CPSC 2150 Project Report

Bennett Starks

Requirements Analysis

Functional Requirements:

1. As a player, I can view the game board so that I can know the current board state.
2. As a player, I can see where distinct player tokens are placed so I can distinguish between one player and another.
3. As a player, I can alternate turns after a piece is placed to play the game.
4. As a player, I can select a column so that I can place a piece.
5. As a player, I need my piece to be inserted onto the lowest empty row on the grid so that my piece is placed correctly.
6. As a player, I can view the column numbers above each column so that I can clearly know which column I am placing my piece in.
7. As a player, I need to be prompted to choose a different column if my placement is invalid so that I cannot lose my turn if the chosen column is full.
8. As a player, I can choose a different column to place a piece so that I cannot lose my turn if the chosen column is out of the possible selection range.
9. As a player, I need my last piece location to be checked for a win for 5 in a row so that I can win the game.
10. As a player, I need a piece placed 5 in a row horizontally to end the game so that I can win.
11. As a player, I need a piece placed 5 in a row vertically to end the game so that I can win.
12. As a player, I need a piece placed 5 in a row diagonally in both directions to end the game so that I can win.
13. As a player, I can tie by filling the board with pieces and no win being found.
14. As a player, I need to not be able to take any more turns and have the victory displayed if the other player wins so that the game properly ends on a victory.
15. As a player, I need to not be able to take anymore turns and have the tie displayed if the game results in a tie so that I will not be stuck being unable to select a column, being trapped in an infinite loop.
16. As a player, I need to be able to take my turn if the other player does NOT win so that the game properly continues until a victory or a tie.
17. As a player, I can see which player won or if we tied so that I know the outcome of the game.
18. As a player, I need a quit prompt when the game is over so that I can exit the game/program.

19. As a player, I need to be asked if I want to play again when a game is over so that I can choose to play again.
20. As a player, I can choose which character to use as a token so that I can know where I have placed my token on the board distinctly from other players.
21. As a player, I can choose whether or not to have a fast or memory efficient implementation so that I can control the speed and memory efficiency of the gameboard I am using.
22. As a player, I can choose how many players will be in the game so that I can have more than just two players if I want.
23. As a player, I can change the size of the board and number of tokens to win so that I can play on different sized game boards.
24. As a player, I can redeclare the size of the board and number of tokens to win when I choose to play again so that I can change the rules if I would like to.
25. As a player, I can change the board implementation when I choose to play again so that I can change the implementation if I would like to.
26. As a player, I can change the number of players when I choose to play again so that I can change the number of active players if I would like to.
27. As a player, I can change the characters representing the players when I choose to play again so that I can select different token appearances if I would like to.
28. As a player, I can re-input erroneous values when making decisions about board size, number of tokens to win, how many players are in the game, and what implementation to use so that I do not select an invalid value.

Non-Functional Requirements

1. The program must run in a Java environment.
2. The program must be a unix program and use the command line.
3. The program must be coded in Java.
4. The program must have clear output for the user to respond to.
5. The program must not crash if invalid columns are selected.
6. The board must have row size at minimum 3 and maximum 100
7. The board must have column size at minimum 3 and maximum 100
8. The board must have a number of tokens to win at minimum 3 and maximum 25.
9. The board must have a number of tokens to win that does not exceed the row and column size.
10. The bottom left tile of the board is at (0 , 0).
11. The program must be programmed using Java Swing

12. The program must produce a window to set the game settings.
13. The program must produce a window to play the game.

Deployment Instructions

For running the main program

- To compile the main program, use the command: make
- To run the main program, use the command: make run
- To clean the files created by compilation, use the command: make clean

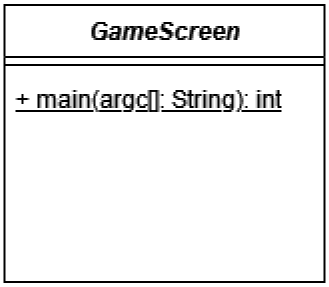
For running the unit tests

- To compile the test cases, use the command: test
- To test the fast implementation of the gameboard, use the command: make testGB
- To test the memory efficient implementation of the gameboard, use the command: make testGBmem
- To clean the files created by compilation, use the command: make clean

