CPSC 2150 Project Report

Davis Little

Requirements Analysis

Functional Requirements:

1. As a player, I need to be able to start the game so that I can play the game.
2. As a player, I need to be able to see my options at the start of the game, so I know what keys to press to play.
3. As a player, I need to be able to see whose turn it is so I know whether to make a move or not.
4. As a player, I want to be able to see the game board, so I know what is happening.
5. As a player, I need to see the number for each row, so I know what to input.
6. As a player, I need to know which player is which, so I know which player I am
7. As a player, I need to know who wins the game so I can tell if I won or not.
8. As a player, I need the program to give me the option to play again so I can play again or quit if I want to.
9. As a player, I need the game to tell me if I selected an invalid column number so I can select a valid one.
10. As a player, I need to be able to place a marker on to the board
11. As a player, I need to be able to pick again if I pick a full column on accident
12. As a player, I need to be able to pick again if I pick a nonexistent column
13. As a player, I need to win when I get 5 in a row horizontally
14. As a player, I need to win when I get 5 in a row vertically
15. As a player, I need to win when I get 5 in a row diagonally
16. As a player, I need to know when there is a tie game
17. As a player, I need to be able to move if my opponent didn't win

Non-Functional Requirements

1. The program must reliably run without crashing.
2. The program must run in a GUI
3. The board must be within the max and minimum size given by the program
4. (0, 0) must always be at the bottom left of the board
5. The program must be easy to use.
6. The program must have good performance.
7. The program must be reliable and not do anything unexpected.
8. The program must be secure and can't modify things it shouldn't be modifying in your computer.

