Le Nguyen
CPSC-2150
Project 3

REQUIREMENT ANALYSIS

Functional Requirements

As a user, I can input the position so that I can place my mark
As a user, I can input “y” so that I can play a new game
As a user, I can input “n” so that I can quit the game
As a user, I can input position again when the position is unavailable so that I can still make my move
As a user, I can input position again when the position is out of bound so that I can still make my move
As users, we can take turn to place marks so that we will have two player in the game
As users, we can fill all the game board with our marks so that we can make the game draw
As a user, I can see the board after each move so that I can keep track on the current game board
As a user, I can input the row so that I can set the number of rows of the game board
As a user, I can input the column so that I can set the number of columns of the game board
As a user, I can input the num to win so that I can set the number of consecutive points leading to win.
As a user, I can input number of player so that I can set how many player in the game
As a user, I can input type of game so that I can make game run in fast or memory effective way.
As a user, I can input the setting again so that the game run in the new setting.

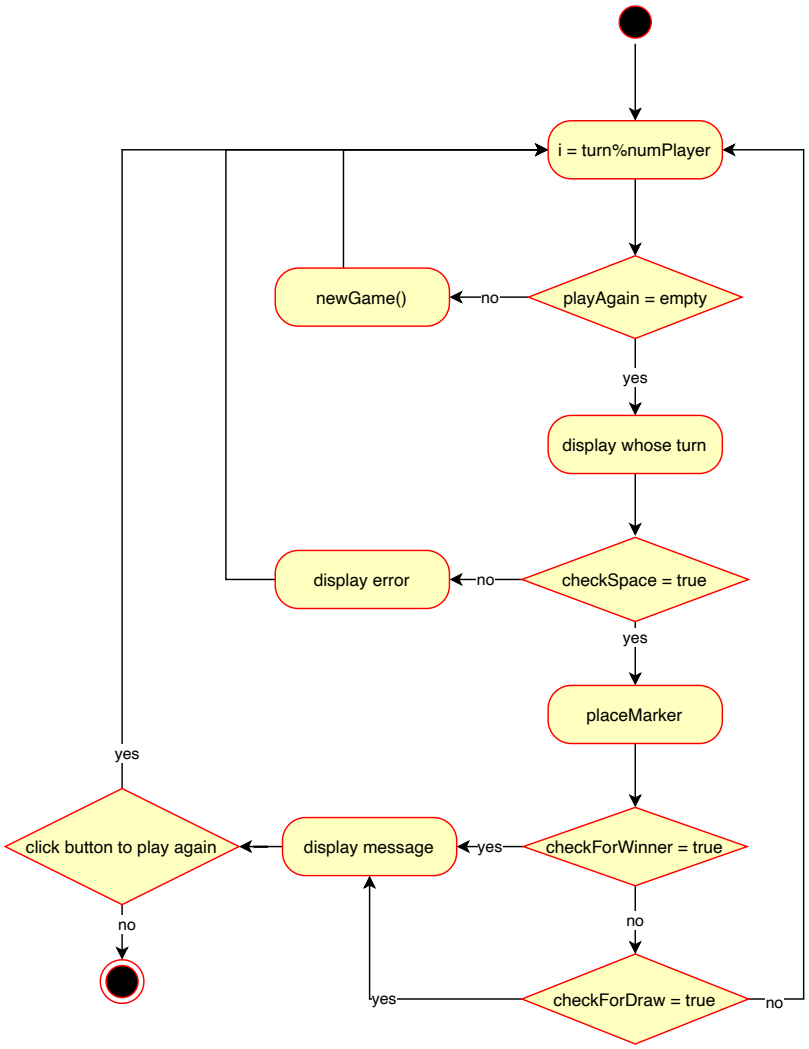
Non-functional Requirements

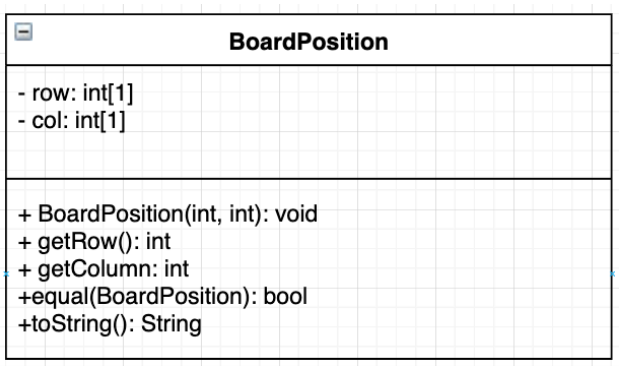
The board has to be printed in a minimal format.
The game takes less than 2 seconds to start.
The game must work on macOS, Linux Windows, Unix.
The program must be coded in Java.
The game board is of size of user’s choice.

