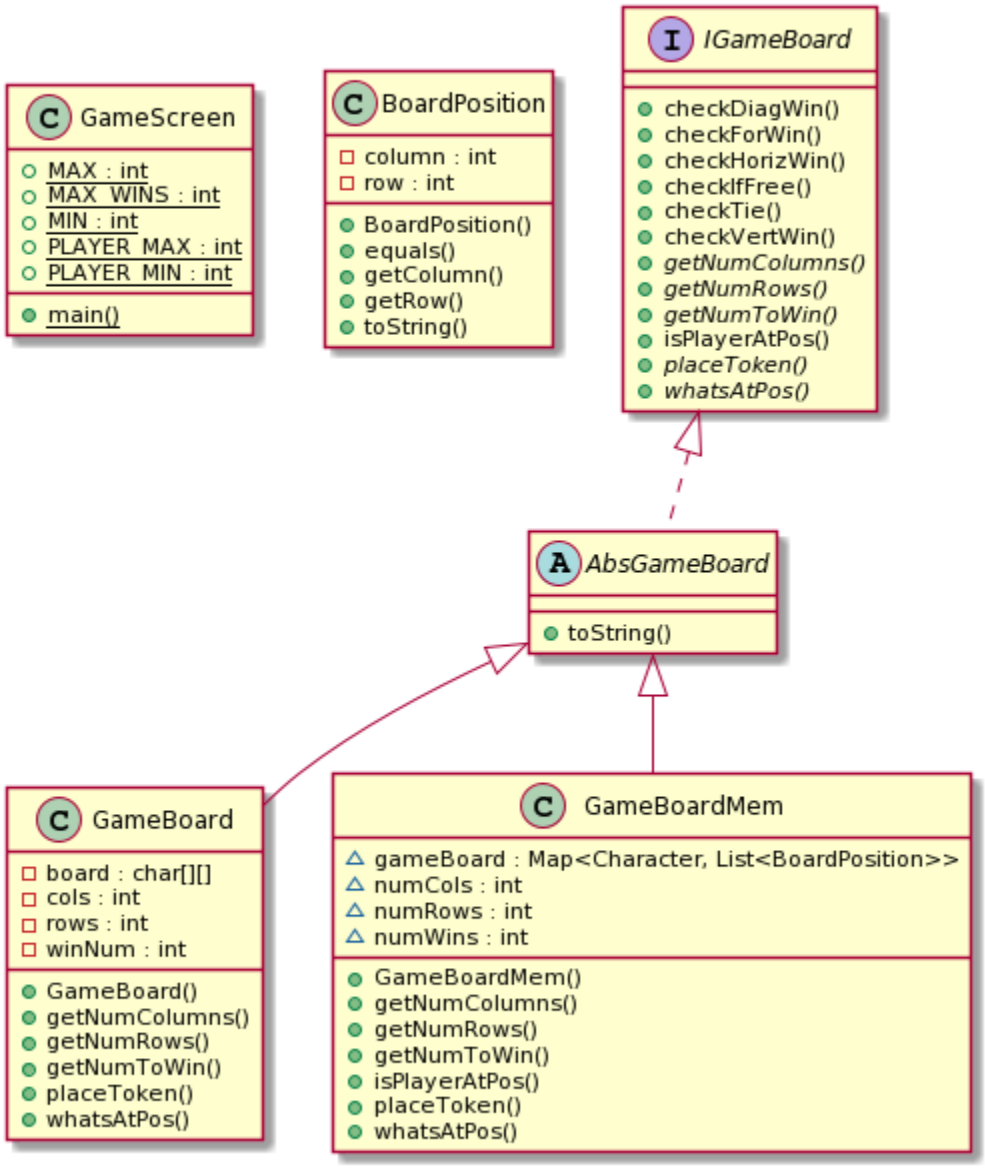
Functional Requirements

1. As a player, I can only place a marker vertically in a column to simulate the connect 4 game.
2. As a user, I can only place a marker in the maximum set number of columns so I don't lose my turn
3. As a player, I cannot place a marker in a column that is already full so the board stays the same.
4. As a user, I can decide if I want to play a fast or memory-efficient game to save memory.
5. As a user, I can win if I have the user specific number of my markers in a row horizontally.
6. As a player, I can win if I have the user specific number of my markers in a row vertically.
7. As a player, I can win if I have the user specific number of my markers in a row diagonally.
8. As a strategy player, I can have up to 10 players play in my game.
9. As a connect-4 pro, I can have up to 100 columns and rows in my game to complicate the game for the players.
10. As a strategist, I can have a tie in the game because all of the columns are full.
11. As the user, I can alternate between players so each player can have a turn.
12. As a player, I can see whose turn it is so I know who is supposed to pick a column
13. As a connect-4 pro, I can pick which column to place my marker so I know which spot I played.
14. As a player, I cannot pick a spot outside of the bounds of the board or I will get an error message.
15. As a player, I can see if I have won by looking if I have the user specific number in a row.
16. As a player, I can play again once the game has ended.
17. As a player, I can enter an integer value to say which column I have selected.
18. As a player, I cannot enter a value over the user specific number since there are only a specific number of columns.
19. As a player, player 1 will start the game so it is consistent every game.
20. As a player, the board will keep track of all of the markers so I can see which positions are filled.