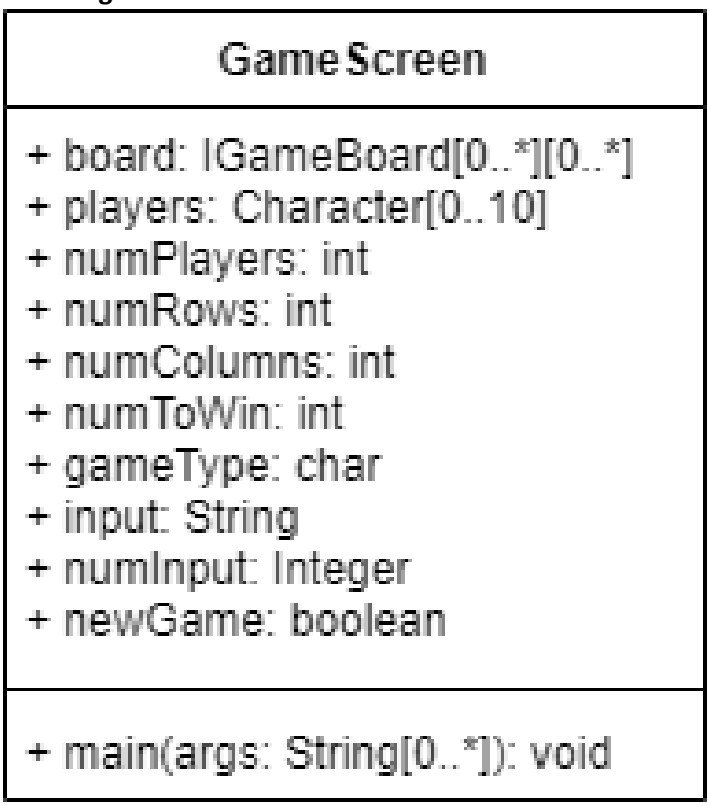
Deployment Instructions

Runs in IntelliJ

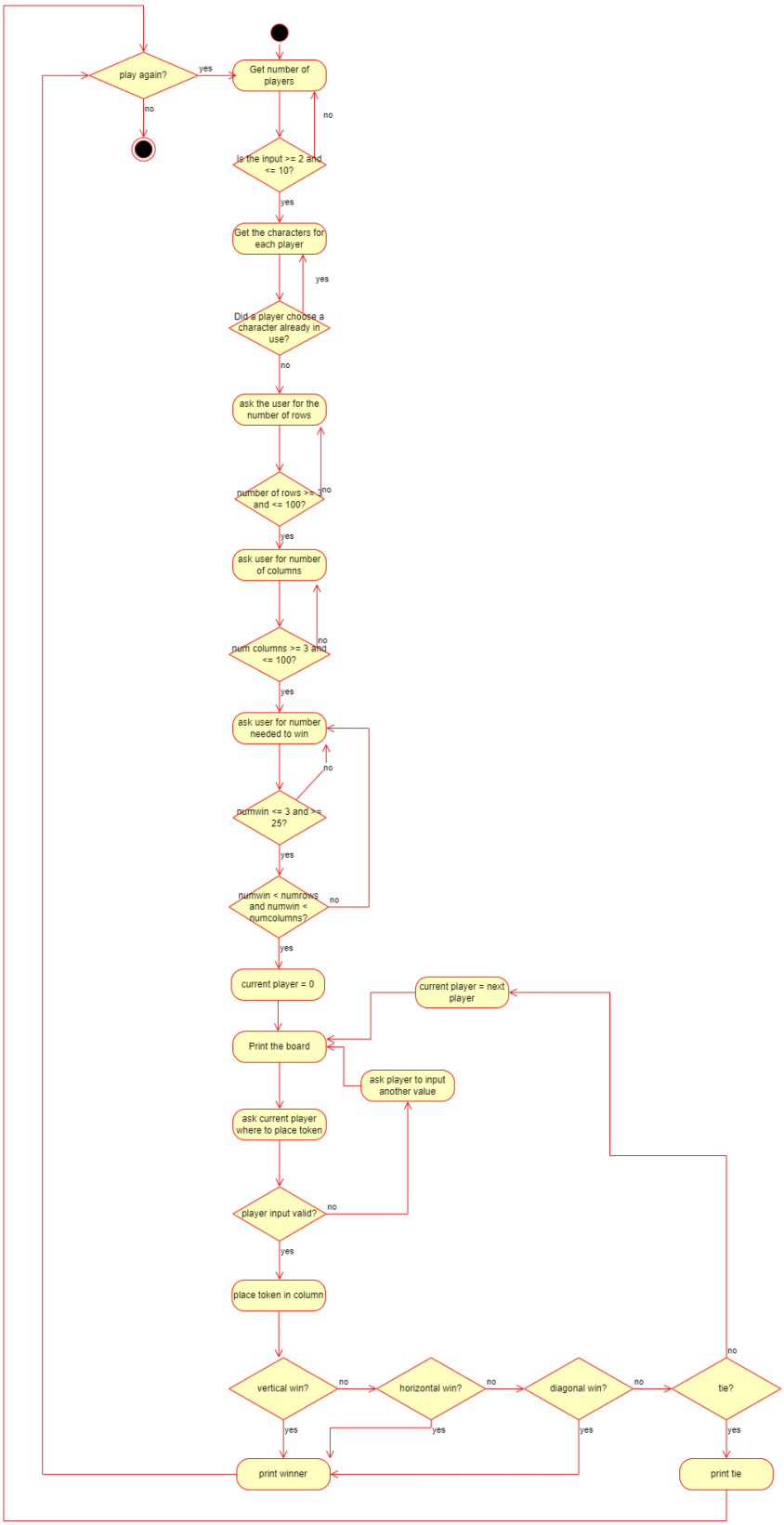
System Design

Class 1: GameScreen.java

Class diagram

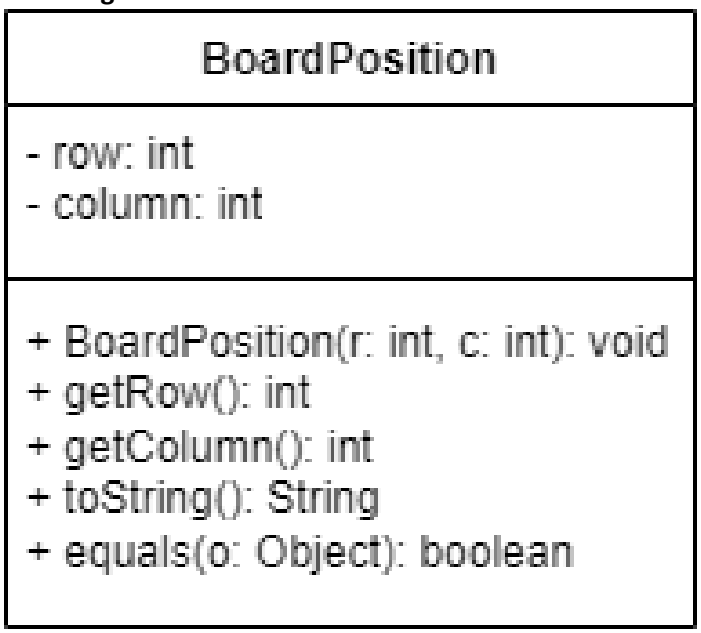


Activity diagrams



Class 2: BoardPosition.java

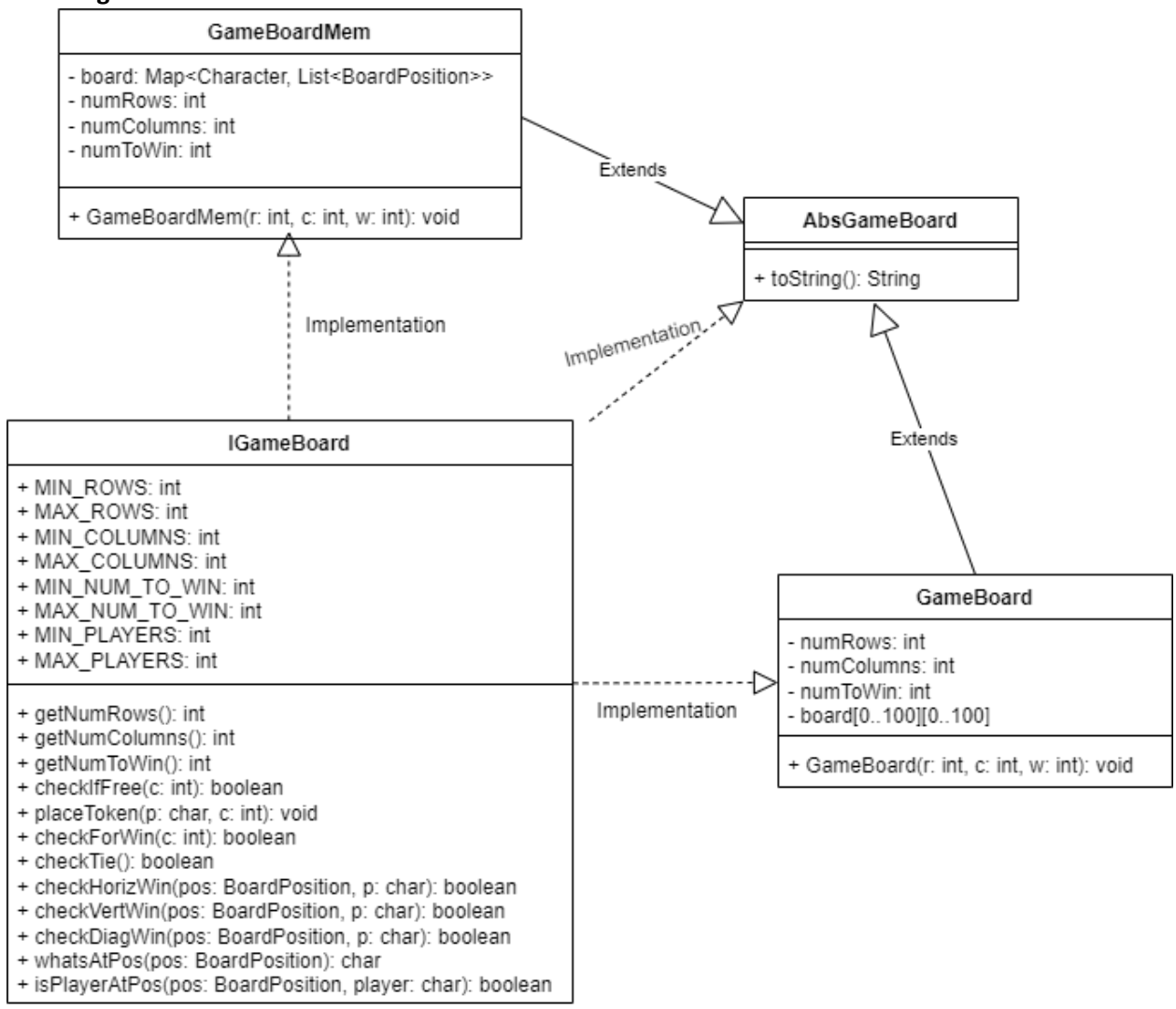
Class diagram