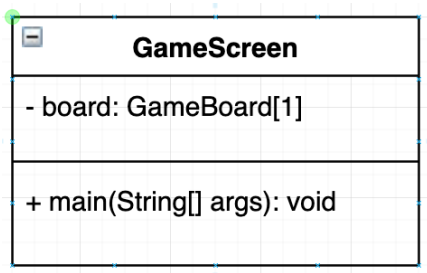
DESIGN

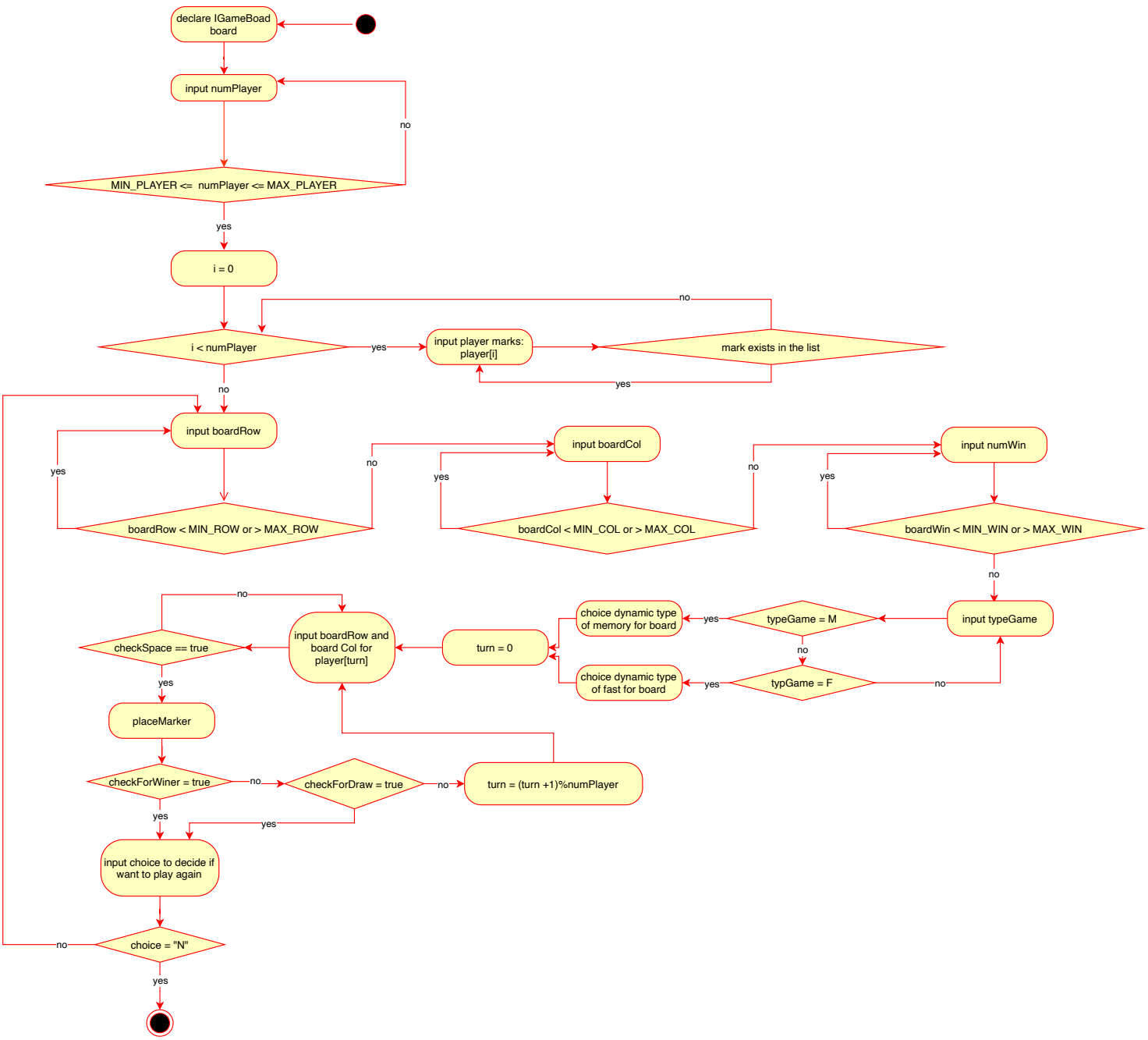
TicTacToeController
-curGame: IGameBoard -screen: TicTacToeView -numPlayers : int -player: Character[numPlayer] -MAX_PLAYER : int -turn:int -playAgain: String
+TicTacToeController(IGameBoard model, TicTacToe view, int np): void +processButtonClick(int row, int col): void -newGame(void): void