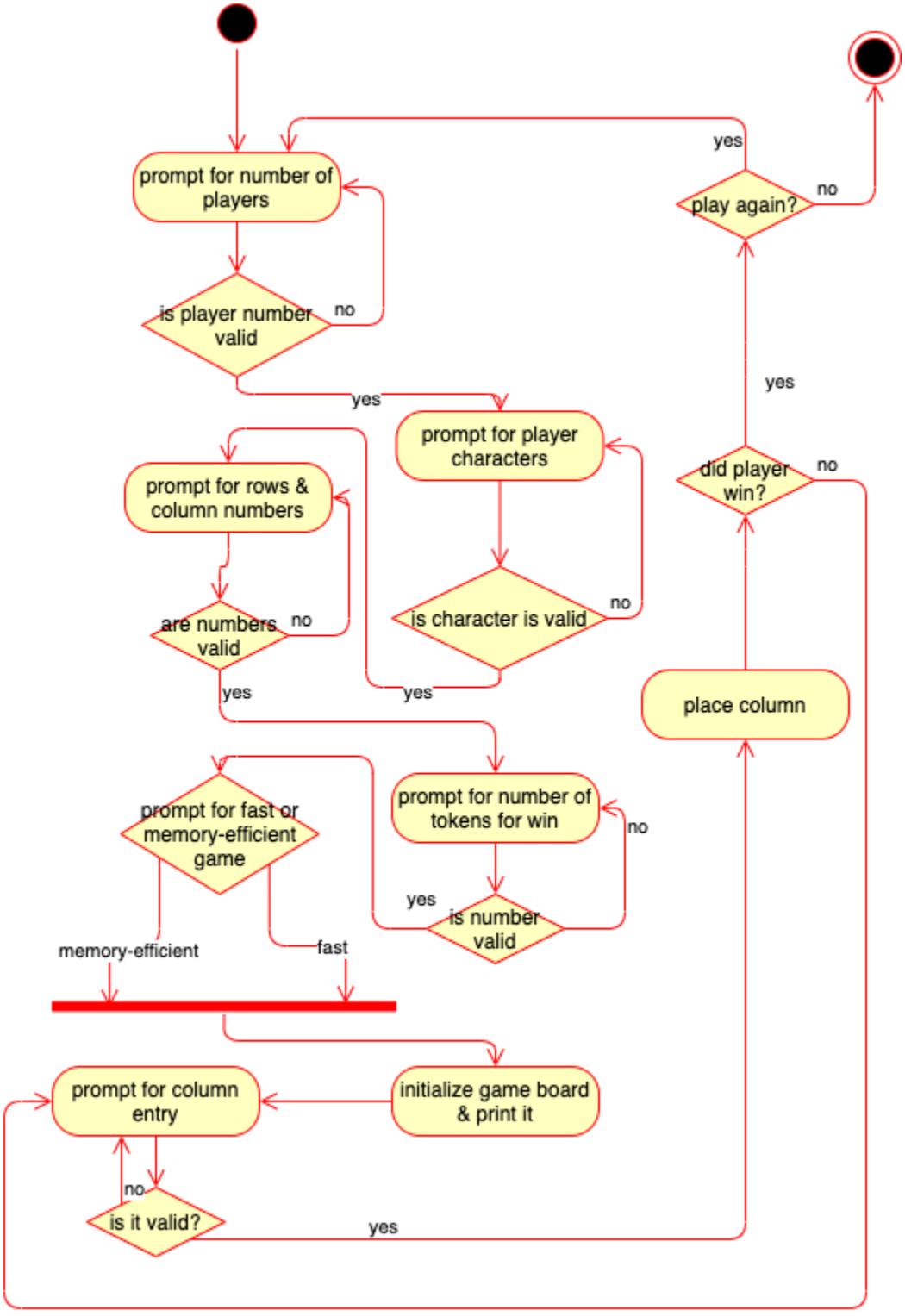
Non-Functional Requirements

1. The system must be coded in Java
2. 0,0 is the bottom left corner of the board
3. game board size cannot exceed 100 x 100
4. player 1 goes first
5. The system must run on Unix
6. Do not use magic numbers
7. use good comments
8. write contracts
9. make a program report
10. make UML class diagrams
11. make UML activity diagrams
12. write code for functions

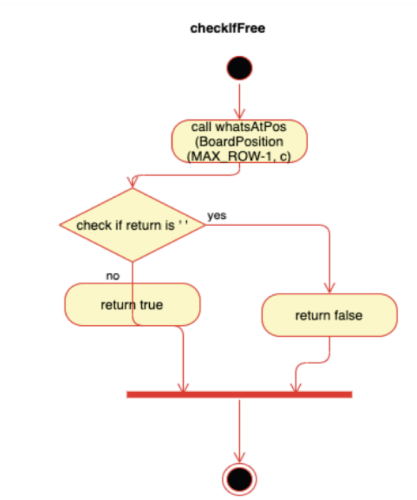
Diagrams

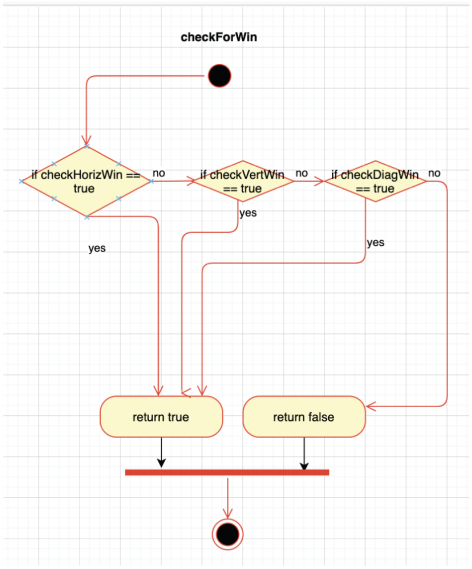

GameScreen
MAX : int
MAX_WINS : int
MIN : int
PLAYER_MAX : int
PLAYER_MIN : int
main()
BoardPosition
column : int
row : int
BoardPosition()
equals()
getColumn()
getRow()
toString()
IGameBoard
checkDiagWin()
checkForWin()
checkHorizWin()
checkIfFree()
checkTie()
checkVertWin()
getNumColumns()
getNumRows()
getNumToWin()
isPlayerAtPos()
placeToken()
whatsAtPos()
AbsGameBoard
toString()
GameBoard
board : char[][]
cols : int
rows : int
winNum : int
GameBoard()
getNumColumns()
getNumRows()
getNumToWin()
placeToken()
whatsAtPos()
GameBoardMem
gameBoard : Map<Character, List<BoardPosition>>
numCols : int
numRows : int
numWins : int
GameBoardMem()
getNumColumns()
getNumRows()
getNumToWin()
isPlayerAtPos()
placeToken()
whatsAtPos()