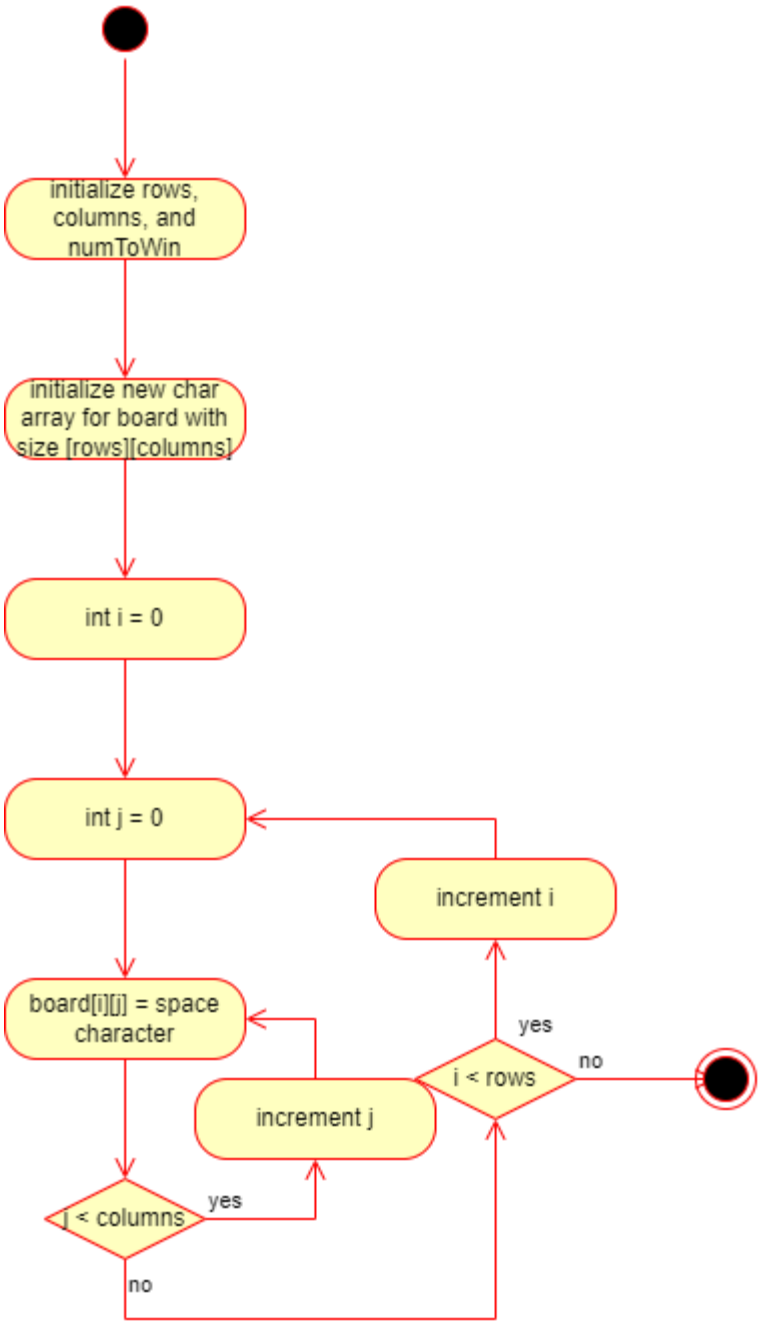
Activity diagrams

Class 3: GameBoard.java

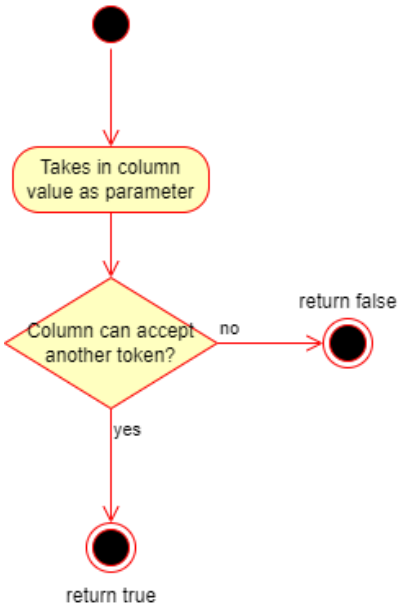
Class diagram



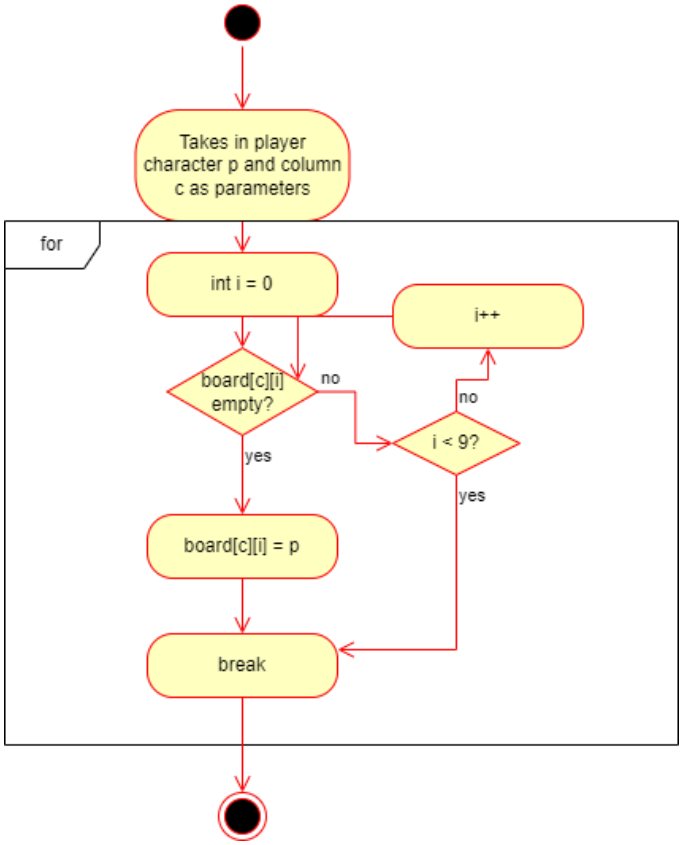
Activity diagrams GameBoard:



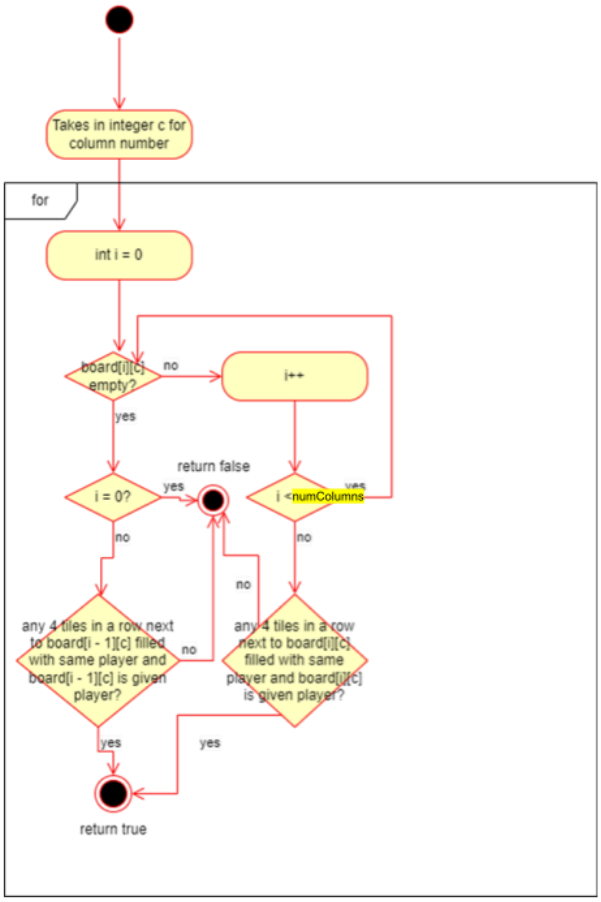
checkIfFree:



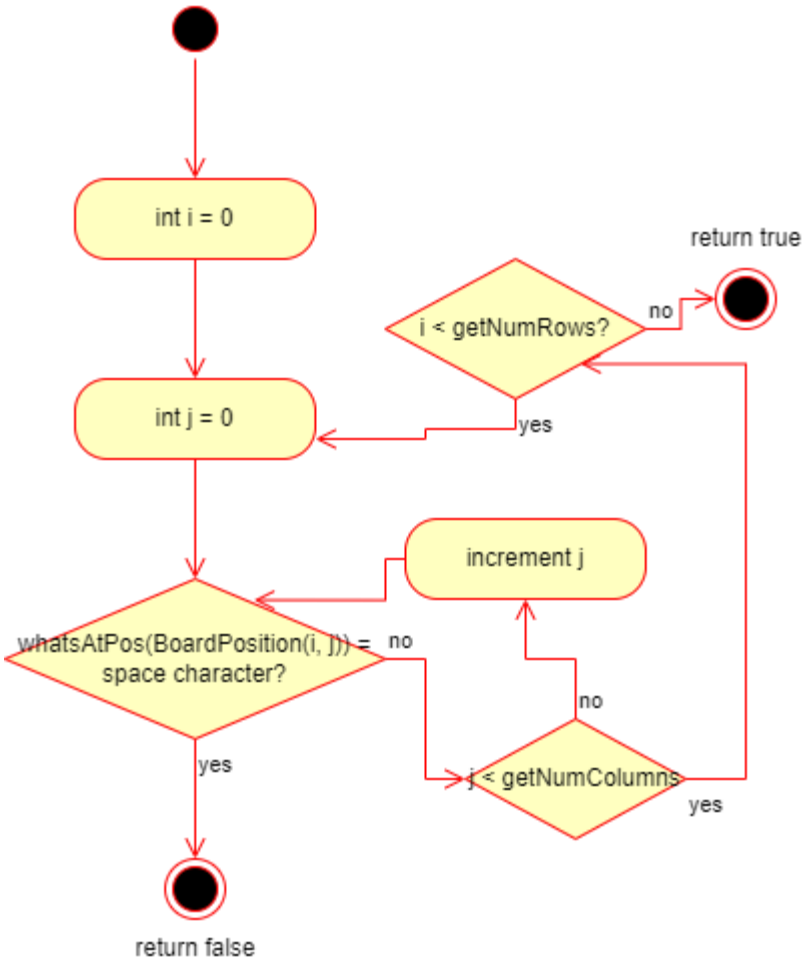
placeToken:



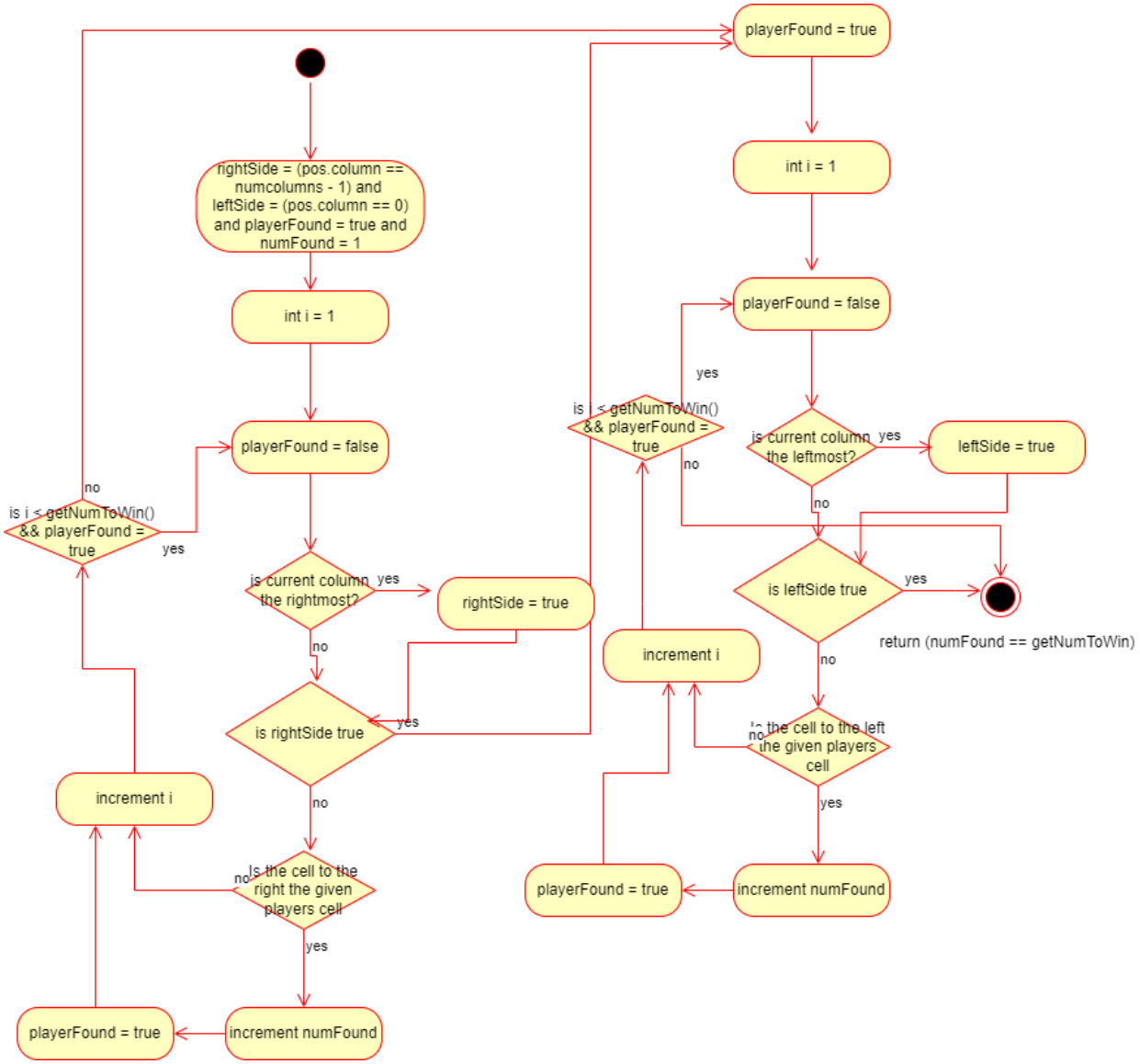
checkForWin:


Takes in integer c for column number
for
int i = 0
board[i][c] empty?
no
i++
yes
return false
i = 0?
yes
i <numColumns
yes
no
no
any 4 tiles in a row next to board[i - 1][c] filled with same player and board[i - 1][c] is given player?
no
any 4 tiles in a row next to board[i][c] filled with same player and board[i][c] is given player?
yes
yes
return true

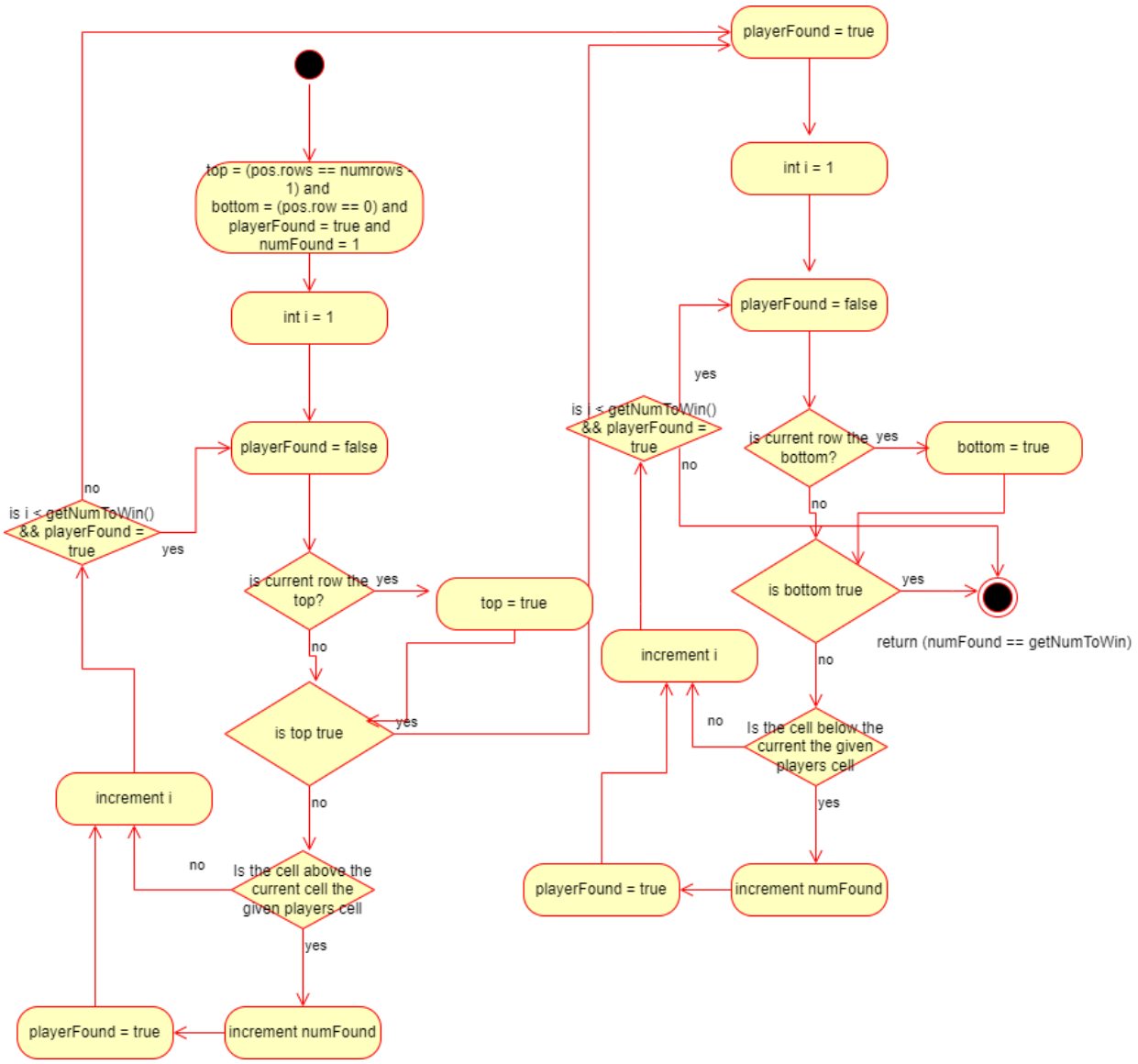
checkTie:



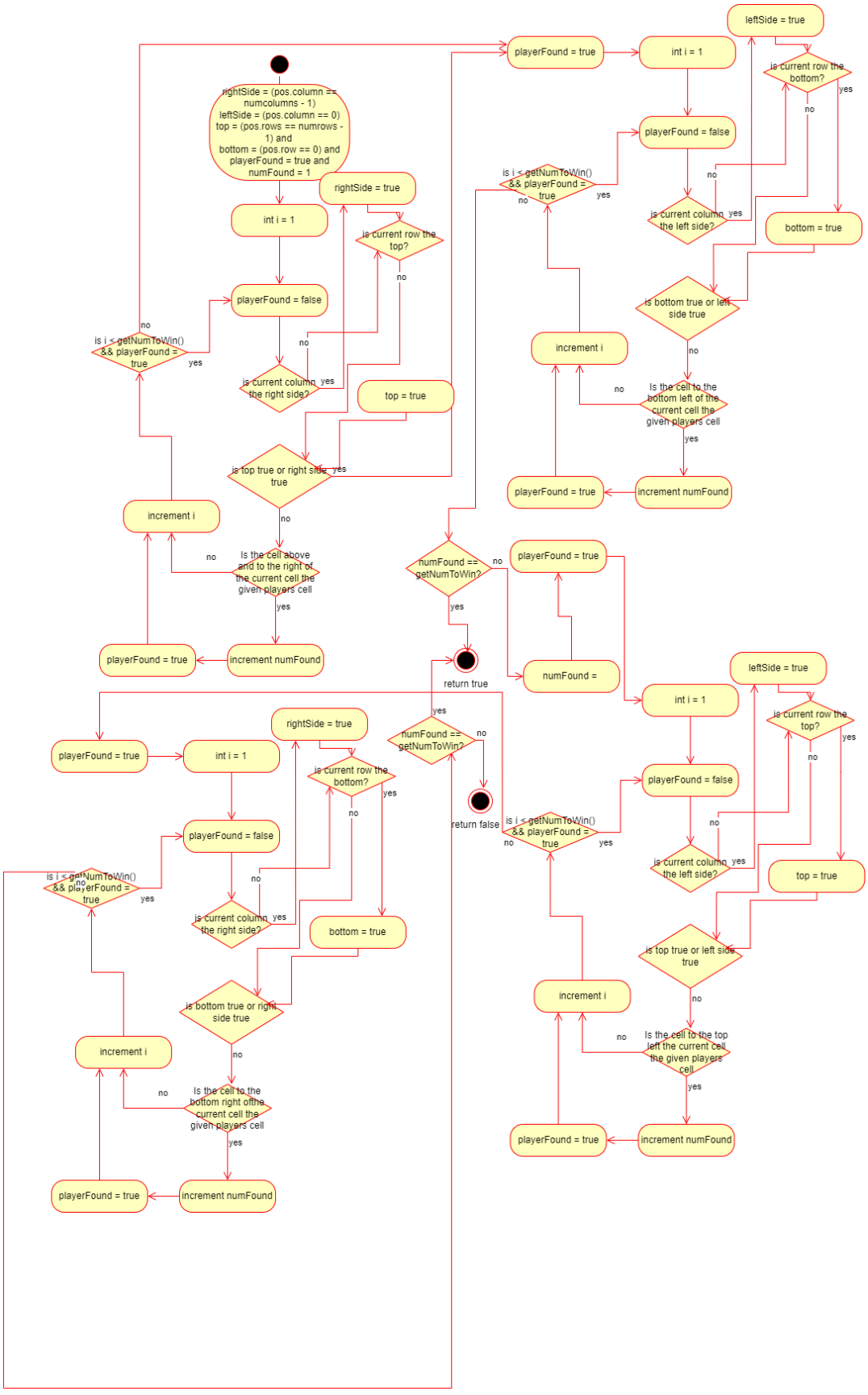
checkHorizWin:



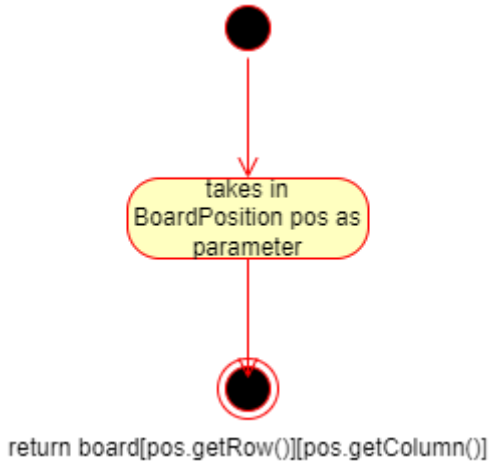
checkVertWin:



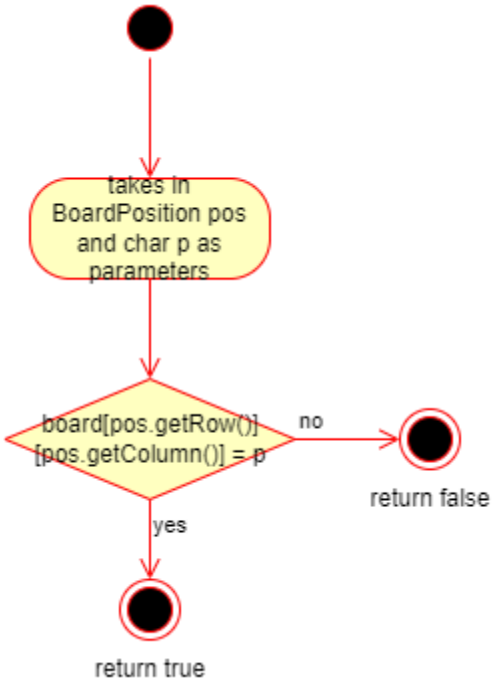
checkDiagWin:


rightSide = (pos.column == numcolumns - 1)
leftSide = (pos.column == 0)
top = (pos.rows == numrows - 1) and
bottom = (pos.row == 0) and
playerFound = true and
numFound = 1
int i = 1
playerFound = false
is i < getNumToWin() && playerFound = true
yes
no
is current column the right side?
rightSide = true
is current row the top?
top = true
is top true or right side true
increment i
Is the cell above and to the right of the current cell the given players cell
increment numFound
playerFound = true
numFound == getNumToWin?
return true
playerFound = true
int i = 1
playerFound = false
is i < getNumToWin() && playerFound = true
is current column the left side?
leftSide = true
is current row the bottom?
bottom = true
is bottom true or left side true
increment i
Is the cell to the bottom left of the current cell the given players cell
increment numFound
playerFound = true
numFound =
numFound == getNumToWin?
return false
playerFound = true
int i = 1
playerFound = false
is i < getNumToWin() && playerFound = true
is current column the right side?
rightSide = true
is current row the bottom?
bottom = true
is bottom true or right side true
increment i
Is the cell to the bottom right of the current cell the given players cell
increment numFound
playerFound = true
leftSide = true
is current row the top?
playerFound = false
is i < getNumToWin() && playerFound = true
is current column the left side?
top = true
is top true or left side true
increment i
Is the cell to the top left the current cell the given players cell
increment numFound
playerFound = true