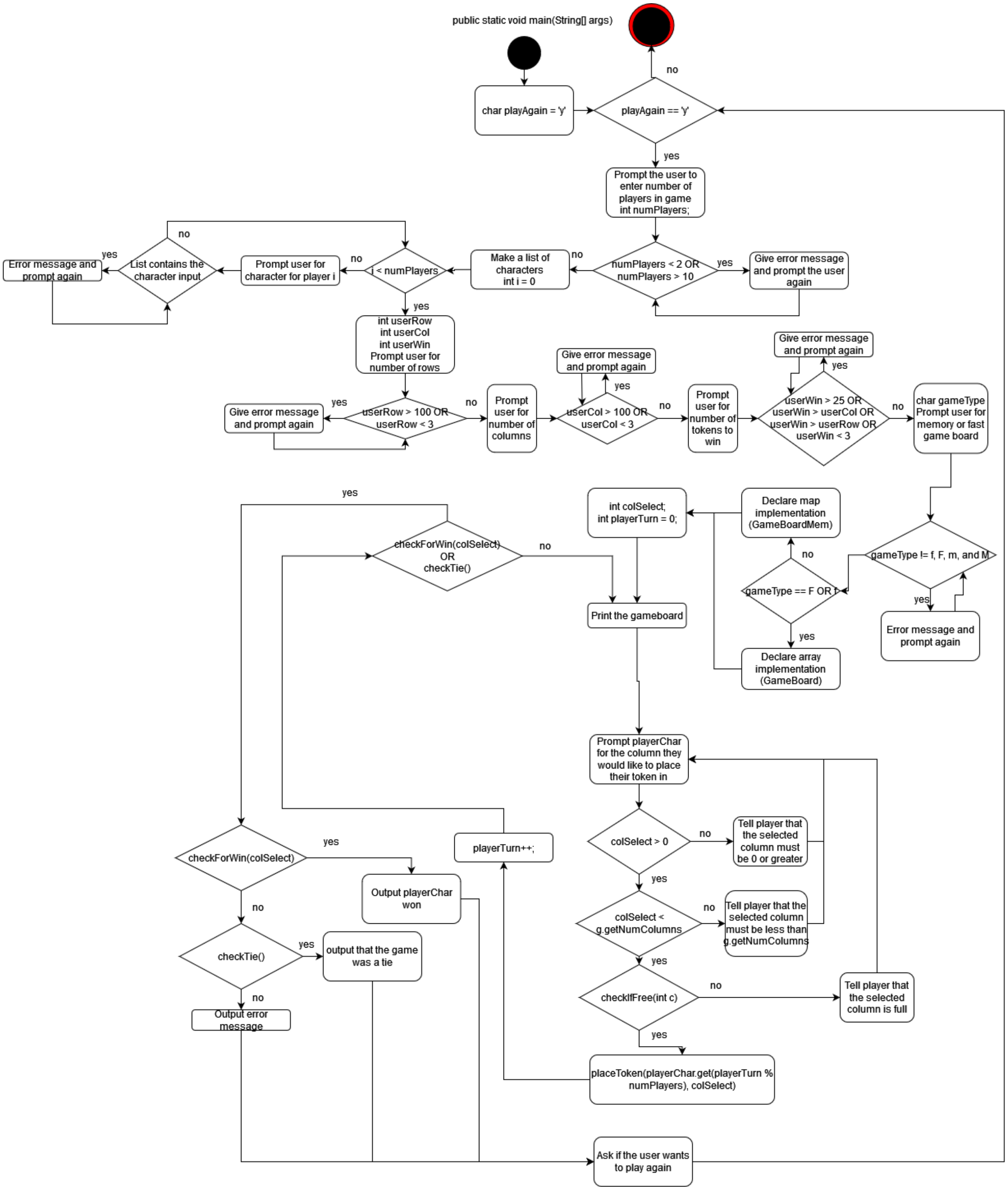
System Design

Class 1: GameScreen

Class diagram

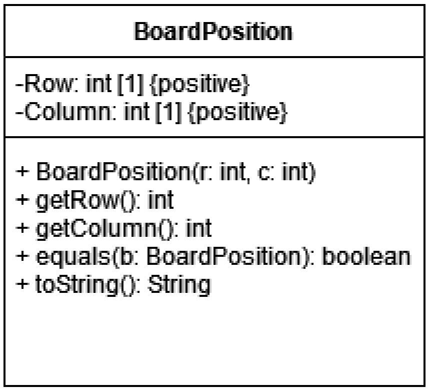


Activity Diagrams

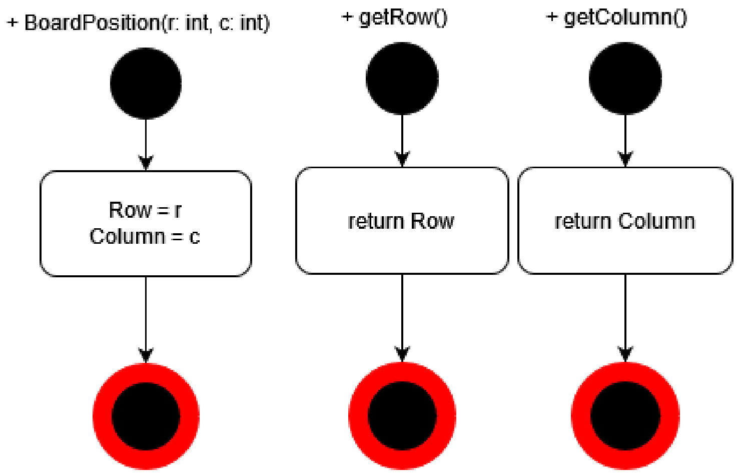

public static void main(String[] args)
char playAgain = 'y'
playAgain == 'y'
no
yes
Prompt the user to enter number of players in game int numPlayers;
numPlayers < 2 OR numPlayers > 10
yes
Give error message and prompt the user again
no
Make a list of characters int i = 0
i < numPlayers
no
Prompt user for character for player i
List contains the character input
yes
Error message and prompt again
no
yes
int userRow int userCol int userWin Prompt user for number of rows
userRow > 100 OR userRow < 3
yes
Give error message and prompt again
no
Prompt user for number of columns
userCol > 100 OR userCol < 3
yes
Give error message and prompt again
no
Prompt user for number of tokens to win
userWin > 25 OR userWin > userCol OR userWin > userRow OR userWin < 3
yes
Give error message and prompt again
no
char gameType Prompt user for memory or fast game board
gameType != f, F, m, and M
yes
Error message and prompt again
gameType == F OR f
no
Declare map implementation (GameBoardMem)
yes
Declare array implementation (GameBoard)
int colSelect; int playerTurn = 0;
Print the gameboard
checkForWin(colSelect) OR checkTie()
no
yes
Prompt playerChar for the column they would like to place their token in
colSelect > 0
no
Tell player that the selected column must be 0 or greater
yes
colSelect < g.getNumColumns
no
Tell player that the selected column must be less than g.getNumColumns
yes
checkIfFree(int c)
no
Tell player that the selected column is full
yes
placeToken(playerChar.get(playerTurn % numPlayers), colSelect)
playerTurn++;
checkForWin(colSelect)
yes
Output playerChar won
no
checkTie()
yes
output that the game was a tie
no
Output error message
Ask if the user wants to play again

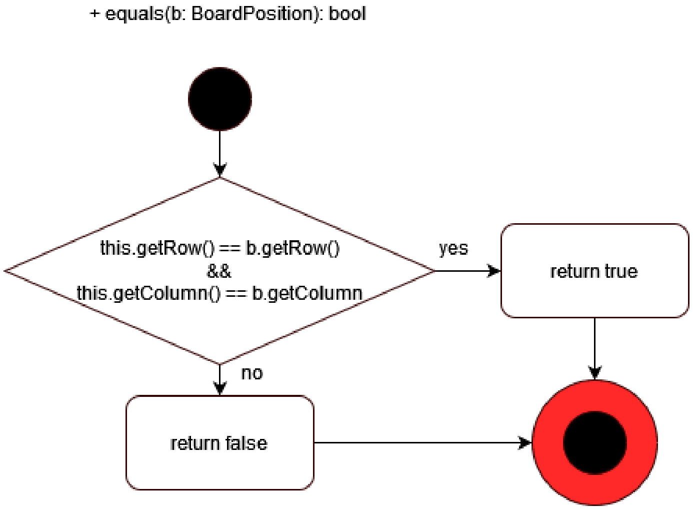
Class 2: BoardPosition

Class diagram


BoardPosition
-Row: int [1] {positive}
-Column: int [1] {positive}
+ BoardPosition(r: int, c: int)
+ getRow(): int
+ getColumn(): int
+ equals(b: BoardPosition): boolean
+ toString(): String

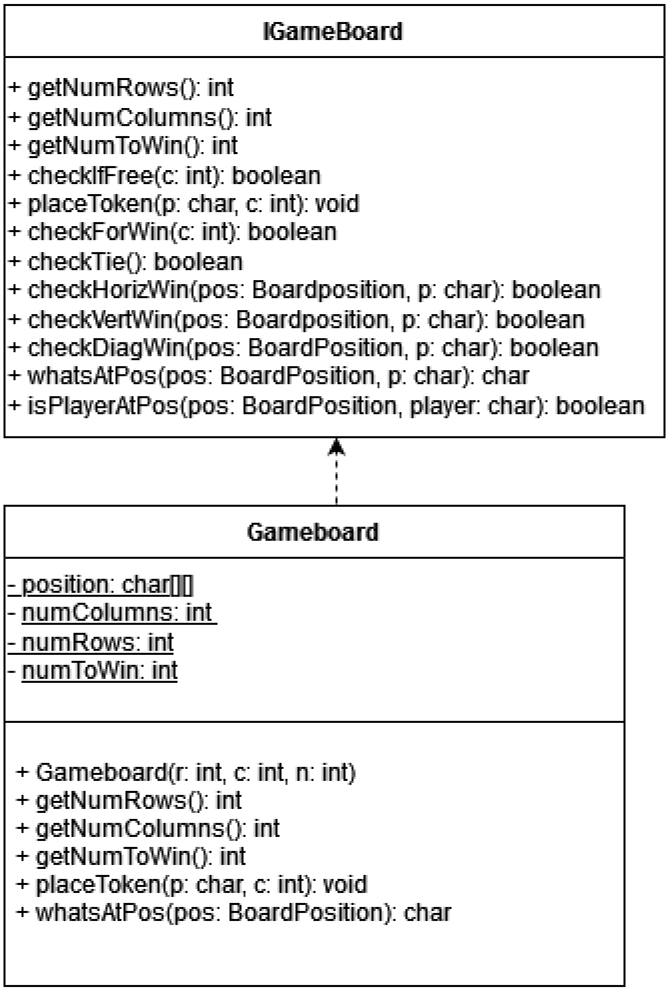
Activity diagrams




+ equals(b: BoardPosition): bool
this.getRow() == b.getRow()
&&
this.getColumn() == b.getColumn
yes
return true
no
return false

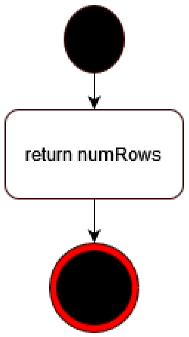
Class 3: Gameboard

Class diagram

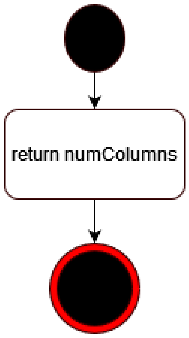

IGameBoard
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ checkIfFree(c: int): boolean
+ placeToken(p: char, c: int): void
+ checkForWin(c: int): boolean
+ checkTie(): boolean
+ checkHorizWin(pos: Boardposition, p: char): boolean
+ checkVertWin(pos: Boardposition, p: char): boolean
+ checkDiagWin(pos: BoardPosition, p: char): boolean
+ whatsAtPos(pos: BoardPosition, p: char): char
+ isPlayerAtPos(pos: BoardPosition, player: char): boolean
Gameboard
- position: char[][]
- numColumns: int
- numRows: int
- numToWin: int
+ Gameboard(r: int, c: int, n: int)
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ placeToken(p: char, c: int): void
+ whatsAtPos(pos: BoardPosition): char