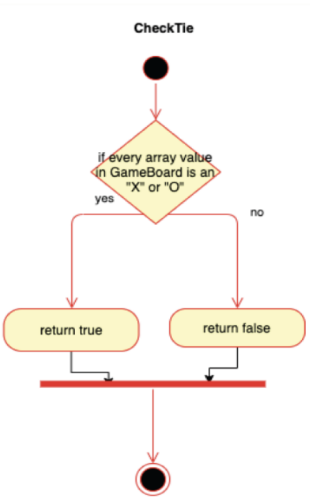
GameScreen.java

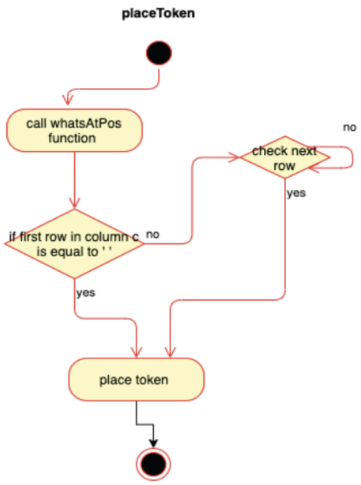


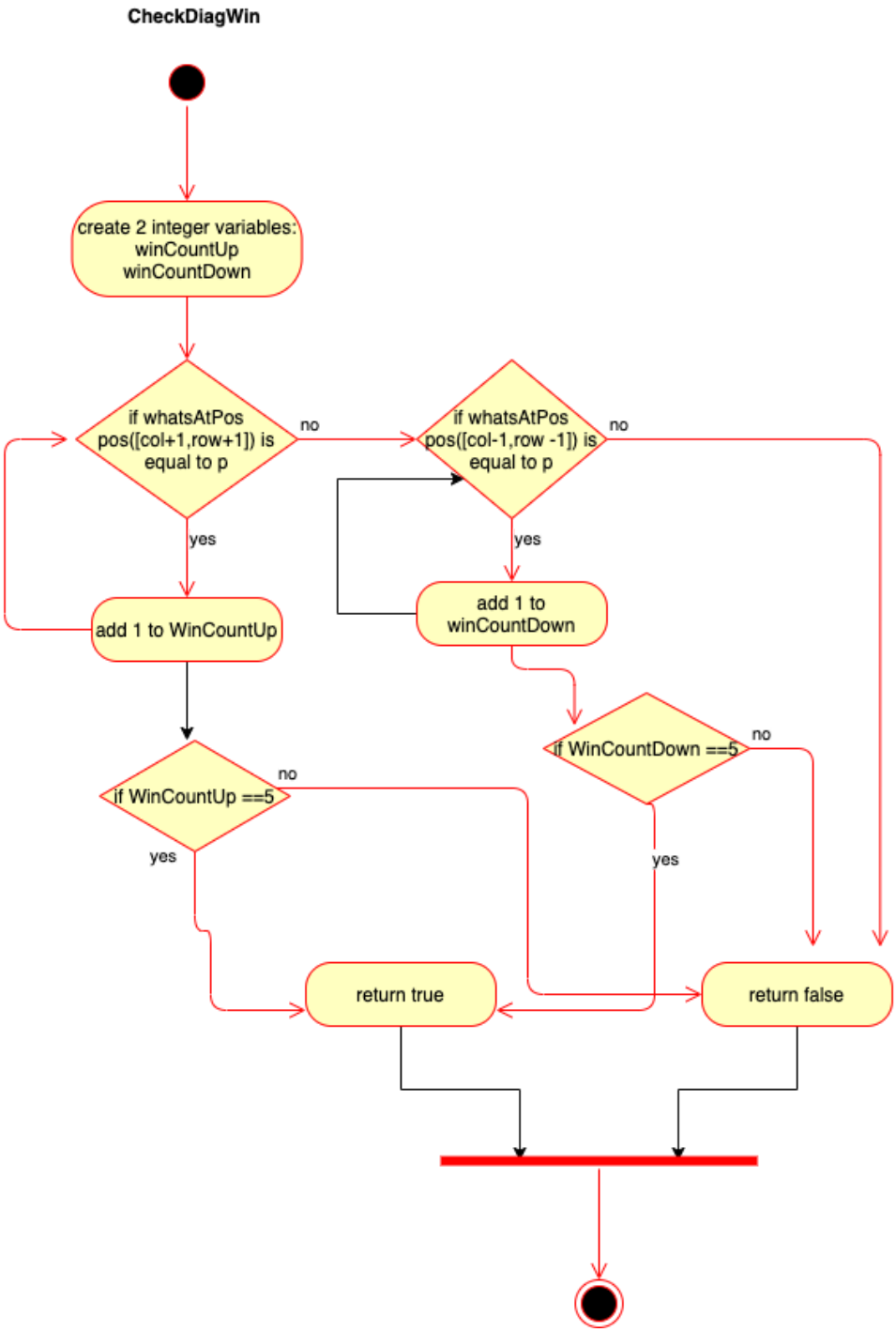
GameBoard.java

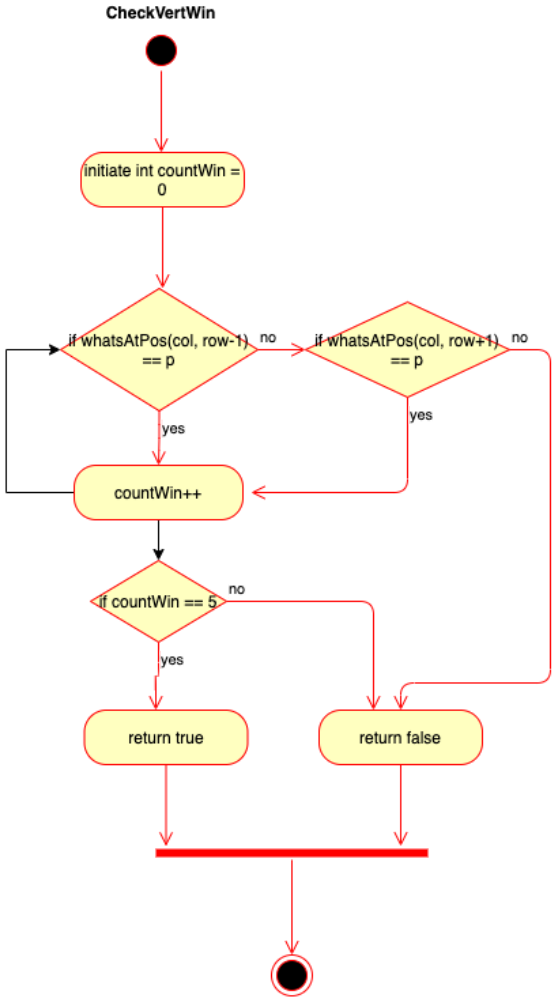


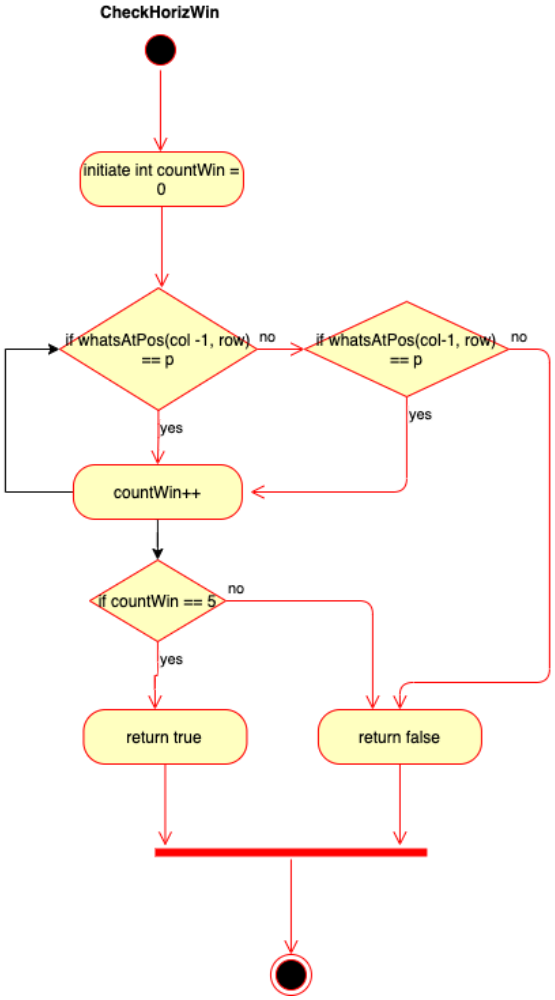


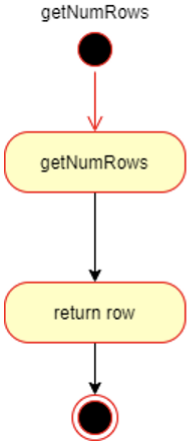


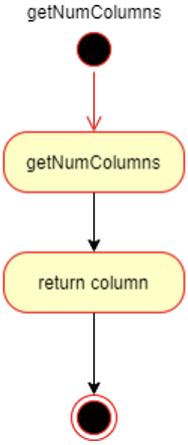


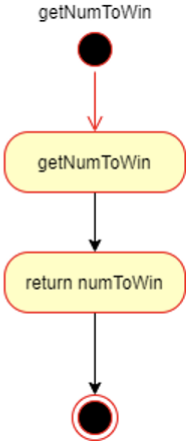

CheckDiagWin
create 2 integer variables:
winCountUp
winCountDown
if whatsAtPos
pos([col+1,row+1]) is
equal to p
no
yes
add 1 to WinCountUp
if whatsAtPos
pos([col-1,row -1]) is
equal to p
no
yes
add 1 to
winCountDown
if WinCountDown ==5
no
yes
if WinCountUp ==5
no
yes
return true
return false