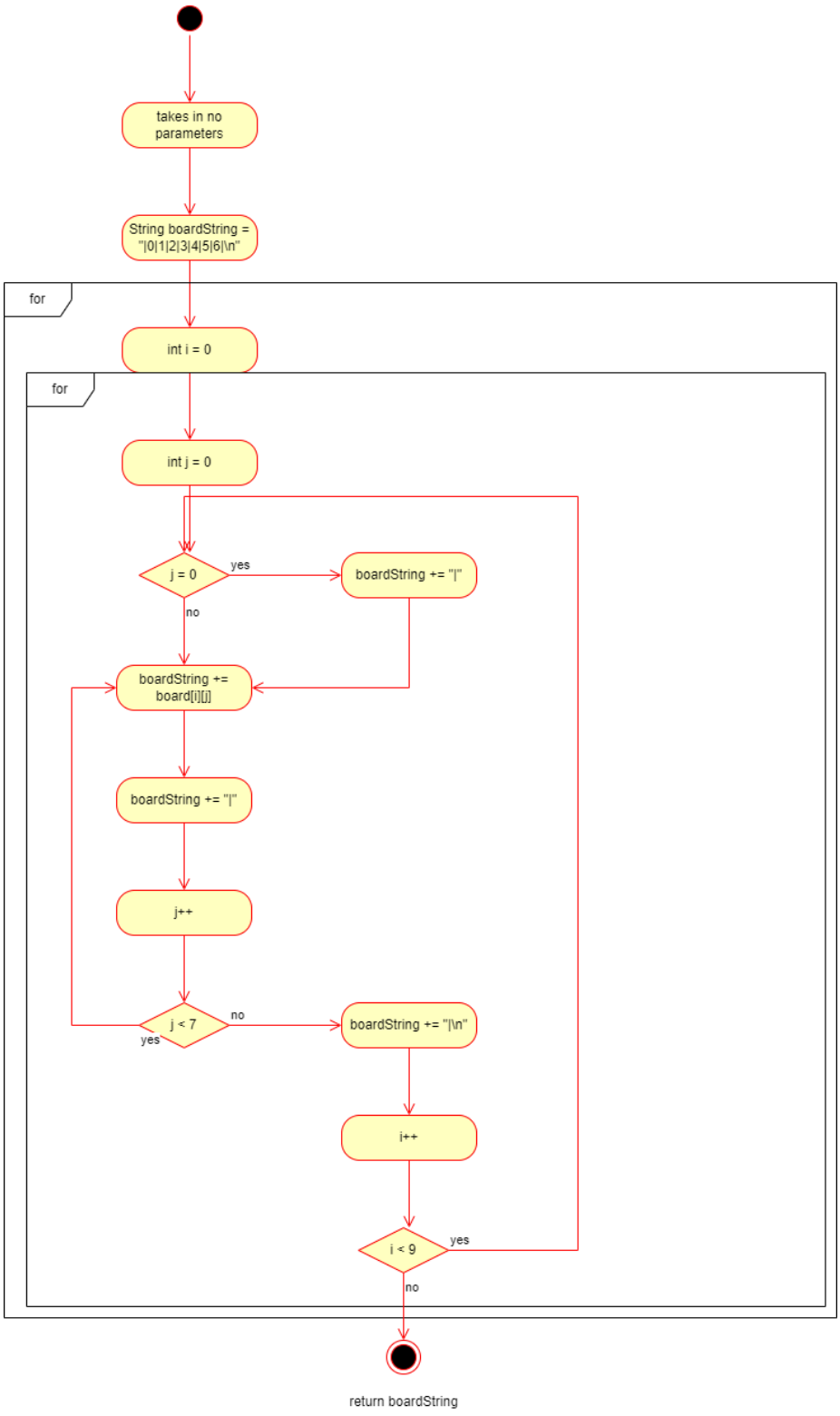
whatsAtPos:



isPlayerAtPos:



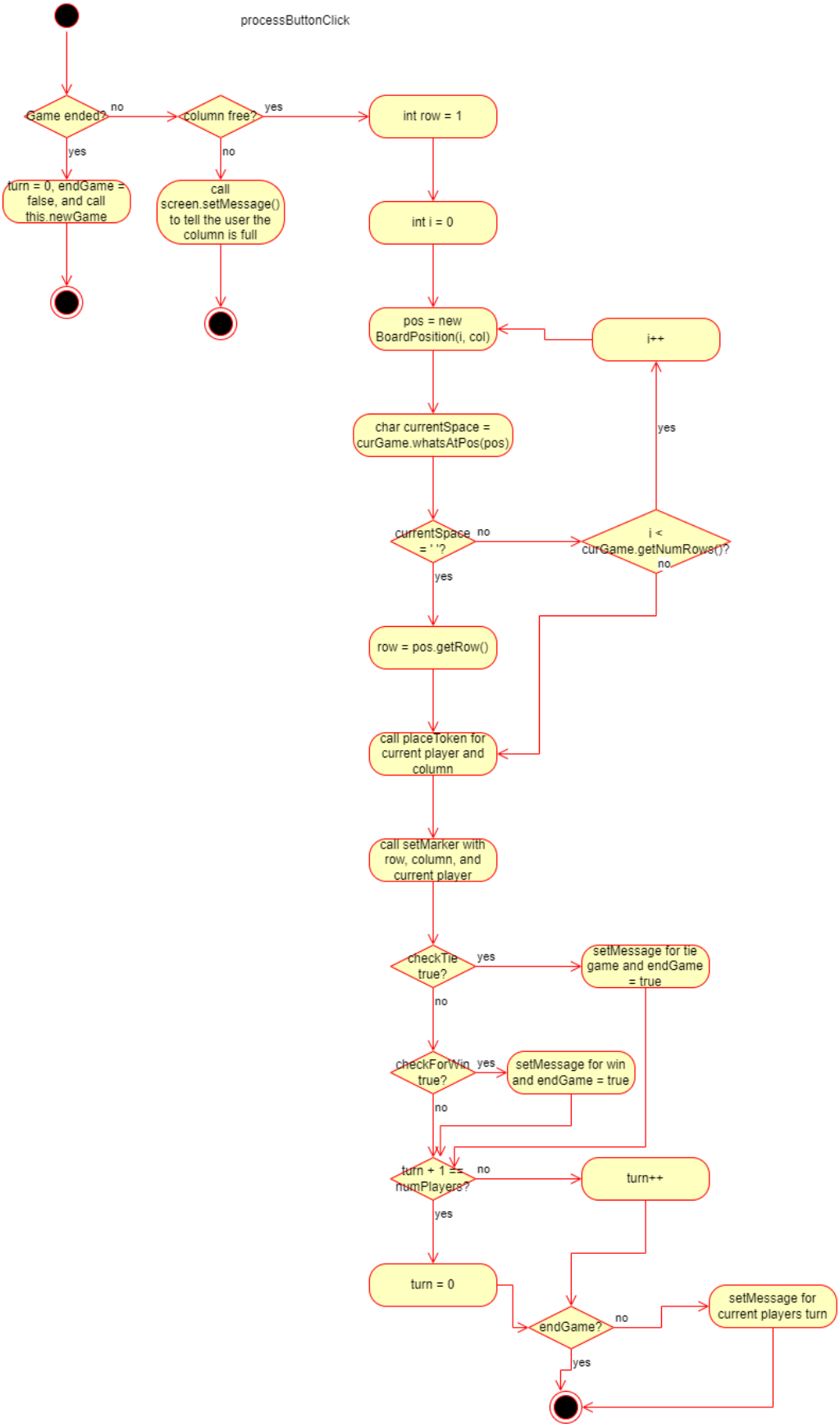
toString:


takes in no parameters
String boardString = "|0|1|2|3|4|5|6|\n"
for
int i = 0
for
int j = 0
j = 0
yes
boardString += "|"
no
boardString += board[i][j]
boardString += "|"
j++
j < 7
no
yes
boardString += "|\n"
i++
i < 9
yes
no
return boardString

Class 4: ConnectXController

ConnectXController
- curGame: IGameBoard - screen: ConnectXView - turn: int - players: char[10] - endGame: boolean - numPlayers: int + MAX_PLAYERS: int
+ ConnectXController(IGameBoard, ConnectXView, int): void + processButtonClick(int): void - newGame(): void

Activity Diagrams:


processButtonClick
Game ended?
no
yes
column free?
yes
no
turn = 0, endGame = false, and call this.newGame
call screen.setMessage() to tell the user the column is full
int row = 1
int i = 0
pos = new BoardPosition(i, col)
i++
char currentSpace = curGame.whatsAtPos(pos)
currentSpace = ' '?
no
yes
i < curGame.getNumRows()?
yes
no
row = pos.getRow()
call placeToken for current player and column
call setMarker with row, column, and current player
checkTie true?
yes
setMessage for tie game and endGame = true
no
checkForWin true?
yes
setMessage for win and endGame = true
no
turn + 1 == numPlayers?
no
turn++
yes
turn = 0
endGame?
no
setMessage for current players turn
yes

Test Cases

public GameBoard(int r, int c, int w)

Input:

State: nothing initialized

r = 100
c = 100
w = 25

Output:

State:

	0	1	2	…	99
99					
…					
2					
1					
0					

Reason:

This test case is unique and distinct because it is testing the maximum values for each variable.

Function Name:

test_constructor_maxValues

Input:

State: nothing initialized

r = 3
c = 3
w = 3

Output:

State:

	0	1	2
2			
1			
0			

Reason:

This test case is unique and distinct because it is testing the minimum values for each variable

Function Name:

test_constructor_minValues

Input:

State: nothing initialized

r = 20 c
= 20 w
= 10

Output:

State:

	0	1	2	...	20
20					
...					
2					
1					
0					

Reason:

This test case is unique and distinct because it is testing random values between min and max.

Function Name:

test_constructor_randValues

public boolean checkIfFree(int c)

Input:	**Output:**	**Reason:**
State: | | 0 | 1 | 2 | | 2 | | | | | 1 | | | | | 0 | | | | c = 1	checkIfFree = true; State: board unchanged	This test case is unique and distinct because it tests that checkIfFree returns correctly when the column is empty **Function Name:** test_checkIfFree_empty

Input:	**Output:**	**Reason:**
State: | | 0 | 1 | 2 | | 2 | | | | | 1 | | | | | 0 | | X | | c = 1	checkIfFree = true; State: board unchanged	This test case is unique and distinct because it tests that checkIfFree returns correctly when the column has values but is not full **Function Name:** test_checkIfFree_someFilled

Input:	**Output:**	**Reason:**
State: | | 0 | 1 | 2 | | 2 | | X | | | 1 | | O | | | 0 | | X | | c = 1	checkIfFree = false State: board unchanged	This test case is unique and distinct because it tests that checkIfFree returns correctly when the column is full **Function Name:** test_checkIfFree_full

public boolean checkHorizWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: | | 0 | 1 | 2 | | 2 | | | | | 1 | | | | | 0 | | | | pos.getRow = 0 pos.getColumn = 0 p = 'X'	checkHorizWin = false State:board unchanged	This test case is unique and distinct because it tests that checkHorizWin returns correctly when the board is empty **Function Name:** test_checkHorizWin_empty

Input:	**Output:**	**Reason:**
State: | | 0 | 1 | 2 | | 2 | | | | | 1 | | | | | 0 | X | | | pos.getRow = 0 pos.getColumn = 0 p = 'X'	checkHorizWin = false State:board unchanged	This test case is unique and distinct because it tests that checkHorizWin returns correctly when the board has one value equal to the player **Function Name:** test_checkHorizWin_oneValue

Input:	**Output:**	**Reason:**
State: | | 0 | 1 | 2 | | 2 | | | | | 1 | | | | | 0 | X | X | X | pos.getRow = 0 pos.getColumn = 1 p = 'X'	checkHorizWin = true State:board unchanged	This test case is unique and distinct because it tests that checkHorizWin returns correctly when the board has three in a row of char p **Function Name:** test_checkHorizWin_threeInARowWin

Input:	**Output:**	**Reason:**
State: | | 0 | 1 | 2 | | 2 | | | | | 1 | | | | | 0 | X | O | X | pos.getRow = 0 pos.getColumn = 0 p = 'X'	checkHorizWin = false State:board unchanged	This test case is unique and distinct because it tests that checkHorizWin returns correctly when the board has three in a row, but one of those is not char p **Function Name:** test_checkHorizWin_threeInARowNoWin

public boolean checkVertWin(BoardPosition pos, char p)

Input:	**Output:**	**Reason:**
State: | | 0 | 1 | 2 | | 2 | | | | | 1 | | | | | 0 | | | | pos.getRow = 0 pos.getColumn = 0 p = 'X'	checkVertWin = false State: board unchanged	This test case is unique and distinct because it tests that checkVertWin returns correctly when the board is empty **Function Name:** test_checkVertWin_empty

Input:	**Output:**	**Reason:**
State: | | 0 | 1 | 2 | | 2 | | | | | 1 | X | | | | 0 | X | | | pos.getRow = 0 pos.getColumn = 0 p = 'X'	checkVertWin = false State: board unchanged	This test case is unique and distinct because it tests that checkVertWin returns correctly when the board has char p at pos, but not enough in a row to win **Function Name:** test_checkVertWin_twoInARow

Input:

State:

	0	1	2
2	X		
1	X		
0	X		

pos.getRow = 1
pos.getColumn = 0
p = 'X'

Output:

checkVertWin = true

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkVertWin returns correctly when the board has numWin in a row vertically

Function Name:

test_checkVertWin_threeInARowWin

Input:

State:

	0	1	2
2	X		
1	O		
0	X		

pos.getRow = 0
pos.getColumn = 0
p = 'X'

Output:

checkVertWin = false

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkVertWin returns correctly when the board has three in a row, but they are not all char p

Function Name:

test_checkVertWin_threeInARowNoWin

public boolean checkDiagWin(BoardPosition pos, char p)

Input:

State:

	0	1	2
2			
1			
0			

pos.getRow = 0
pos.getColumn = 0
p = 'X'

Output:

checkDiagWin = false

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkDiagWin returns correctly when the board is empty