Activity diagrams

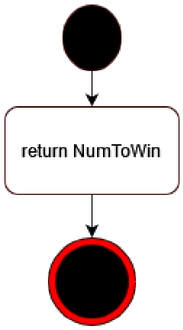
+ getNumRows(): int

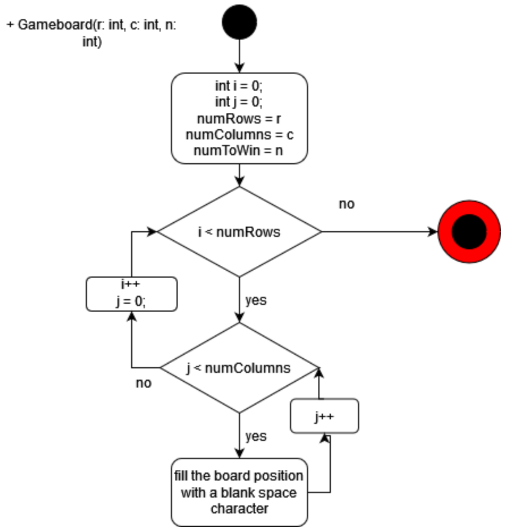


+ getNumColumns(): int

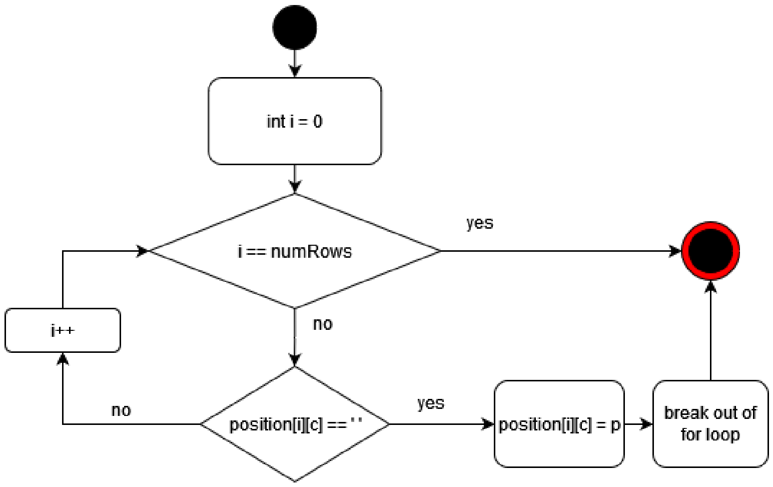


+ getNumToWin(): int

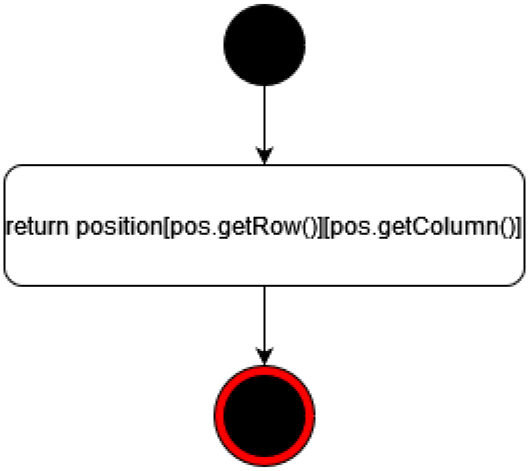




void placeToken(char p, int c)



char whatsAtPos(Boardposition pos)

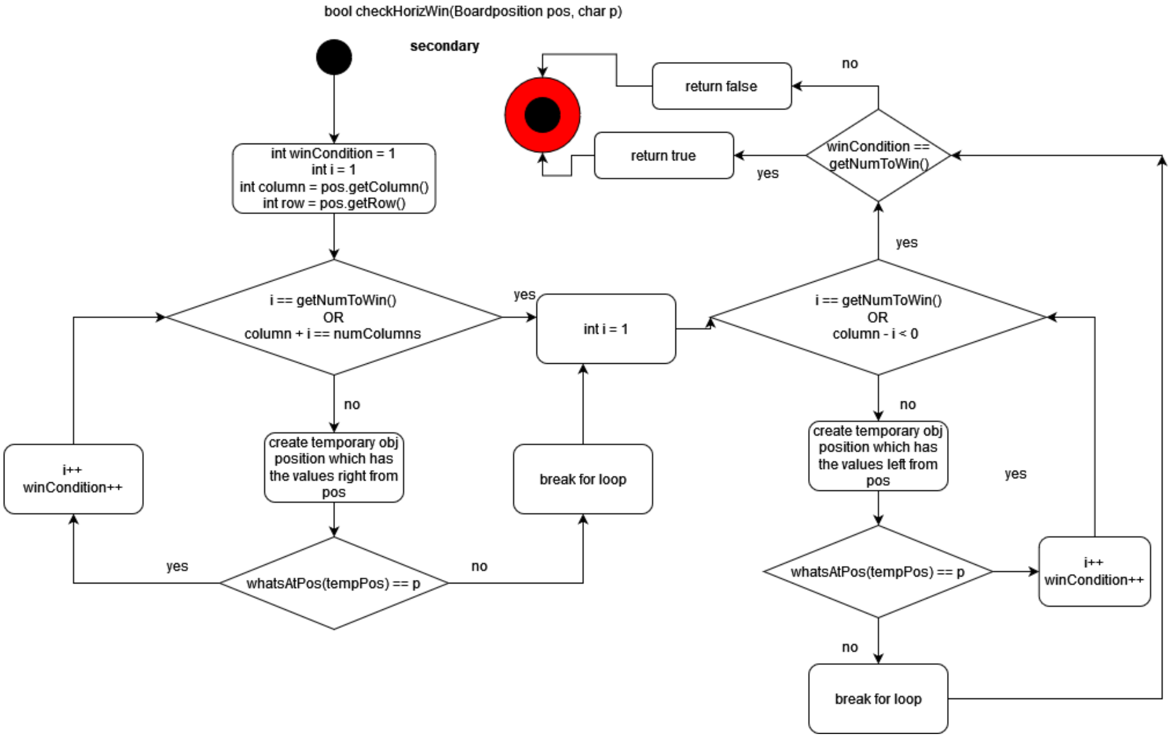


Class 4: IGameboard

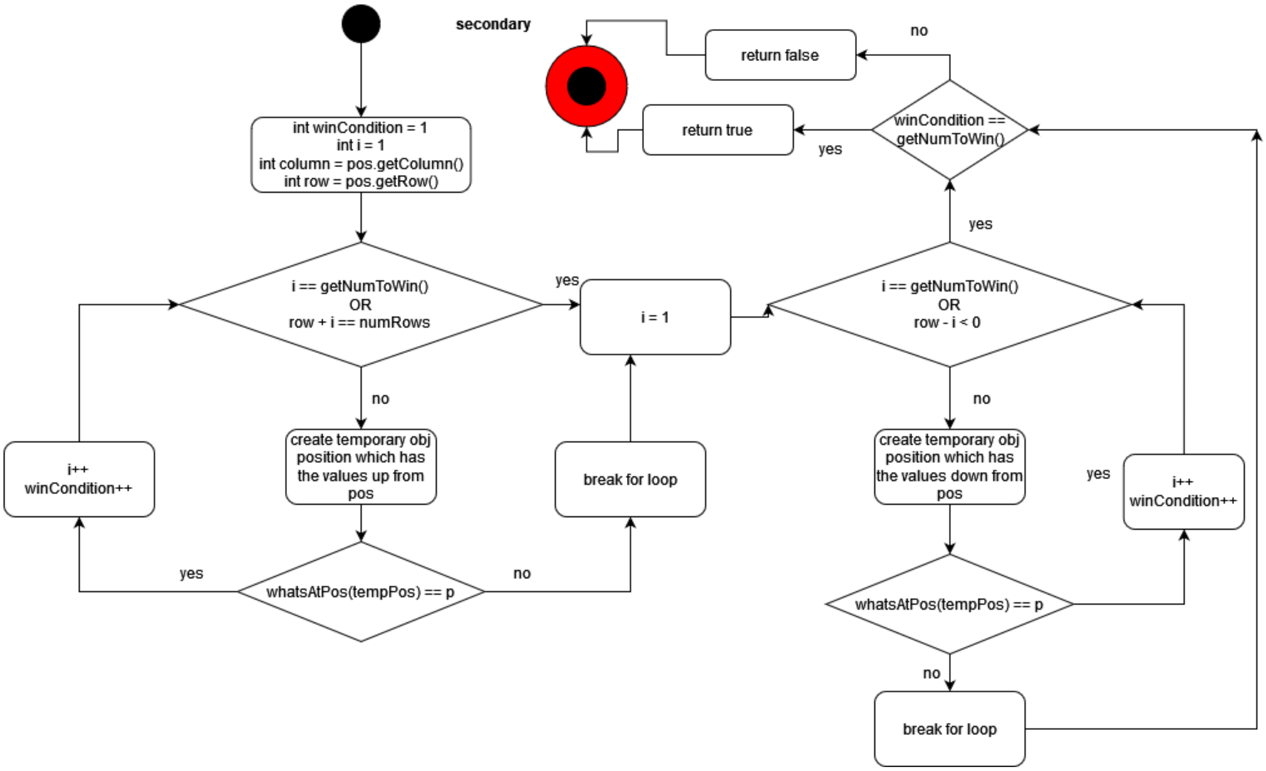
Class diagram

IGameBoard
+ getNumRows(): int + getNumColumns(): int + getNumToWin(): int + checkIfFree(c: int): boolean + placeToken(p: char, c: int): void + checkForWin(c: int): boolean + checkTie(): boolean + checkHorizWin(pos: Boardposition, p: char): boolean + checkVertWin(pos: Boardposition, p: char): boolean + checkDiagWin(pos: BoardPosition, p: char): boolean + whatsAtPos(pos: BoardPosition, p: char): char + isPlayerAtPos(pos: BoardPosition, player: char): boolean