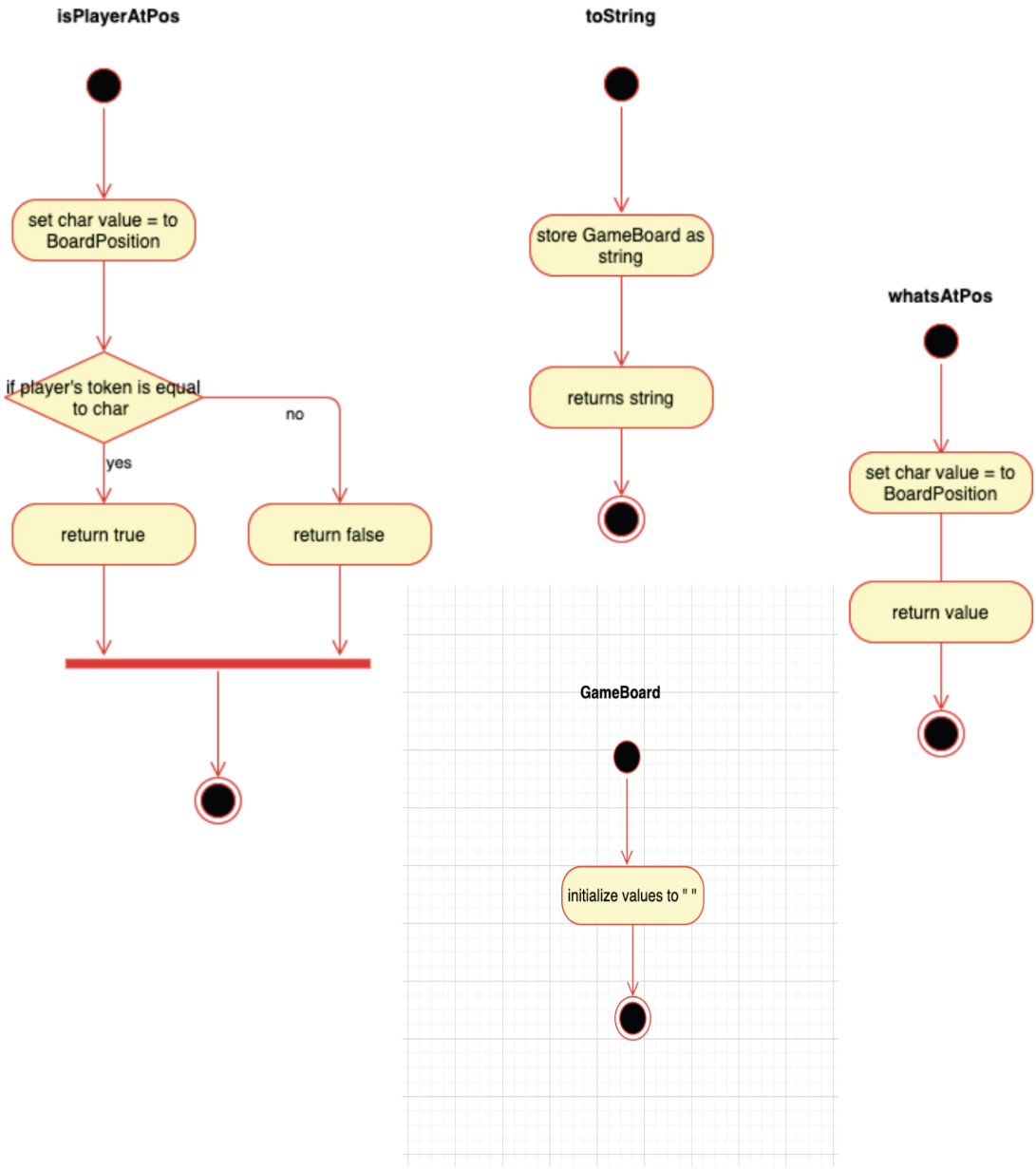

CheckVertWin
initiate int countWin = 0
if whatsAtPos(col, row-1) == p
no
if whatsAtPos(col, row+1) == p
no
yes
yes
countWin++
if countWin == 5
no
yes
return true
return false


CheckHorizWin
initiate int countWin = 0
if whatsAtPos(col -1, row) == p
no
if whatsAtPos(col-1, row) == p
no
yes
yes
countWin++
if countWin == 5
no
yes
return true
return false