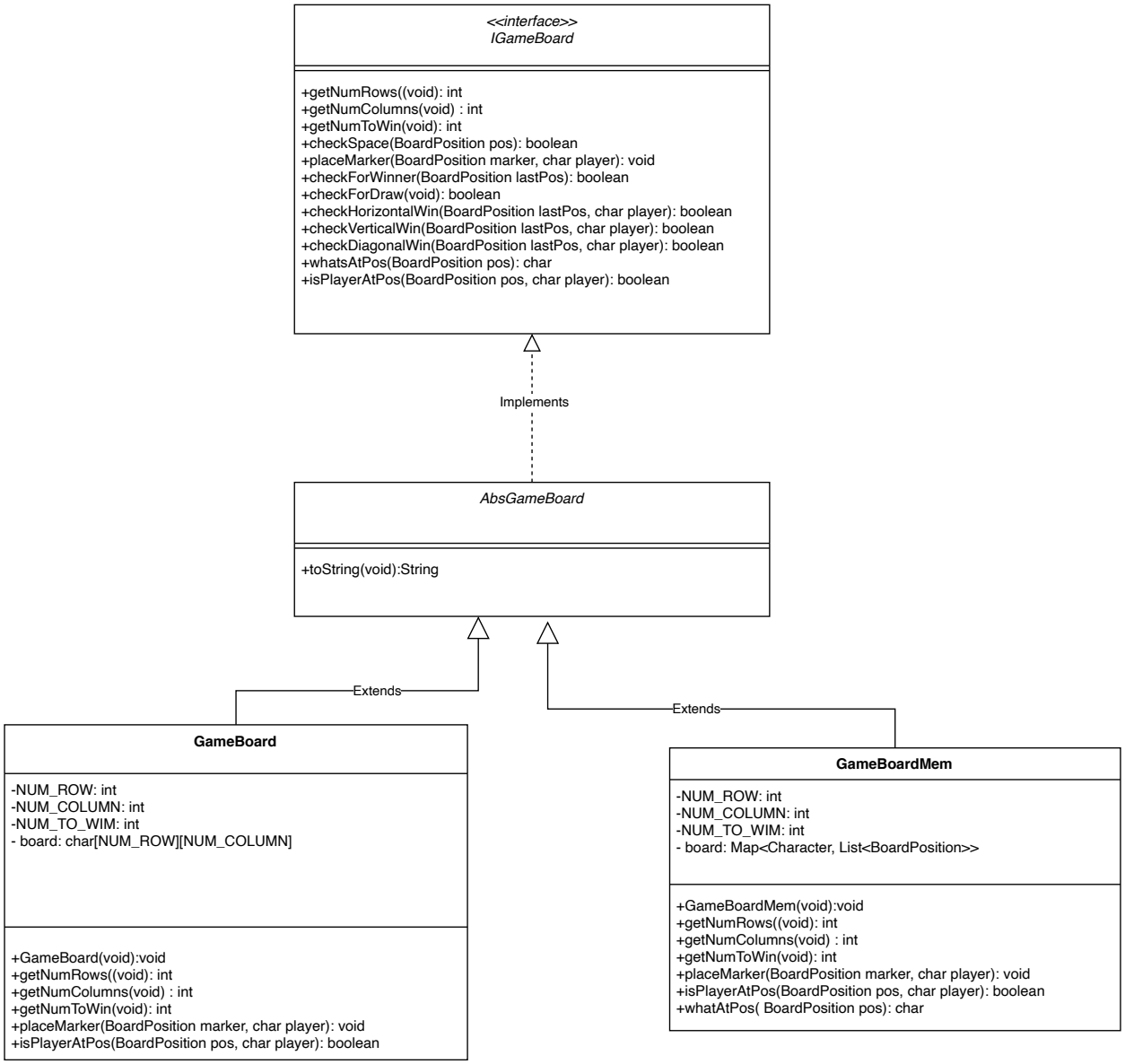
public void processButtonClick(int row, int col)




BoardPosition
- row: int[1]
- col: int[1]
+ BoardPosition(int, int): void
+ getRow(): int
+ getColumn: int
+equal(BoardPosition): bool
+toString(): String