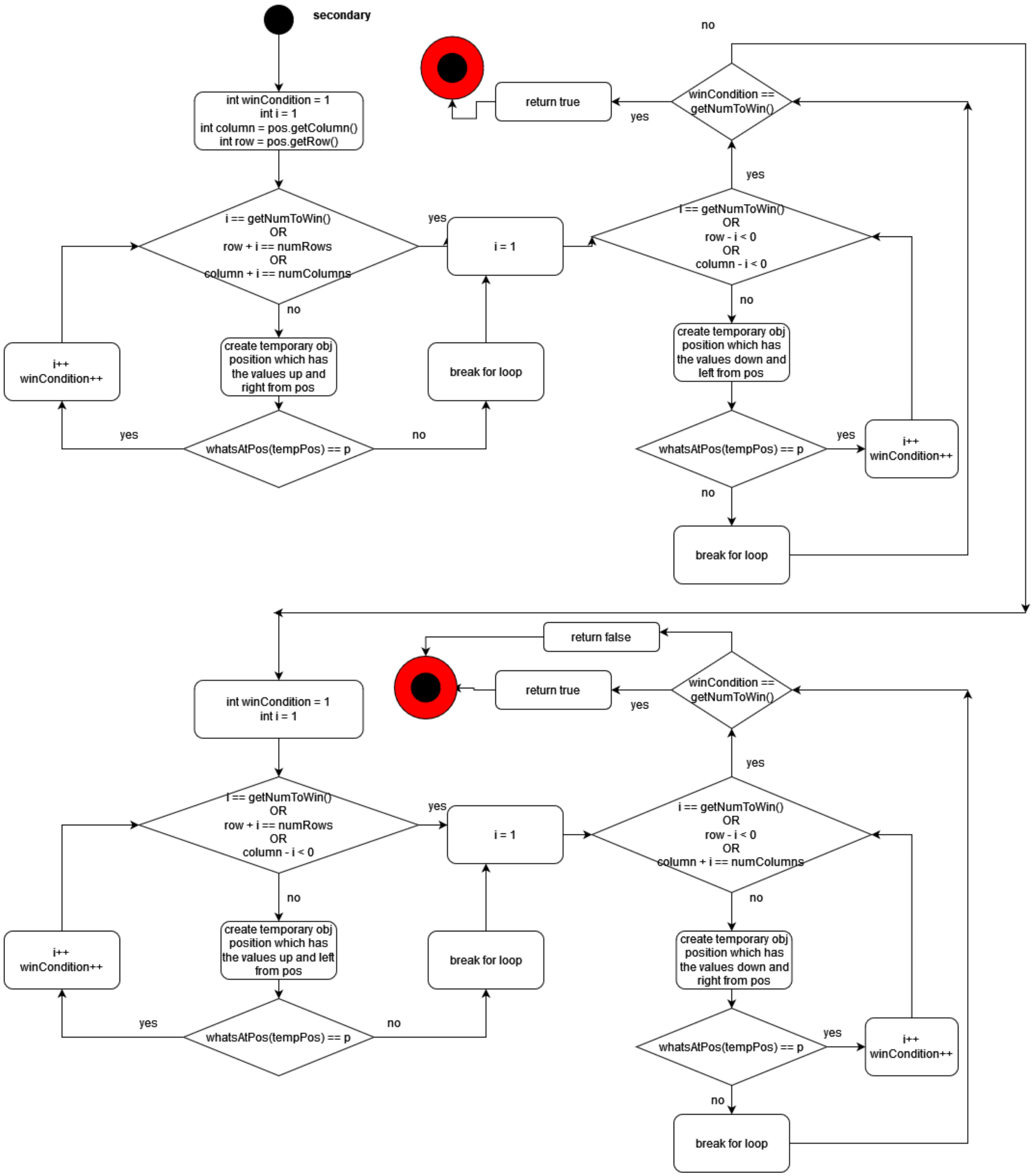
Activity diagram



bool checkVertWin(Boardposition pos, char p)

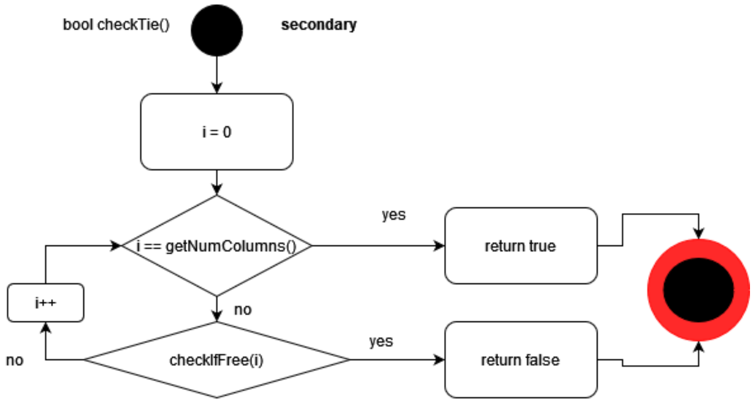


bool checkDiagWin(Boardposition pos, char p)



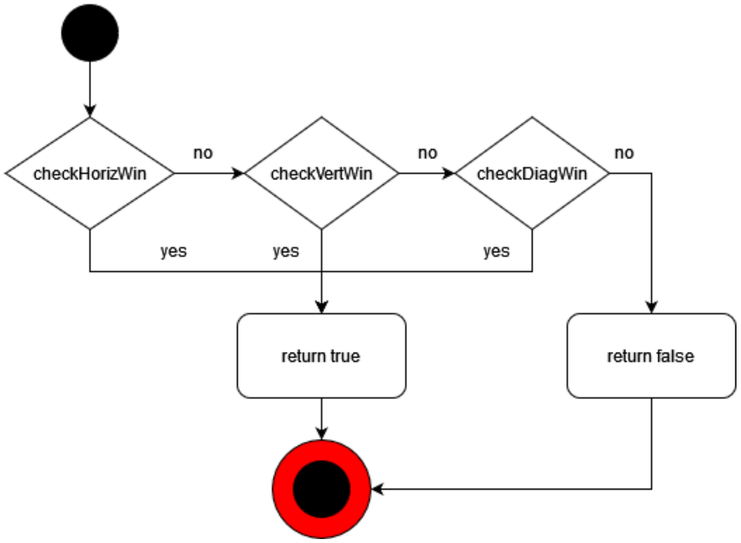
bool checkTie()

secondary



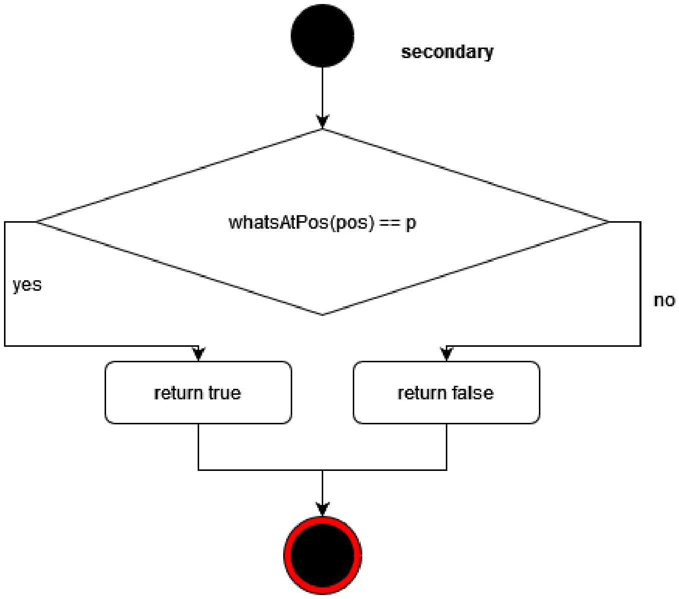
bool checkForWin(int c);

secondary

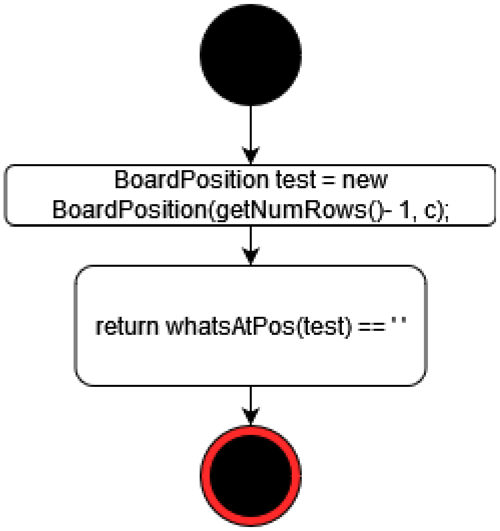


bool isPlayerAtPos(Boardposition pos, char p)