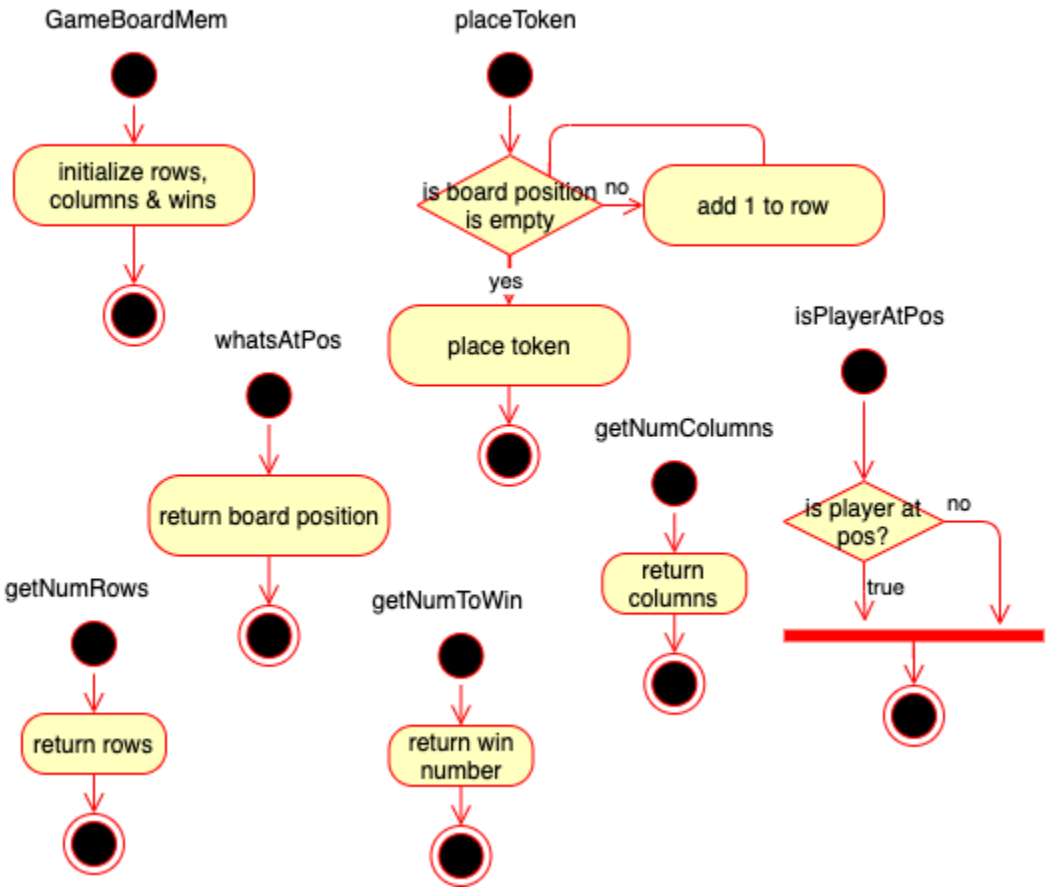

getNumRows
getNumRows
return row


getNumColumns
getNumColumns
return column


getNumToWin
getNumToWin
return numToWin


isPlayerAtPos
set char value = to BoardPosition
if player's token is equal to char
yes
no
return true
return false
toString
store GameBoard as string
returns string
whatsAtPos
set char value = to BoardPosition
return value
GameBoard
initialize values to " "

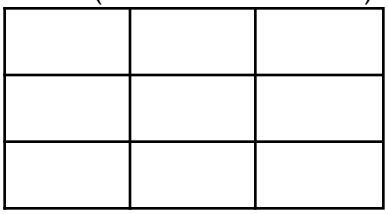
GameBoardMem.java



Test Cases

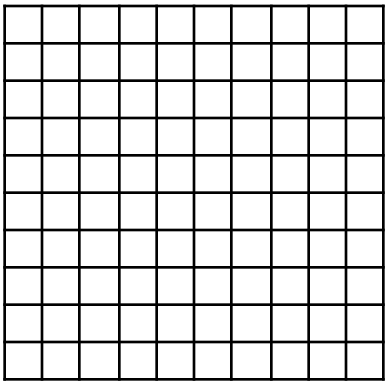
1. ConstructorTest1_Min

```
IGameBoard gb(int r, int c, int w)
```

Input:	Output:	Reasoning:
State: (number to win = 3)  r = 3 c = 3 w = 3	output is unchanged	This test case is distinct because it is testing that the minimum number of rows & columns can be initiated **Function Name:** ConstructorTest1_Min

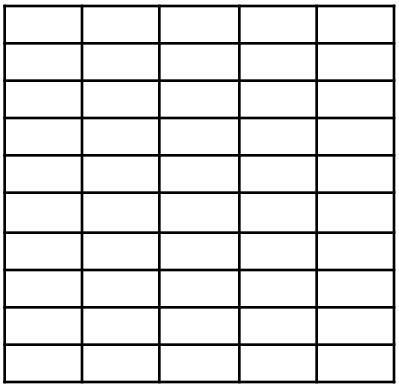
2. ConstructorTest1_Max

```
IGameBoard gb(int r, int c, int w)
```

Input:	Output:	Reasoning:
(table size is 1/10th actual size)  r = 100 c = 100 w = 25	output is unchanged	This test case is distinct because it is testing the maximum number of rows & columns that can be initiated **Function Name:** ConstructorTest1_Max

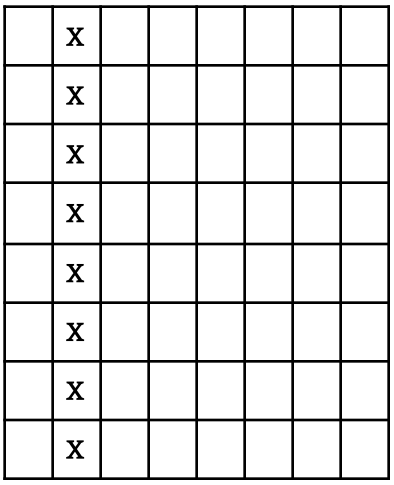
3. ConstructorTest3_Uneven

```
IGameBoard gb(int r, int c, int w)
```

Input:	Output:	Reasoning:
 r = 10 c = 5 w = 4	output is unchanged	This test case is distinct because it is testing that differing number of rows & columns can be initiated **Function Name:** ConstructorTest3_Uneven

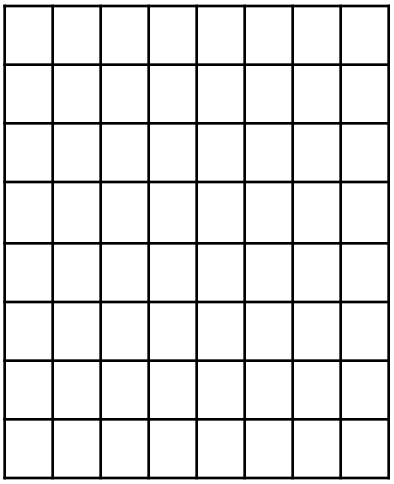
4. CheckIfFree1_Full

```
boolean checkIfFree(int c)
```

Input:	Output:	Reasoning:
 c = 1	checkIfFree = false output is unchanged	This test case is distinct & unique because it is testing that column one is full. **Function Name:** CheckIfFree1_Full

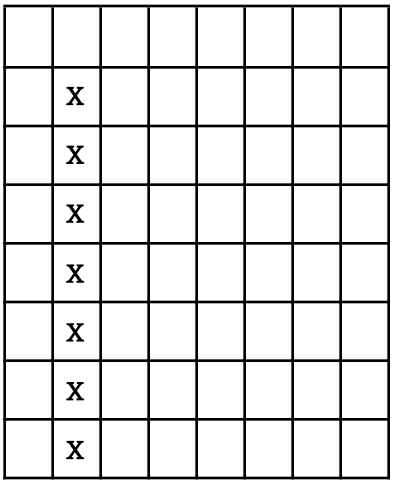
5. CheckIfFree2_Empty

```
boolean checkIfFree(int c)
```

Input:	Output:	Reasoning:
 c = 1	checkIfFree = true output is unchanged	This test case is distinct & unique because it is testing that column one is empty in an empty board. **Function Name:** CheckIfFree2_Empty

6. CheckIfFree3_OneLess

```
boolean checkIfFree(int c)
```

Input:	Output:	Reasoning:
 c = 1	checkIfFree = true output is unchanged	This test case is distinct & unique because it is testing that column one is free to accept another token in a column that is almost full. **Function Name:** CheckIfFree3_OneLess

7. CheckHorizWin1_Min

```
boolean checkHorizWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>X</td><td>X</td><td>X</td></tr></table> pos = (0, 1) p = 'X'	checkHorizWin = true output is unchanged	This test case is distinct & unique because it is testing that there is a win on the minimum win count (3). **Function Name:** CheckHorizWin1_Min

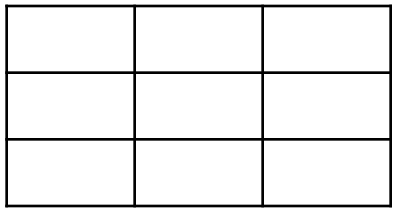
8. CheckHorizWin2_Middle