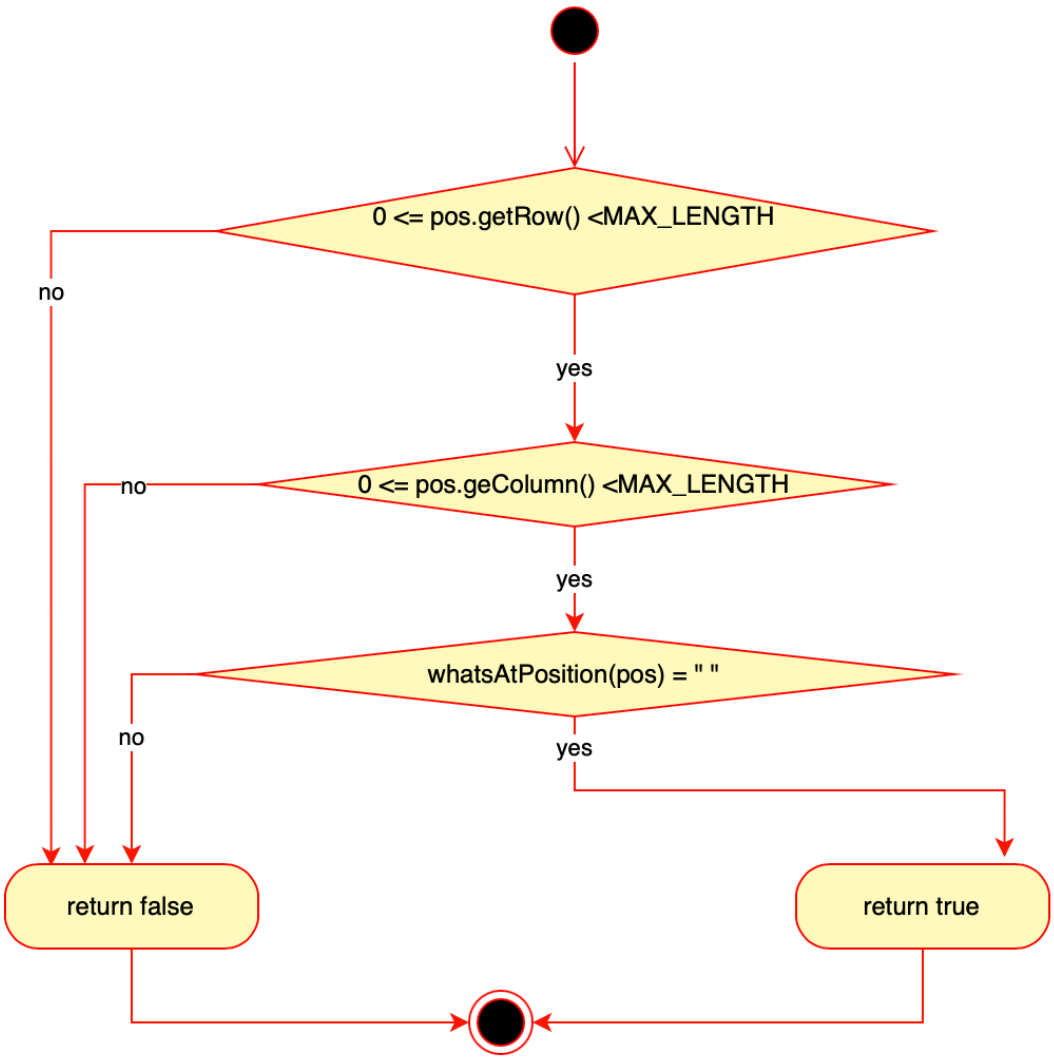

GameScreen
- board: GameBoard[1]
+ main(String[] args): void


declare IGameBoad board
input numPlayer
no
MIN_PLAYER <= numPlayer <= MAX_PLAYER
yes
i = 0
no
i < numPlayer
yes
input player marks: player[i]
mark exists in the list
yes
no
input boardRow
yes
boardRow < MIN_ROW or > MAX_ROW
no
input boardCol
yes
boardCol < MIN_COL or > MAX_COL
no
input numWin
yes
boardWin < MIN_WIN or > MAX_WIN
no
input typeGame
typeGame = M
yes
choice dynamic type of memory for board
no
typGame = F
yes
choice dynamic type of fast for board
no
turn = 0
input boardRow and board Col for player[turn]
checkSpace == true
no
yes
placeMarker
checkForWiner = true
no
checkForDraw = true
no
turn = (turn +1)%numPlayer
yes
yes
input choice to decide if want to play again
choice = "N"
no
yes