secondary

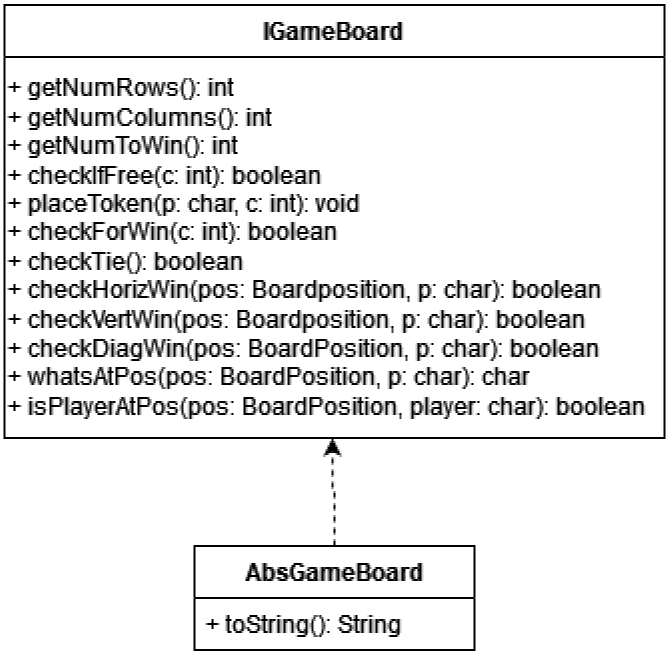


+ checkIfFree(c: int): boolean

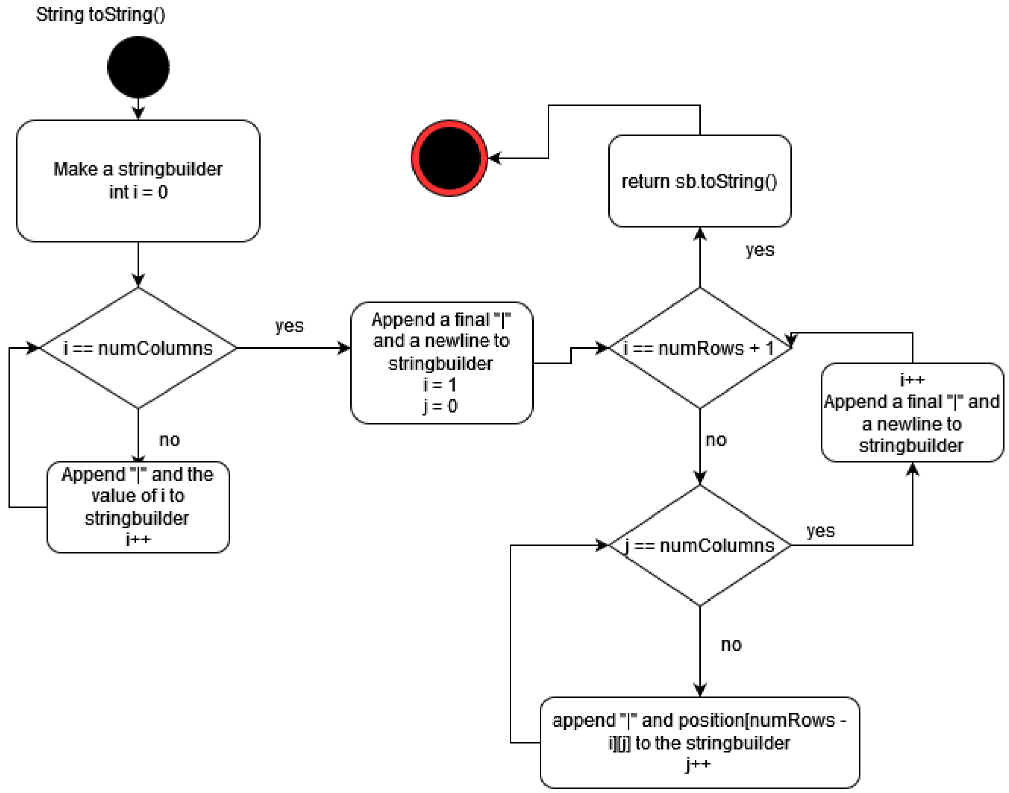

BoardPosition test = new BoardPosition(getNumRows()- 1, c);
return whatsAtPos(test) == ' '

Class 5: AbsGameBoard

Class diagram

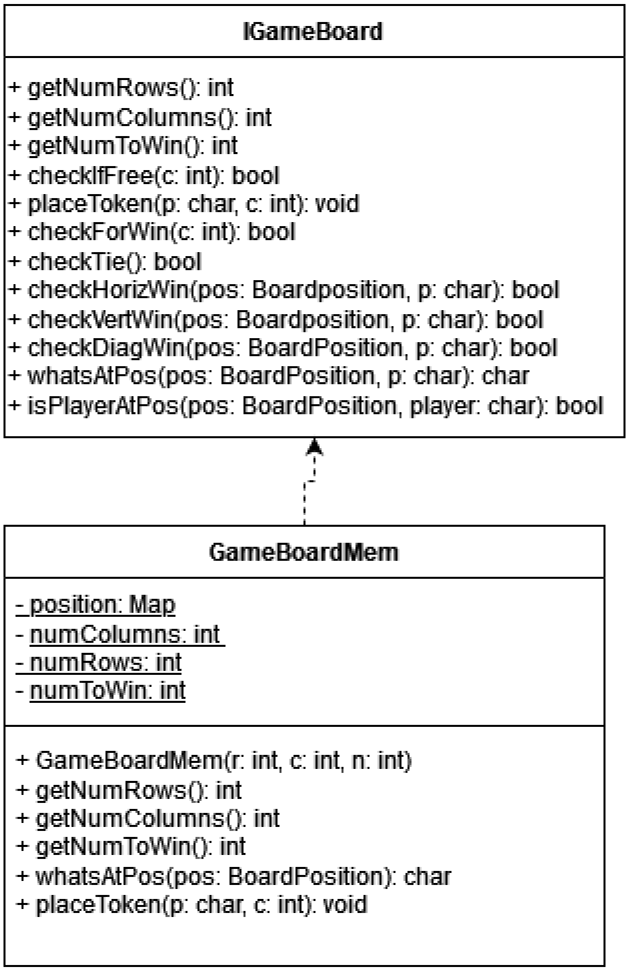

IGameBoard
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ checkIfFree(c: int): boolean
+ placeToken(p: char, c: int): void
+ checkForWin(c: int): boolean
+ checkTie(): boolean
+ checkHorizWin(pos: Boardposition, p: char): boolean
+ checkVertWin(pos: Boardposition, p: char): boolean
+ checkDiagWin(pos: BoardPosition, p: char): boolean
+ whatsAtPos(pos: BoardPosition, p: char): char
+ isPlayerAtPos(pos: BoardPosition, player: char): boolean
AbsGameBoard
+ toString(): String

Activity diagram


String toString()
Make a stringbuilder
int i = 0
i == numColumns
yes
no
Append "|" and the value of i to stringbuilder
i++
Append a final "|" and a newline to stringbuilder
i = 1
j = 0
i == numRows + 1
yes
no
return sb.toString()
i++
Append a final "|" and a newline to stringbuilder
j == numColumns
yes
no
append "|" and position[numRows - i][j] to the stringbuilder
j++

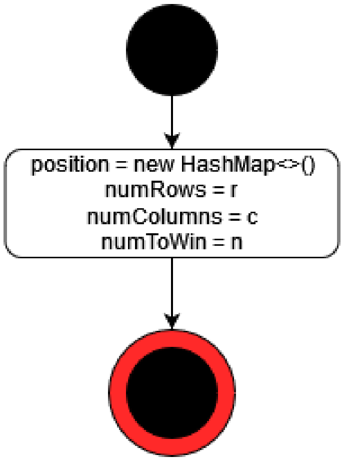
Class 6: GameBoardMem

Class Diagram

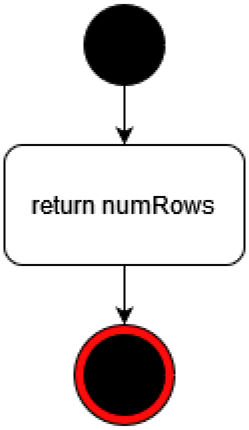

IGameBoard
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ checkIfFree(c: int): bool
+ placeToken(p: char, c: int): void
+ checkForWin(c: int): bool
+ checkTie(): bool
+ checkHorizWin(pos: Boardposition, p: char): bool
+ checkVertWin(pos: Boardposition, p: char): bool
+ checkDiagWin(pos: BoardPosition, p: char): bool
+ whatsAtPos(pos: BoardPosition, p: char): char
+ isPlayerAtPos(pos: BoardPosition, player: char): bool
GameBoardMem
- position: Map
- numColumns: int
- numRows: int
- numToWin: int
+ GameBoardMem(r: int, c: int, n: int)
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
+ whatsAtPos(pos: BoardPosition): char
+ placeToken(p: char, c: int): void

Activity diagram

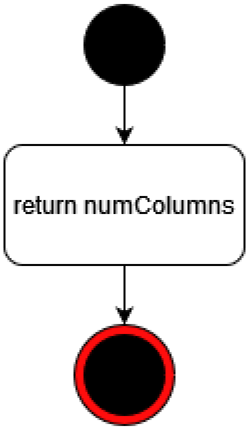
+ GameBoardMem(r: int, c: int, n: int)