```
boolean checkHorizWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
<table><tr><td></td><td></td><td></td></tr><tr><td>O</td><td>O</td><td>O</td></tr><tr><td>X</td><td>X</td><td>X</td></tr></table> pos = (1, 1) p = 'O'	checkHorizWin = true output is unchanged	This test case is distinct & unique because there is a win starting in the middle of the board, so the function counts to the left & right. **Function Name:** CheckHorizWin2_Middle

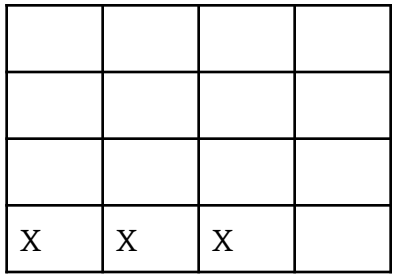
9. CheckHorizWin3_Empty

```
boolean checkHorizWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
 pos = (2, 2) p = 'X'	checkHorizWin = false output is unchanged	This test case is distinct & unique because the board is empty and there is not a horizontal win. **Function Name:** CheckHorizWin3_Empty

10. CheckHorizWin4_OneLess

```
boolean checkHorizWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
 pos = (0, 1) p = 'X'	checkHorizWin = false output is unchanged	This test case is distinct & unique because there is only 3 in a row and there is not a horizontal win. **Function Name:** CheckHorizWin4_OneLess

11. CheckVertWin1_Min

```
boolean checkVertWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
<table><tr><td>X</td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table> pos = (2, 0) p = 'X'	checkVertWin = true output is unchanged	This test case is distinct & unique because there is a 3 in a row vertical win on the minimum amount of columns and rows. **Function Name:** CheckVertWin1_Min

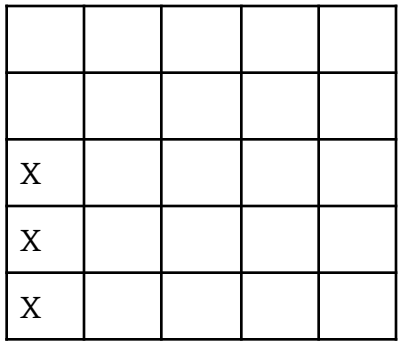
12. CheckVertWin2_TooMany

```
boolean checkVertWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
<table><tr><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td><td></td></tr></table> pos (2, 0) p = 'X'	checkVertWin = true output is unchanged	This test case is distinct & unique because the win count is 4 & there is a 4 in a row vertical win on a row that has 5 in a row vertically. **Function Name:** CheckVertWin2_TooMany

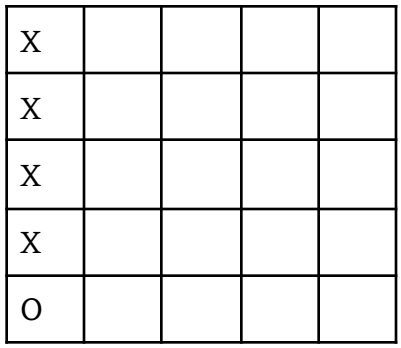
13. CheckVertWin3_OneLess

```
boolean checkVertWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
 pos (2, 0) p = 'X'	checkVertWin = false output is unchanged	This test case is unique because the win count is 4 & there is 3 in a row, so there is one less than the amount needed for a vertical win. **Function Name:** CheckVertWin3_OneLess

14. CheckVertWin4_Middle

```
boolean checkVertWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
 pos (3, 0) p = 'X'	checkVertWin = true output is unchanged	This test case is unique because the win count is 4 & there is a win at the top starting from the middle of the row. **Function Name:** CheckVertWin4_Middle

15. CheckDiagWin1_MinNorthEast

```
boolean checkDiagWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
<table><tr><td></td><td></td><td>X</td></tr><tr><td></td><td>X</td><td>O</td></tr><tr><td>X</td><td>O</td><td>O</td></tr></table> pos = (0, 0) p = 'X'	checkDiagWin = true output is unchanged	This test case is distinct because the win count is the minimum & it is starting from the bottom left of the board. **Function Name:** CheckDiagWin1_MinNorthEast

16. CheckDiagWin2_MinSouthWest

```
boolean checkDiagWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
<table><tr><td></td><td></td><td>X</td></tr><tr><td></td><td>X</td><td>O</td></tr><tr><td>X</td><td>O</td><td>O</td></tr></table> pos = (2, 2) p = 'X'	checkDiagWin = true output is unchanged	This test case is distinct because the win count is the minimum & it is starting from the top right of the board. **Function Name:** CheckDiagWin2_MinSouthWest

17. CheckDiagWin3_MinSouthEast

```
boolean checkDiagWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
<table><tr><td>X</td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td></tr><tr><td>O</td><td>O</td><td>X</td></tr></table> pos = (0, 2) p = 'X'	checkDiagWin = true output is unchanged	This test case is distinct because the win count is the minimum & it is starting from the top left of the board. **Function Name:** CheckDiagWin3_MinSouthEast

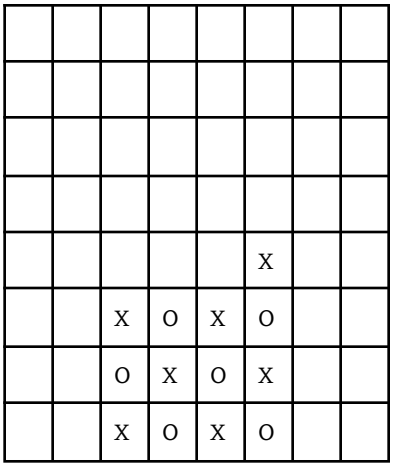
18. CheckDiagWin4_MinNorthWest

```
boolean checkDiagWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
<table><tr><td>X</td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td></tr><tr><td>O</td><td>O</td><td>X</td></tr></table> pos = (2, 0) p = 'X'	checkDiagWin = true output is unchanged	This test case is distinct because the win count is the minimum & it is starting from the bottom right of the board. **Function Name:** CheckDiagWin4_MinNorthWest

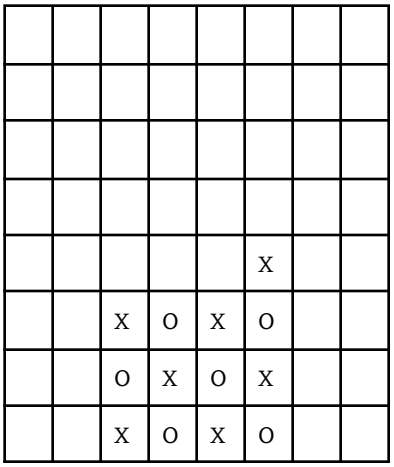
19. CheckDiagWin5_MiddleSouthWest

```
boolean checkDiagWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
 pos = (0, 2) p = 'X'	**c**heckDiagWin = true output is unchanged	This test case is distinct because the horizontal win is in the middle of the board & it is starting from the bottom left of the board. **Function Name:** CheckDiagWin5_MiddleSouthWest

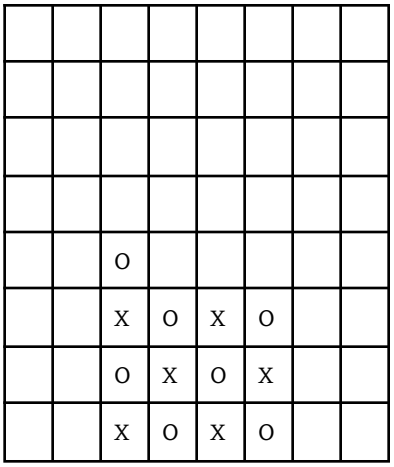
20. CheckDiagWin6_MiddleNorthEast