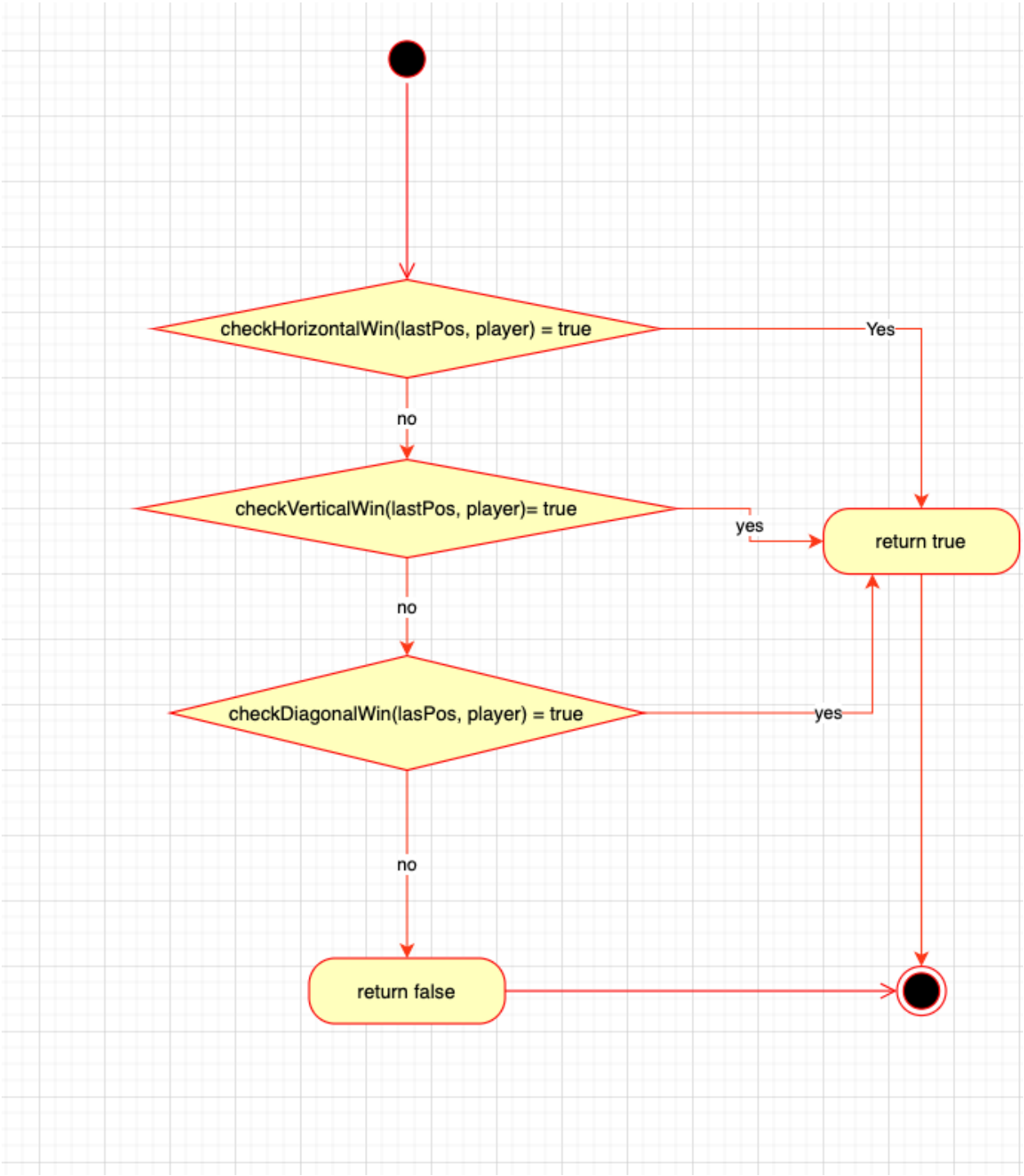

<<interface>>
IGameBoard
+getNumRows((void): int
+getNumColumns(void) : int
+getNumToWin(void): int
+checkSpace(BoardPosition pos): boolean
+placeMarker(BoardPosition marker, char player): void
+checkForWinner(BoardPosition lastPos): boolean
+checkForDraw(void): boolean
+checkHorizontalWin(BoardPosition lastPos, char player): boolean
+checkVerticalWin(BoardPosition lastPos, char player): boolean
+checkDiagonalWin(BoardPosition lastPos, char player): boolean
+whatsAtPos(BoardPosition pos): char
+isPlayerAtPos(BoardPosition pos, char player): boolean
Implements
AbsGameBoard
+toString(void):String
Extends
Extends
GameBoard
-NUM_ROW: int
-NUM_COLUMN: int
-NUM_TO_WIM: int
- board: char[NUM_ROW][NUM_COLUMN]
+GameBoard(void):void
+getNumRows((void): int
+getNumColumns(void) : int
+getNumToWin(void): int
+placeMarker(BoardPosition marker, char player): void
+isPlayerAtPos(BoardPosition pos, char player): boolean
GameBoardMem
-NUM_ROW: int
-NUM_COLUMN: int
-NUM_TO_WIM: int
- board: Map<Character, List<BoardPosition>>
+GameBoardMem(void):void
+getNumRows((void): int
+getNumColumns(void) : int
+getNumToWin(void): int
+placeMarker(BoardPosition marker, char player): void
+isPlayerAtPos(BoardPosition pos, char player): boolean
+whatAtPos(BoardPosition pos): char