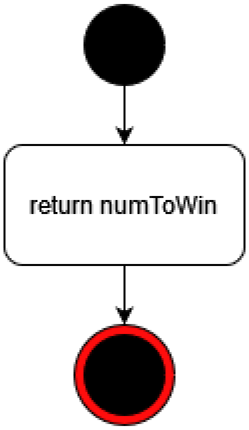
+ getNumRows(): int



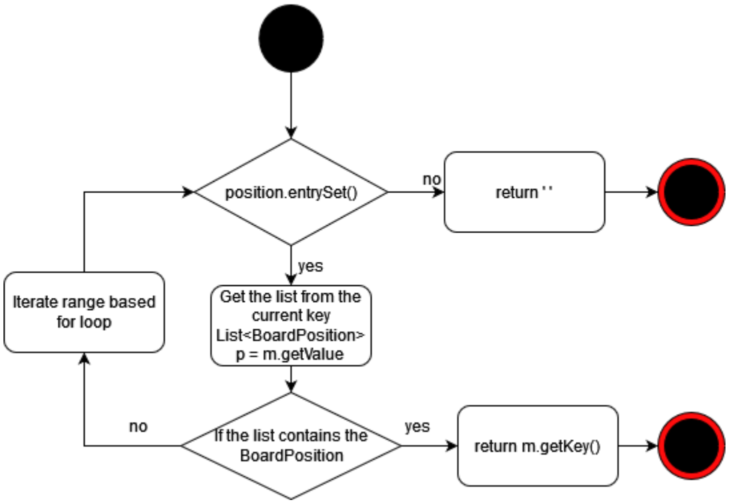
+ getNumColumns(): int



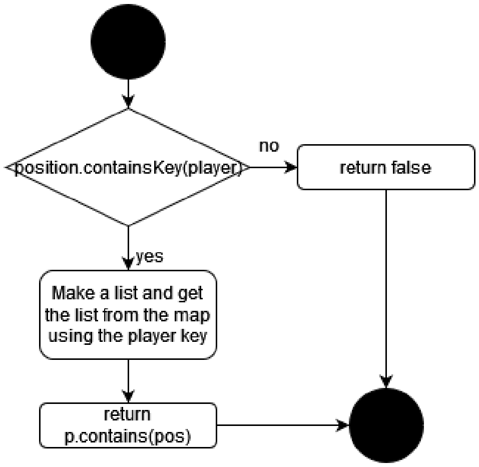
+ getNumToWin(): int



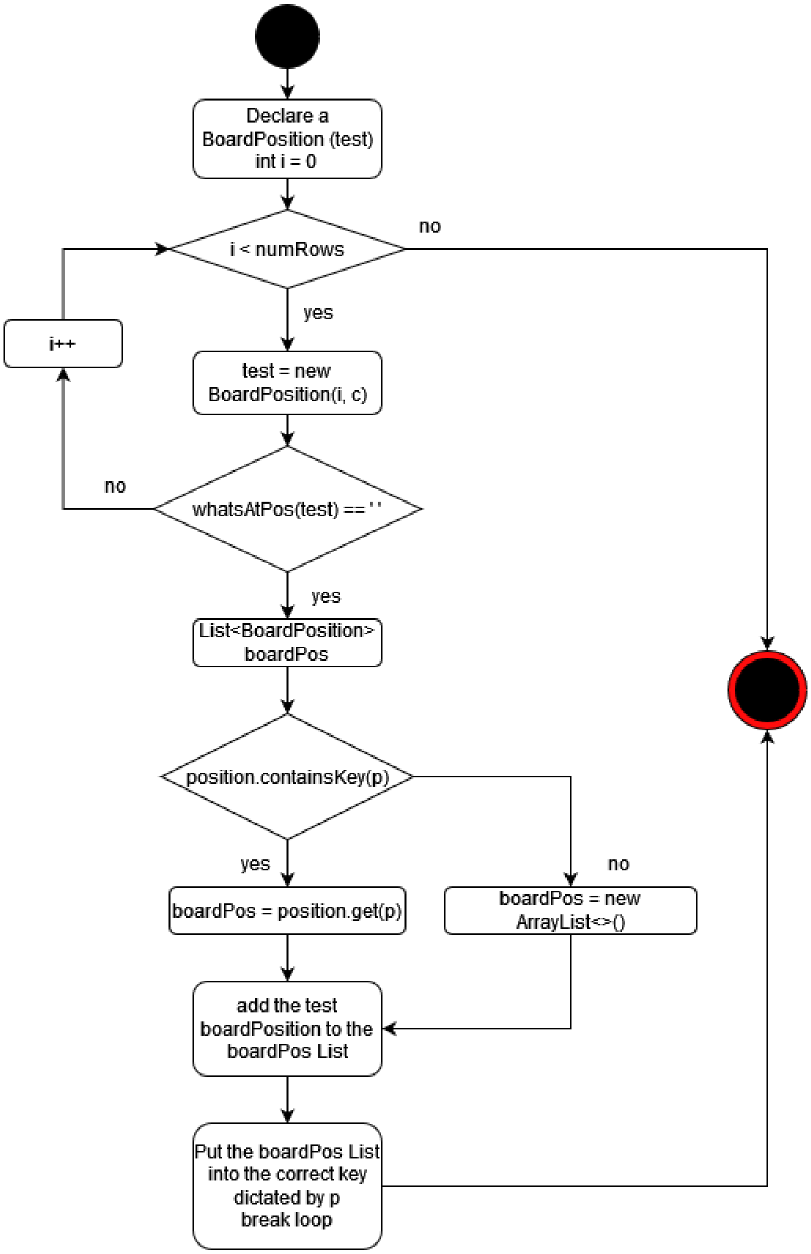
+ whatsAtPos(pos: BoardPosition): char


position.entrySet()
no
return ' '
yes
Iterate range based for loop
Get the list from the current key List<BoardPosition> p = m.getValue
If the list contains the BoardPosition
no
yes
return m.getKey()

+ isPlayerAtPos(pos: BoardPosition, player: char): bool

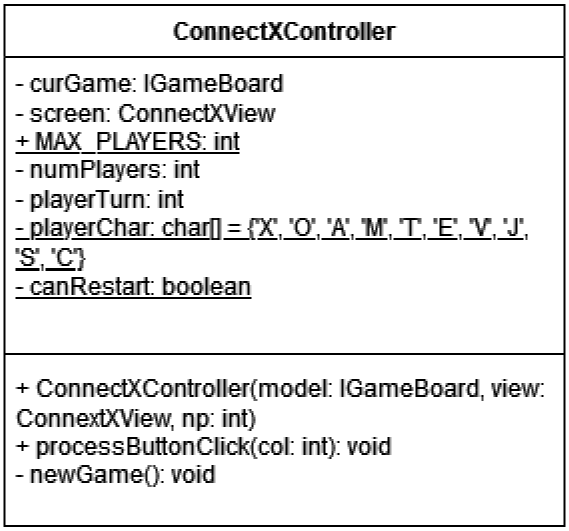

position.containsKey(player)
no
return false
yes
Make a list and get the list from the map using the player key
return p.contains(pos)