Function Name:

test_checkDiagWin_empty

Input:

State:

	0	1	2
2			
1			
0		X	

pos.getRow = 0
pos.getColumn = 1
p = 'X'

Output:

checkDiagWin = false

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkDiagWin returns correctly when the board only has one char p

Function Name:

test_checkDiagWin_oneValue

Input:

State:

	0	1	2
2			
1		X	
0	X	O	

pos.getRow = 0
pos.getColumn = 0
p = 'X'

Output:

checkDiagWin = false

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkDiagWin returns correctly when the board has only two in a row diagonally to the top right/bottom left

Function Name:

test_checkDiagWin_twoValues_topRight

Input:

State:

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

pos.getRow = 1
pos.getColumn = 1
p = 'X'

Output:

checkDiagWin = true

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkDiagWin returns correctly when the board has 3 in a row diagonally to the top right/bottom left

Function Name:

test_checkDiagWin_threeValues_topRight

Input:

State:

	0	1	2
2			
1	X		
0	O	X	

pos.getRow = 1
pos.getColumn = 0
p = 'X'

Output:

checkDiagWin = false

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkDiagWin returns correctly when the board has two values to the top left/bottom right

Function Name:

test_checkDiagWin_twoValues_topLeft

Input:

State:

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

pos.getRow = 1
pos.getColumn = 1
p = 'X'

Output:

checkDiagWin = true

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkDiagWin returns correctly when the board has three values to the top left/bottom right

Function Name:

test_checkDiagWin_threeValues_topLeft

Input:

State:

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

pos.getRow = 2
pos.getColumn = 0
p = 'X'

Output:

checkDiagWin = false

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkDiagWin returns correctly when the board has three values in a row diagonally, but not of the same char p

Function Name:

test_checkDiagWin_notSameChar

public boolean checkTie()

Input:

State:

	0	1	2
2			
1			
0			

Output:

checkTie = false

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkTie returns correctly when the board is empty

Function Name:

test_checkTie_empty

Input:

State:

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

Output:

checkTie = true

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkTie returns correctly when the board is full

Function Name:

test_checkTie_full

Input:

State:

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

Output:

checkTie = false

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkTie returns correctly when the board is almost empty

Function Name:

test_checkTie_almostEmpty

Input:

State:

	0	1	2
2			
1			
0		X	

Output:

checkTie = false

State: board unchanged

Reason:

This test case is unique and distinct because it tests that checkTie returns correctly when the board only has one value

Function Name:

test_checkTie_oneValue

public char whatsAtPos(BoardPosition pos)

Input:

State:

	0	1	2
2			
1			
0			

pos.getRow = 0 pos.getColumn = 0

Output:

whatsAtPos = ' ';

State: board unchanged

Reason:

This test case is unique and distinct because it tests that whatsAtPos returns correctly when the board is empty

Function Name:

test_whatsAtPos_empty

Input:

State:

	0	1	2
2			
1			
0		X	

Output:

whatsAtPos = ' ';

State: board unchanged

Reason:

This test case is unique and distinct because it tests that whatsAtPos returns correctly when the board has a character, but it's not at pos

Function Name:

pos.getRow = 0 pos.getColumn = 0		test_whatsAtPos_charNotAtPos

Input: State: <table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>0</td><td>X</td><td></td><td></td></tr></table> pos.getRow = 0 pos.getColumn = 0	**Output:** whatsAtPos = 'X'; State: board unchanged	**Reason:** This test case is unique and distinct because it tests that whatsAtPos returns correctly when the board has a character only at pos **Function Name:** test_whatsAtPos_charAtPos

Input: State: <table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>0</td><td>X</td><td>O</td><td>X</td></tr></table> pos.getRow = 0 pos.getColumn = 1	**Output:** whatsAtPos = 'O'; State: board unchanged	**Reason:** This test case is unique and distinct because it tests that whatsAtPos returns correctly when the board has a character at pos and other characters around it **Function Name:** test_whatsAtPos_charAtPosAndOthers

Input:

State:

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

pos.getRow = 1 pos.getColumn = 1

Output:

whatsAtPos = 'O';

State: board unchanged

Reason:

This test case is unique and distinct because it tests that whatsAtPos returns correctly when the board is full

Function Name:

test_whatsAtPos_full

public boolean isPlayerAtPos(BoardPosition pos, char player)

Input:

State:

	0	1	2
2			
1			
0			

pos.getRow = 0
pos.getColumn = 0 player = X

Output:

isPlayerAtPos = false;

State: board unchanged

Reason:

This test case is unique and distinct because it tests that isPlayerAtPos returns correctly when the board is empty

Function Name:

test_isPlayerAtPos_empty

Input:

State:

	0	1	2
2			
1			
0		X	

pos.getRow = 0
pos.getColumn = 0 player = X

Output:

isPlayerAtPos = false;

State: board unchanged

Reason:

This test case is unique and distinct because it tests that isPlayerAtPos returns correctly when the board has a character, but it's not at pos

Function Name:

test_isPlayerAtPos_charNotAtPos

Input:

State:

	0	1	2
2			
1			
0	X		

pos.getRow = 0
pos.getColumn = 0 player = X

Output:

isPlayerAtPos = true;

State: board unchanged

Reason:

This test case is unique and distinct because it tests that isPlayerAtPos returns correctly when the board has a character only at pos and it is player

Function Name:

test_isPlayerAtPos_charAtPos

Input:

State:

	0	1	2
2			
1			
0	O	X	O

pos.getRow = 0
pos.getColumn = 1 player = X

Output:

isPlayerAtPos = true;

State: board unchanged

Reason:

This test case is unique and distinct because it tests that whatsAtPos returns correctly when the board has a character at pos and other characters around it are not the player character but pos is the player character

Function Name:

test_isPlayerAtPos_charAtPosAndOthers

Input:

State:

	0	1	2
2			
1			
0	X		

pos.getRow = 0
pos.getColumn = 0 player = O

Output:

isPlayerAtPos = false;

State: board unchanged

Reason:

This test case is unique and distinct because it tests that whatsAtPos returns correctly when the board has a player at pos but it is not the player

Function Name:

test_isPlayerAtPos_wrongPlayer

public void placeToken(char p, int c)

Input:

State:

	0	1	2
2			
1			
0			

p = X c
= 0

Output:

State:

	0	1	2
2			
1			
0	X		

Reason:

This test case is unique and distinct because it tests that placeToken correctly places a token when the board is empty

Function Name:

test_placeToken_empty

Input:

State:

	0	1	2
2			
1			
0	O		

p = X c
= 0

Output:

State:

	0	1	2
2			
1	X		
0	O		

Reason:

This test case is unique and distinct because it tests that placeToken correctly places a token on top of another token

Function Name:

test_placeToken_onTop

Input:

State:

	0	1	2
2			
1	O		
0	X		

p = X c
= 0

Output:

State:

	0	1	2
2	X		
1	O		
0	X		

Reason:

This test case is unique and distinct because it tests that placeToken correctly places a token when the column is almost full

Function Name:

test_placeToken_columnAlmostFull

Input:

State:

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

p = X c
= 0

Output:

State:

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

Reason:

This test case is unique and distinct because it tests that placeToken correctly places a token when the board is almost full

Function Name:

test_placeToken_boardAlmostFull

Input:

State:

	0	1	2
2			
1			
0	X	O	

p = X c
= 2

Output:

State:

	0	1	2
2			
1			
0	X	O	X

Reason:

This test case is unique and distinct because it tests that placeToken correctly places a token when the board has two characters taken, but not in the same column

Function Name:

test_placeToken_notSameColumn