Default Methods

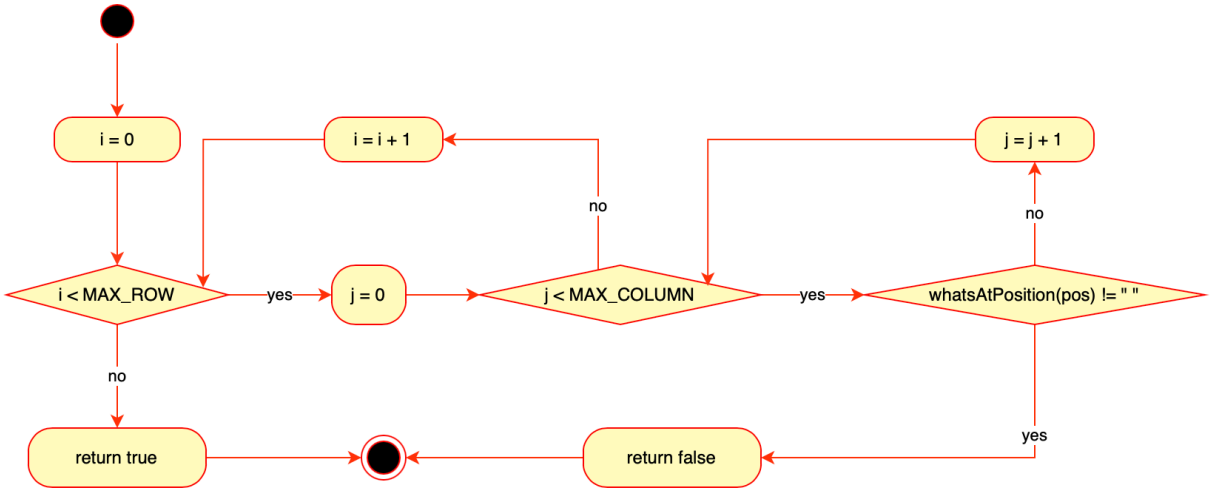
default boolean checkSpace(BoardPosition pos)



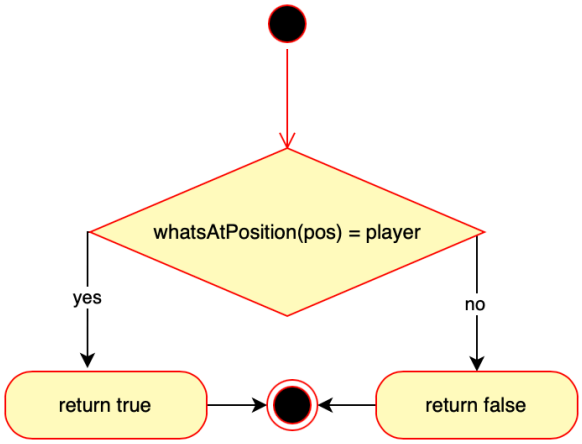
default boolean checkForWinner(BoardPosition lastPos)



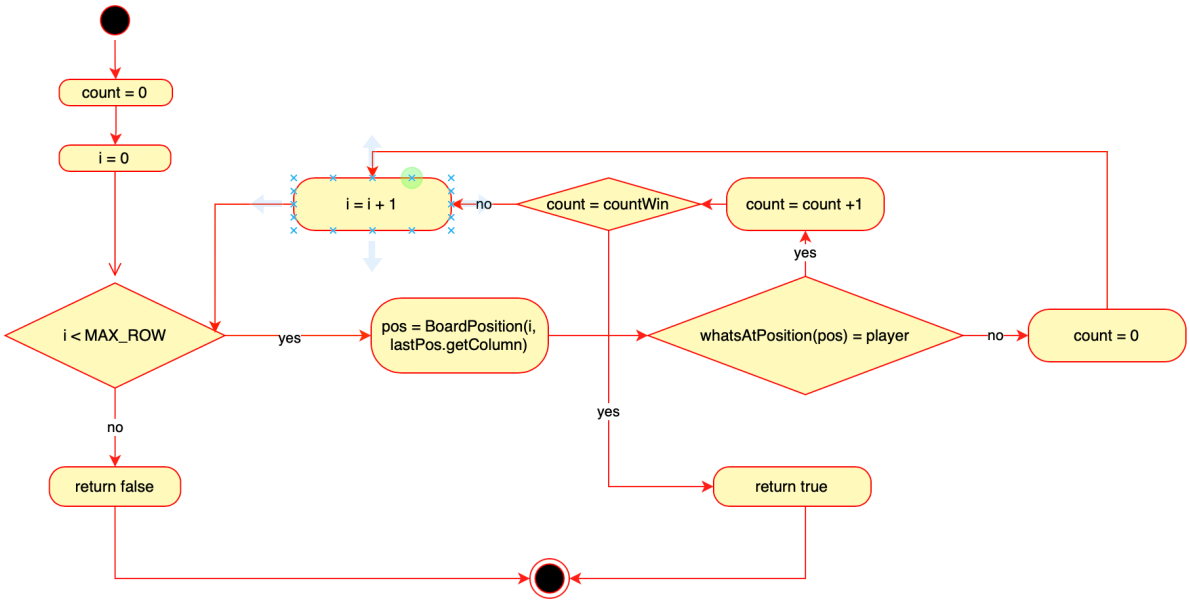
default boolean checkForDraw()