+ placeToken(p: char, c: int): void



Class 7: ConnectXController

Class Diagram

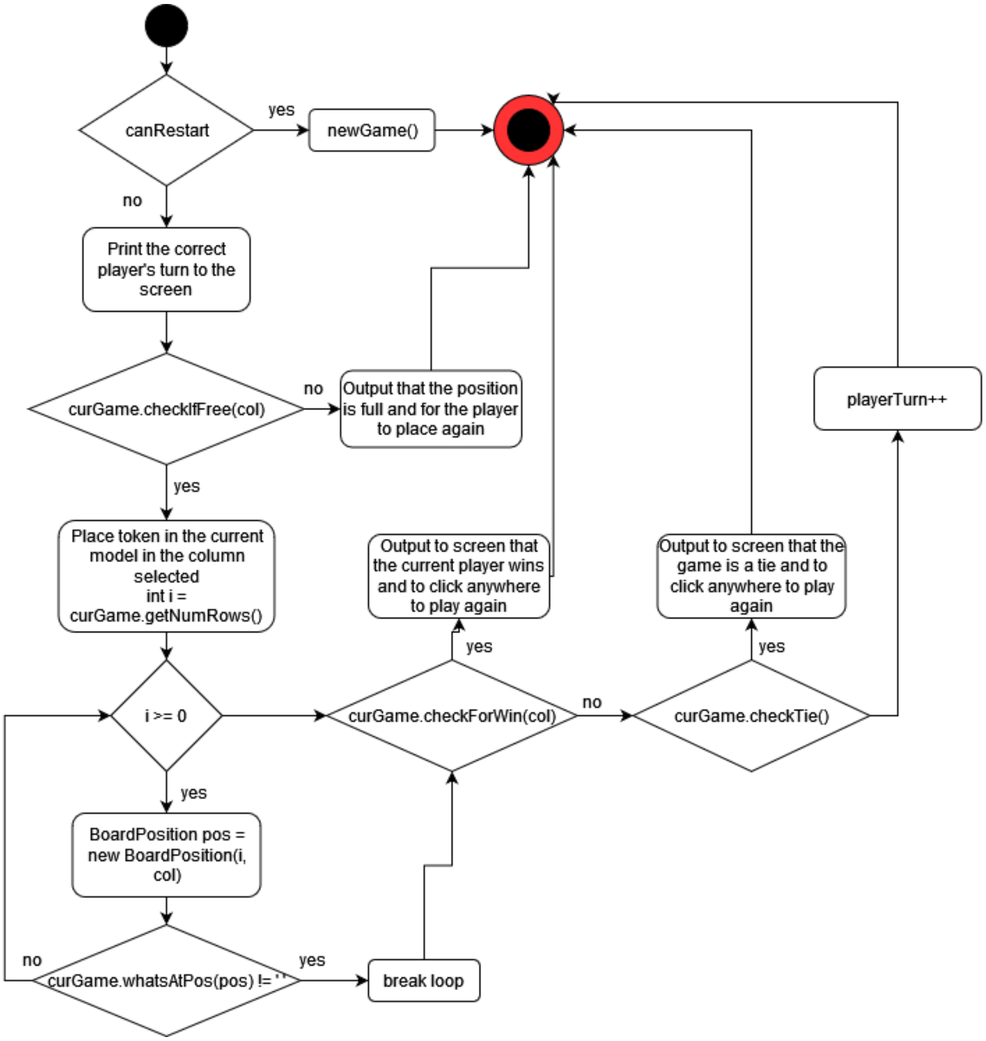


ConnectXController

- curGame: IGameBoard
- screen: ConnectXView
+ MAX_PLAYERS: int
- numPlayers: int
- playerTurn: int
- playerChar: char[] = {'X', 'O', 'A', 'M', 'T', 'E', 'V', 'J', 'S', 'C'}
- canRestart: boolean

+ ConnectXController(model: IGameBoard, view: ConnextXView, np: int)
+ processButtonClick(col: int): void
- newGame(): void

Activity Diagram


canRestart
yes
newGame()
no
Print the correct player's turn to the screen
curGame.checkIfFree(col)
no
Output that the position is full and for the player to place again
yes
Place token in the current model in the column selected
int i =
curGame.getNumRows()
playerTurn++
Output to screen that the current player wins and to click anywhere to play again
Output to screen that the game is a tie and to click anywhere to play again
yes
yes
i >= 0
curGame.checkForWin(col)
no
curGame.checkTie()
yes
BoardPosition pos = new BoardPosition(i, col)
no
yes
curGame.whatsAtPos(pos) != ' '
break loop

Test Cases

- **Input State refers to important variables that will be used during the method and what condition they will be in at the time of being called for the test cases**
- **The toString grids have proper spacing and may look off in document form but they are like that to preserve the proper output of toString**

Constructor for GameBoard

Input State	**Expected Output**	**Reason**

r = 3, c = 3, n = 3	position = [3x3 2d array of blank spaces];
	numRows = 3;
	numColumns = 3;
	numToWin = 3;
		Lower boundary
		
		Name:
		test_GameBoard_Lower_Boundary_3

r = 10, c = 5, n = 5	position = [10x5 2d array of blank spaces];
	numRows = 10;
	numColumns = 5;
	numToWin = 5;
		Rectangular input
		
		Name:
		test_GameBoard_Rectangular_Input

r = 100, c = 100, n = 25	position = [100x100 2d array of blank spaces];
	numRows = 100;
	numColumns = 100;
	numToWin = 25;
		Upper boundary
		
		Name:
		test_GameBoard_Upper_Boundary_100

Constructor for GameBoardMem

Input State	**Expected Output**	**Reason**

r = 3, c = 3, n = 3	position = new HashMap<>
	numRows = 3;
	numColumns = 3;
	numToWin = 3;
		Lower Boundary
		
		Name:
		test_GameBoard_Lower_Boundary_3

r = 10, c = 5, n = 5	position = new HashMap<>
	numRows = 10;
	numColumns = 5;
	numToWin = 5;
		Rectangular input
		
		Name:
		test_GameBoard_Rectangular_Input

r = 100, c = 100, n = 25	position = new HashMap<>
	numRows = 100;
	numColumns = 100;
		Upper boundary
		
		Name:

<table>
<tr><td></td><td>numToWin = 25;</td><td>test_GameBoard_Upper_Boundary_100</td></tr>
</table>

checkIfFree

<table>
<tr><th>Input State</th><th>Expected Output</th><th>Reason</th></tr>
<tr><td>1st column is empty
c = 0</td><td>checkIfFree = true
position = #position</td><td>Column free boundary case

<b>Name:</b>
test_checkIfFree_Column_Free</td></tr>
<tr><td>1st column is full
c = 0</td><td>checkIfFree = false
position = #position</td><td>Column full boundary case

<b>Name:</b>
test_checkIfFree_Column_Full</td></tr>
<tr><td>numColumns = 3
c = 2
toString =
| 0| 1| 2|
| | | |
| | |O |
| | |X |</td><td>checkIfFree = true
position = #position</td><td>A middle ground between full and free. Also tests max column.

<b>Name:</b>
test_checkIfFree_Column_Max_Almost_Full</td></tr>
</table>

checkHorizWin

<table>
<tr><th>Input State</th><th>Expected Output</th><th>Reason</th></tr>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(0,0)
p = ‘X’
toString =
| 0| 1| 2|
| | | |
| | | |
|X |X |X |</td><td>checkHorizWin = true
position = #position</td><td>3 in a row horizontally normally

<b>Name:</b>
test_checkHorizWin_Horizontal_Win</td></tr>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(0,0)
p = ‘X’
toString =
| 0| 1| 2|
| | | |
| | | |</td><td>checkHorizWin = false
position = #position</td><td>3 characters in a row that are not the same

<b>Name:</b>
test_checkHorizWin_Horizontal_Not_Same_Token</td></tr>
</table>

<table>
<tr><td>|X |O |X |</td><td></td><td></td></tr>
<tr><td>numRows = 3
numColumns = 4
numToWin = 3
pos = BoardPosition(0,0)
p = ‘X’
toString =
| 0| 1| 2| 3|
| | | | |
| | | | |
|X | |X |X |</td><td>checkHorizWin = false
position = #position</td><td>3 characters of the same type are in the row, but not necessarily consecutively. Making sure it’s checking for consecutive characters instead of just 3 anywhere

<b>Name:</b>
test_checkHorizWin_Horizontal_Not_Consecutive</td></tr>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(0,0)
p = ‘X’
toString =
| 0| 1| 2|
| | | |
| | | |
|X |X | |</td><td>checkHorizWin = false
position = #position</td><td>2 characters of the same type in a row when numToWin = 3 should simply return false

<b>Name:</b>
test_checkHorizWin_Horizontal_Not_NumToWin</td></tr>
</table>

checkVertWin

<table>
<tr><th>Input State</th><th>Expected Output</th><th>Reason</th></tr>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(2,0)
p = ‘X’
toString =
| 0| 1| 2|
|X | | |
|X | | |
|X | | |</td><td>checkVertWin = true
position = #position</td><td>3 in a row vertically normally

<b>Name:</b>
test_checkVertWin_Vertical_Win</td></tr>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(2,0)
p = ‘X’
toString =
| 0| 1| 2|
|X | | |
|O | | |
|X | | |</td><td>checkVertWin = false
position = #position</td><td>3 characters of the same type in a vertical that are not the same

<b>Name:</b>
test_checkVertWin_Vertical_Not_Same_Token</td></tr>
</table>

<table>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(0,0)
p = ‘X’
toString =
| 0| 1| 2|
| | | |
| | | |
|X |X |X |</td><td>checkVertWin = false
position = #position</td><td>3 characters of the same type are consecutively in a row. However, this method checks for a vertical win, not horizontal, so it would return false

<b>Name:</b>
test_checkVertWin_Vertical_Horizontal_Win</td></tr>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(1,0)
p = ‘X’
toString =
| 0| 1| 2|
| | | |
|X | | |
|X | | |</td><td>checkVertWin = false
position = #position</td><td>2 characters of the same type in a consecutive column when numToWin = 3 should simply return false

<b>Name:</b>
test_checkVertWin_Vertical_Not_NumToWin</td></tr>
</table>

<u>checkDiagWin</u>

<table>
<tr><th>Input State</th><th>Expected Output</th><th>Reason</th></tr>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(0,0)
p = ‘X’
toString =
| 0| 1| 2|
| | |X |
| |X |O |
|X |O |O |</td><td>checkDiagWin = true</td><td>3 characters of the same type consecutively in a diagonal facing the up-right, down-left direction

<b>Name:</b>
test_checkDiagWin_Diagonal_Win_Up_Right</td></tr>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(0,2)
p = ‘X’
toString =
| 0| 1| 2|
|X | | |
|O |X | |
|O |O |X |</td><td>checkDiagWin = true</td><td>3 characters of the same type consecutively in a diagonal facing the up-left, down-right direction

<b>Name:</b>
test_checkDiagWin_Diagonal_Win_Up_Left</td></tr>
<tr><td>numRows = 3
numColumns = 3</td><td>checkDiagWin = true</td><td>3 characters of the same type consecutively when the last piece</td></tr>
</table>

<table>
<tr><td>numToWin = 3
pos = BoardPosition(1,1)
p = ‘X’
toString =
| 0| 1| 2|
| | |X |
| |X |O |
|X |O |O |</td><td></td><td>inserted is in the middle of the consecutive characters. The check has to check in both directions to reach numToWin = 3

Name:
test_checkDiagWin_Diagonal_Win_Middle_Token</td></tr>
<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(0,0)
p = ‘X’
toString =
| 0| 1| 2|
| | |X |
| |O |O |
|X |O |O |</td><td>checkDiagWin = false</td><td>3 characters are consecutive and in a diagonal in the up-right, down-left direction, but one is not of the same type p.