```
boolean checkDiagWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
 pos = (3, 5) p = 'X'	**c**heckDiagWin = true output is unchanged	This test case is distinct because the horizontal win is in the middle of the board & it is starting from the top right of the board. **Function Name:** CheckDiagWin6_MiddleNorthEast

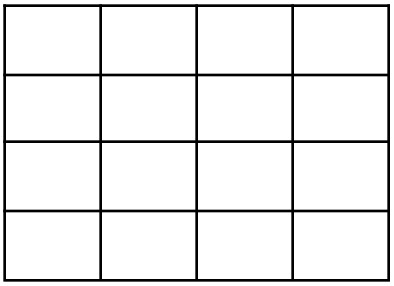
21. CheckDiagWin7_MiddleSouthEast

```
boolean checkDiagWin(BoardPosition pos, char p)
```

Input:	Output:	Reasoning:
 pos = (3, 2) p = 'X'	**c**heckDiagWin = true output is unchanged	This test case is distinct because the horizontal win is in the middle of the board & it is starting from the middle of the board. **Function Name:** CheckDiagWin7_MiddleSouthEast

22. CheckTie1_Empty

```
boolean checkTie()
```

Input:	Output:	Reasoning:
	**c**heckTie = false output is unchanged	This test case is unique because the board is blank. **Function Name:** CheckTie1_Empty

23. CheckTie2_Full

```
boolean checkTie()
```

Input:	Output:	Reasoning:
X X X O O X X O O O O X X O X X	**c**heckTie = true output is unchanged	This test case is unique because the board is full. **Function Name:** CheckTie2_Full

24. CheckTie3_AlmostFullRight

```
boolean checkTie()
```

Input:	Output:	Reasoning:
X X X O X X O O O O X X O X X	**c**heckTie = false output is unchanged	This test case is unique because the board is almost full except for the top right. **Function Name:** CheckTie3_AlmostFullRight

25. CheckTie4_AlmostFullLeft

```
boolean checkTie()
```

Input:	Output:	Reasoning:
<table><tr><td></td><td>X</td><td>X</td><td>O</td></tr><tr><td>O</td><td>X</td><td>X</td><td>O</td></tr><tr><td>O</td><td>O</td><td>O</td><td>X</td></tr><tr><td>X</td><td>O</td><td>X</td><td>X</td></tr></table>	**c**heckTie = false output is unchanged	This test case is unique because the board is almost full except for the top left. **Function Name:** CheckTie4_AlmostFullLeft

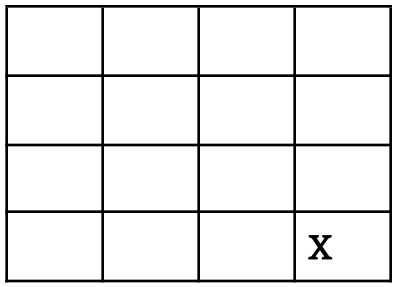
26. WhatAtPos1_SouthWest

```
char whatsAtPos(BoardPosition pos)
```

Input:	Output:	Reasoning:
<table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr></table> pos = (0, 0)	whatsAtPos = ‘X’ output is unchanged	This test case is unique because it places a token in the bottom left. **Function Name:** WhatAtPos1_SouthWest

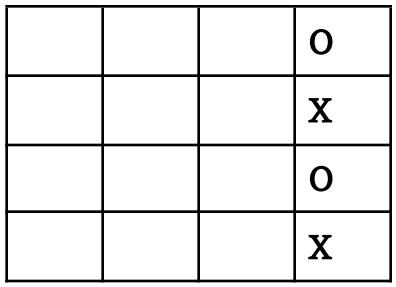
27. WhatAtPos2_SouthEast

```
char whatsAtPos(BoardPosition pos)
```

Input:	Output:	Reasoning:
 pos = (0, 3)	whatsAtPos = 'X' output is unchanged	This test case is unique because it places a token in the bottom right. **Function Name:** WhatAtPos2_SouthEast

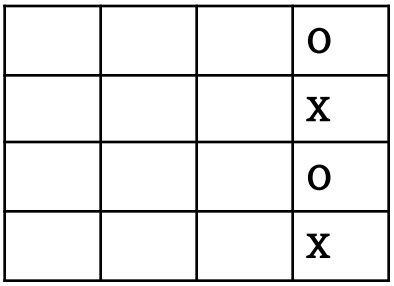
28. WhatAtPos3_Multiple

```
char whatsAtPos(BoardPosition pos)
```

Input:	Output:	Reasoning:
 pos = (3, 3)	whatsAtPos = 'O' output is unchanged	This test case is unique because it places a token in the bottom right. **Function Name:** WhatAtPos3_Multiple

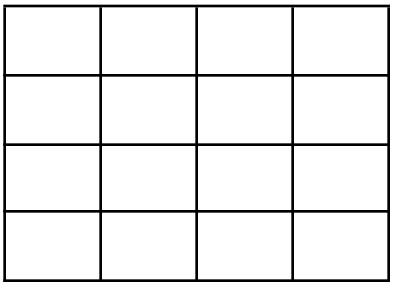
29. WhatAtPos4_WrongSymbol

```
char whatsAtPos(BoardPosition pos)
```

Input:	Output:	Reasoning:
 pos = (3, 3)	whatsAtPos = 'X' output is unchanged	This test case is unique because it places a token and checks to see if it is the other character. **Function Name:** WhatAtPos4_WrongSymbol

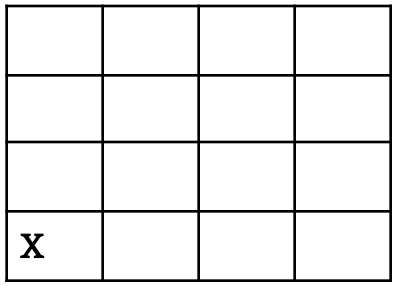
30. WhatAtPos5_Empty

```
char whatsAtPos(BoardPosition pos)
```

Input:	Output:	Reasoning:
 pos = (3, 3)	whatsAtPos = ' ' output is unchanged	This test case is unique because the board is empty. **Function Name:** WhatAtPos5_Empty