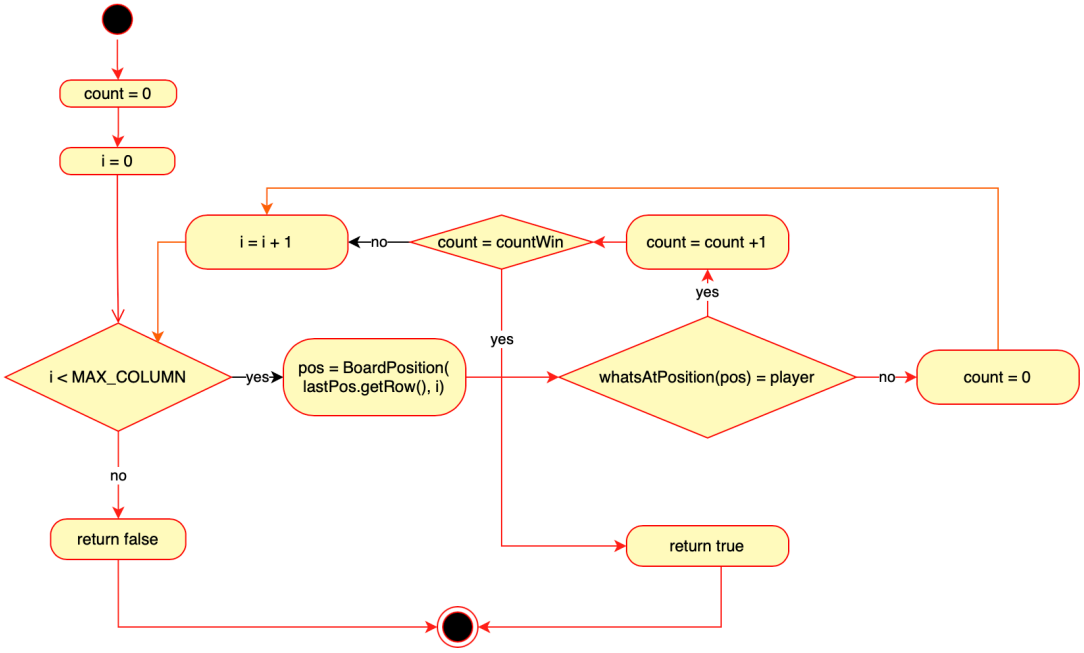
default boolean isPlayerAtPos(BoardPosition pos, char player)



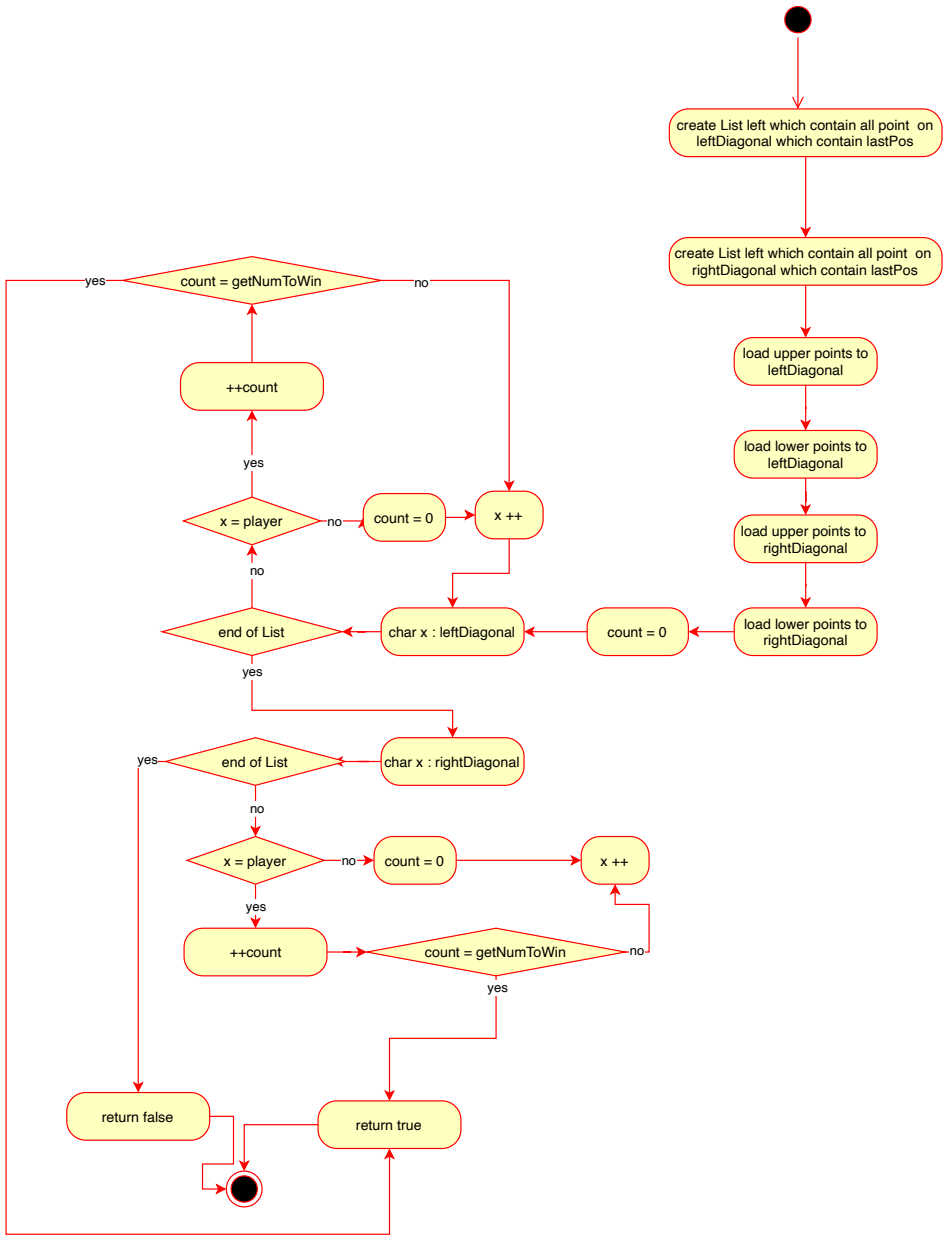
default boolean checkVerticalWin(BoardPosition lastPos, char player)