Name:
test_checkDiagWin_Diagonal_Win_Up_Right_Not_Same_Token</td></tr>
<tr><td>numRows = 4
numColumns = 4
numToWin = 3
pos = BoardPosition(0,0)
p = ‘X’
toString =
| 0| 1| 2| 3|
			X
		X	X
		O	O
X	O	O	X	</td><td>checkDiagWin = false</td><td>3 characters of the same type are in a diagonal in the up-right, down-left direction, but they are not consecutive **Name:** test_checkDiagWin_Diagonal_Win_Up_Right_Not_Consecutive</td></tr>

<tr><td>numRows = 3
numColumns = 3
numToWin = 3
pos = BoardPosition(0,2)
p = ‘X’
toString =
| 0| 1| 2|

X		
O	O	
O	O	X	</td><td>checkDiagWin = false</td><td>3 characters are consecutive and in a diagonal in the up-left, down-right direction, but one is not of the same type p. **Name:** test_checkDiagWin_Diagonal_Win_Up_Left_Not_Same_Token</td></tr>

<tr><td>numRows = 4
numColumns = 4
numToWin = 3
pos = BoardPosition(0,0)
p = ‘X’
toString =
| 0| 1| 2| 3|

X			
X |X		|</td><td>checkDiagWin = false</td><td>3 characters of the same type are in a diagonal in the up-left, down-right direction, but they are not consecutive **Name:** test_checkDiagWin_Diagonal_Win_Up_Left_Not_Consecutive</td></tr>
</table>

<table>
<tr><td>|O |O | | |
|X |O |O |X |</td><td></td><td></td></tr>
</table>

checkTie

Input State	**Expected Output**	**Reason**

numRows = 3
numColumns = 3
position = [board full of ‘ ‘]
	checkTie = false	Check for a tie when board is completely empty
		
		Name:
		test_checkTie_Empty

numRows = 3 numColumns = 3 toString = | 0| 1| 2| |O |X |X | |X |O |O | |X |O |X |	checkTie = true	Check for a tie when the upper row is full, indicating no more spaces can be selected. **Name:** test_checkTie_Full
numRows = 3 numColumns = 3 toString = | 0| 1| 2| |O | |X | |X |O |O | |X |O |X |	checkTie = false	Check for a tie when the upper row has one single space still open. **Name:** test_checkTie_Column_Open
numRows = 3 numColumns = 3 toString = | 0| 1| 2| |X |X |X | |X |O |O | |X |O |O | checkForWin = true	checkTie = true	This is to show that checkTie does not actually care about the winCondition, and should be called only after checking the win condition. **Name:** test_checkTie_Full_with_Win

whatsAtPos

Input State	**Expected Output**	**Reasoning**

numRows = 3
numColumns = 3
position = [board full of ‘ ‘]
pos = BoardPosition(0,0)
	whatsAtPos = ‘ ‘	Testing with an empty game board
		
		Name:
		test_whatsAtPos_Empty

numRows = 3 numColumns = 3 toString = | 0| 1| 2| | | | | | | | |	whatsAtPos = ‘X’	Testing with a character in the position being at 0, 0 **Name:** test_whatsAtPos_X_at_Location

<table>
<tr><th></th><th></th><th></th></tr>
<tr><td>|X | | |
pos = BoardPosition(0,0)</td><td></td><td></td></tr>
<tr><td>numRows = 3
numColumns = 3
toString =
| 0| 1| 2|
| | | |
| |X | |
|X |O |O |
pos = BoardPosition(1,1)</td><td>whatsAtPos = ‘X’</td><td>Testing reading a position in a central part of the board

<b>Name:</b>
test_whatsAtPos_X_at_Center</td></tr>
<tr><td>numRows = 3
numColumns = 3
toString =
| 0| 1| 2|
| | | |
| | | |
|X |O |X |
pos = BoardPosition(1,1)</td><td>whatsAtPos = ‘ ’</td><td>Testing making sure whatsAtPos locates the correct location in a board instead of just selecting the first filled character it finds

<b>Name:</b>
test_whatsAtPos_Space_at_Center_Non_Empty</td></tr>
<tr><td>numRows = 3
numColumns = 3
toString =
| 0| 1| 2|
| | |T |
|O |T |O |
|T |O |T |
pos = BoardPosition(2,2)</td><td>whatsAtPos = ‘T’</td><td>Checks a boundary case at max columns and max rows to see if whatsAtPos works

<b>Name:</b>
test_whatsAtPos_Max_Boundary</td></tr>
</table>

isPlayerAtPos

<table>
<tr><th>Input State</th><th>Expected Output</th><th>Reasoning</th></tr>
<tr><td>numRows = 3
numColumns = 3
position = [board full of ‘ ‘]
pos = BoardPosition(0,0)
player = ‘X’</td><td>isPlayerAtPos = false</td><td>Testing with an empty game board

<b>Name:</b>
test_isPlayerAtPos_Empty_GameBoard</td></tr>
<tr><td>numRows = 3
numColumns = 3
toString =
| 0| 1| 2|
| | | |
| | | |
|X | | |
pos = BoardPosition(0,0)
player = ‘X’</td><td>isPlayerAtPos = true</td><td>Testing with a character of the same type as player is in the position being checked at 0, 0

<b>Name:</b>
test_isPlayerAtPos_Value_At_Pos</td></tr>
</table>

<table>
<tr><td>numRows = 3
numColumns = 3
toString =
| 0| 1| 2|
| | | |
| | | |
|O | | |
pos = BoardPosition(0,0)
player = ‘X’</td><td>isPlayerAtPos = false</td><td>Testing with a different character at the checked position than the player character

<b>Name:</b>
test_isPlayerAtPos_Wrong_Value_At_Pos</td></tr>
<tr><td>numRows = 3
numColumns = 3
toString =
| 0| 1| 2|
| | | |
| |X | |
|X |O |X |
pos = BoardPosition(0,1)
player = ‘O’</td><td>isPlayerAtPos = true</td><td>Testing making sure isPlayerAtPos checks the correct location in a board instead of just selecting the first one it finds

<b>Name:</b>
test_isPlayerAtPos_Check_Position_Finding_Center</td></tr>
<tr><td>numRows = 3
numColumns = 3
toString =
| 0| 1| 2|
| | |T |
|O |T |O |
|T |O |T |
pos = BoardPosition(2,2)
player = ‘T’</td><td>isPlayerAtPos = true</td><td>Checks a boundary case at max columns and max rows to see if whatsAtPos works

<b>Name:</b>
test_isPlayerAtPos_Max_Boundary</td></tr>
</table>

placeToken

<table>
<tr><th>Input State</th><th>Expected Output</th><th>Reason</th></tr>
<tr><td>numRows = 3
numColumns = 3
p = ‘X’
c = 0
toString =
| 0| 1| 2|
		|</td><td>toString = | 0| 1| 2
X		|</td><td>Testing placing a character in an empty board <b>Name:</b> test_placeToken_Empty_GameBoard</td></tr>
<tr><td>numRows = 3 numColumns = 3 p = ‘O’ c = 0 toString = | 0| 1| 2|</td><td>toString = | 0| 1| 2|
O		
X		|</td><td>Testing placing a character in a column with a value in it <b>Name:</b> test_placeToken_Char_In_Column</td></tr>
</table>

<table>
<tr><td>| | | |
| | | |
|X | | |</td><td></td><td></td></tr>
<tr><td>numRows = 3
numColumns = 3
p = ‘X’
c = 0
toString =
| 0| 1| 2|
O		
X		|</td><td>toString = | 0| 1| 2
X		
O		
X		|</td><td>Testing placing a character in a column that is almost full, boundary case <b>Name:</b> test_placeToken_Column_Almost_Full</td></tr>

<tr><td>numRows = 3
numColumns = 3
p = ‘O’
c = 1
toString =
| 0| 1| 2|

X		
O		
X		|</td><td>toString = | 0| 1| 2
X		
O		
X |O	|</td><td>Testing placing a character in a different column when one column is full <b>Name:</b> test_placeToken_Another_Column_Full</td></tr>

<tr><td>numRows = 3
numColumns = 3
p = ‘X’
c = 0
toString =
| 0| 1| 2|

X	O	
O	X	O
X |O |X |</td><td>toString = | 0| 1| 2
X	O	X
O	X	O
X	O	X	</td><td>Testing placing a character when the entire board is nearly full <b>Name:</b> test_placeToken_GameBoard_Almost_Full</td></tr>
</table>