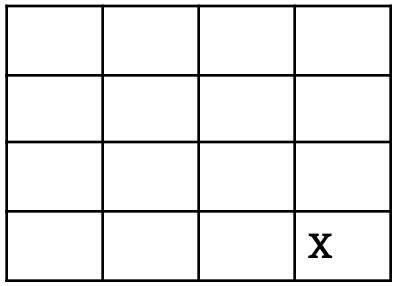
31. IsPlayerAtPos1_SouthWest

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

Input:	Output:	Reasoning:
 pos = (0, 0) player = X	isPlayerAtPos = true output is unchanged	This test case is unique because the token is placed in the bottom left. **Function Name:** IsPlayerAtPos1_SouthWest

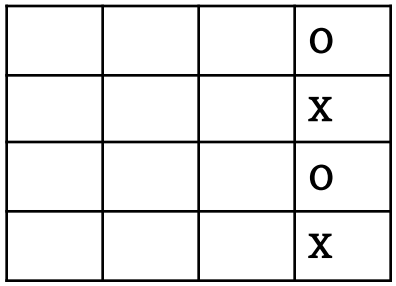
32. IsPlayerAtPos2_SouthEast

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

Input:	Output:	Reasoning:
 pos = (0, 3) player = X	isPlayerAtPos = true output is unchanged	This test case is unique because the token is placed in the bottom right. **Function Name:** IsPlayerAtPos2_SouthEast

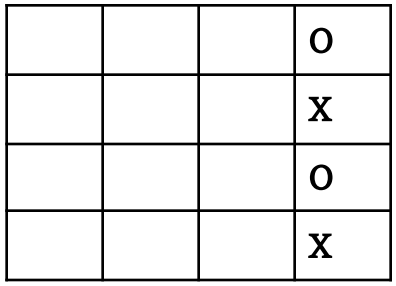
33. IsPlayerAtPos3_Multiple

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

Input:	Output:	Reasoning:
 pos = (3, 3) player = O	isPlayerAtPos = true output is unchanged	This test case is unique because there are alternating tokens and it checks for the correct symbol. **Function Name:** IsPlayerAtPos3_Multiple

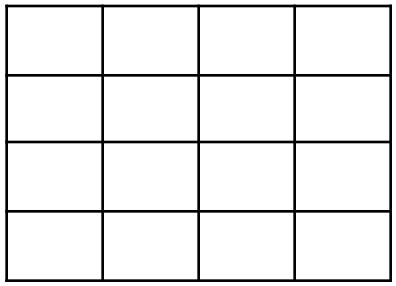
34. IsPlayerAtPos4_WrongSymbol

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

Input:	Output:	Reasoning:
 pos = (3, 3) player = O	isPlayerAtPos = false output is unchanged	This test case is unique because there are alternating tokens and it checks for the wrong symbol. **Function Name:** IsPlayerAtPos4_WrongSymbol

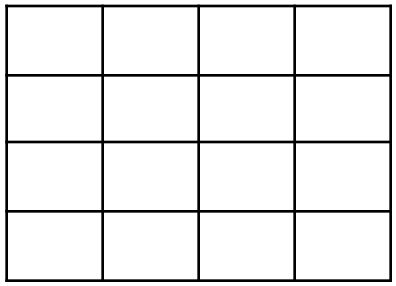
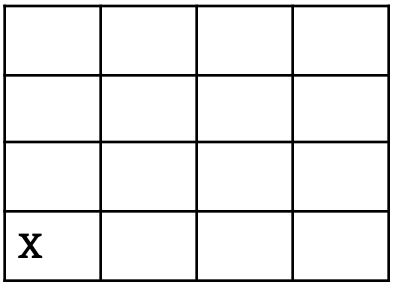
35. IsPlayerAtPos5_Empty

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

Input:	Output:	Reasoning:
 pos = (3, 3) player = O	isPlayerAtPos = false output is unchanged	This test case is unique because the board is blank and it checks for the wrong symbol. **Function Name:** IsPlayerAtPos5_Empty

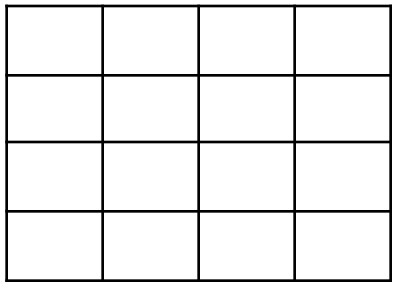
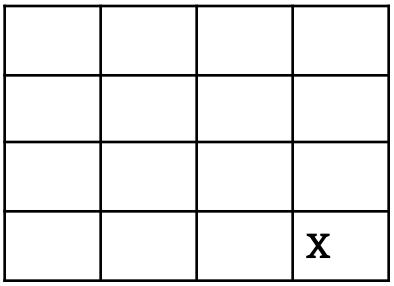
36. PlaceToken1_SouthWest

```
void placeToken(char p, int c)
```

Input:	Output:	Reasoning:
 p = X c = 0	 **X**	This test case is distinct because the board is blank and it places X in the bottom left boundary. **Function Name:** PlaceToken1_SouthWest

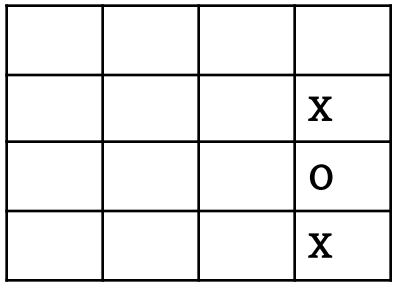
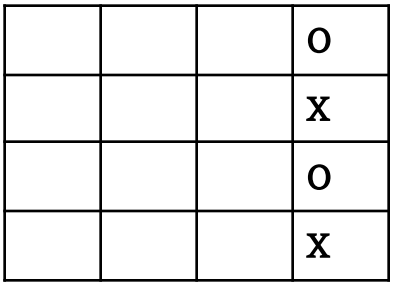
37. PlaceToken2_SouthEast

```
void placeToken(char p, int c)
```

Input:	Output:	Reasoning:
 p = X c = 3		This test case is distinct because the board is blank and it places X in the bottom right boundary. **Function Name:** PlaceToken2_SouthEast

38. PlaceToken3_NorthEast

```
void placeToken(char p, int c)
```

Input:	Output:	Reasoning:
 p = O c = 3		This test case is distinct because the last column is almost full & it places X in the top right boundary. **Function Name:** PlaceToken3_NorthEast

39. PlaceToken4_NorthWest

```
void placeToken(char p, int c)
```

Input:	Output:	Reasoning:
<table><tr><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr></table> p = O c = 0	<table><tr><td>O</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr></table>	This test case is distinct because the last column is almost full & it places X in the top left boundary. **Function Name:** PlaceToken4_NorthWest

40. PlaceToken5_Middle

```
void placeToken(char p, int c)
```

Input:	Output:	Reasoning:
<table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td>O</td><td>O</td><td></td></tr><tr><td></td><td>X</td><td>X</td><td></td></tr></table> p = X c = 2	<table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td>X</td><td></td></tr><tr><td></td><td>O</td><td>O</td><td></td></tr><tr><td></td><td>X</td><td>X</td><td></td></tr></table>	This test case is distinct because the last column is almost full & it places X in the middle. **Function Name:** PlaceToken5_Middle