default boolean checkHorizontalWin(BoardPosition lastPos, char player)

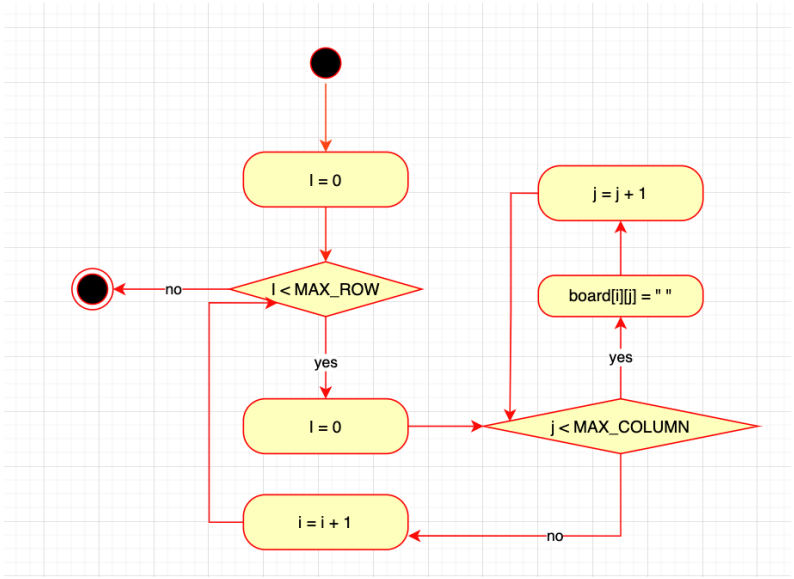


default boolean checkDiagonalWin(BoardPosition lastPos, char player)

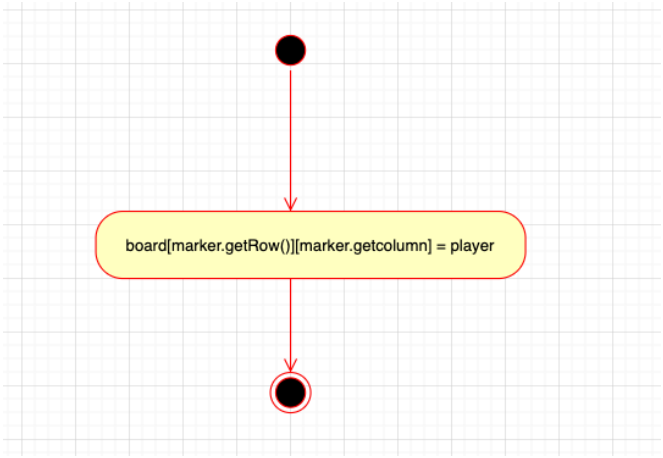


GameBoard class

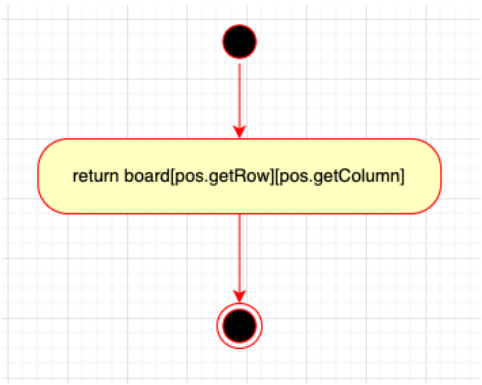
void GameBoard(void)



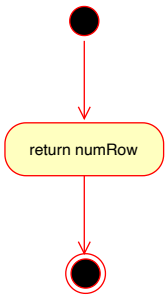
public void placeMarker(BoardPosition marker, char player)



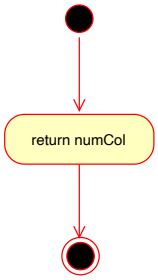
public char whatsAtPos(BoardPosition pos)



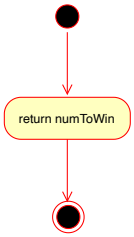
public int getNumRows()



public int getNumRows()

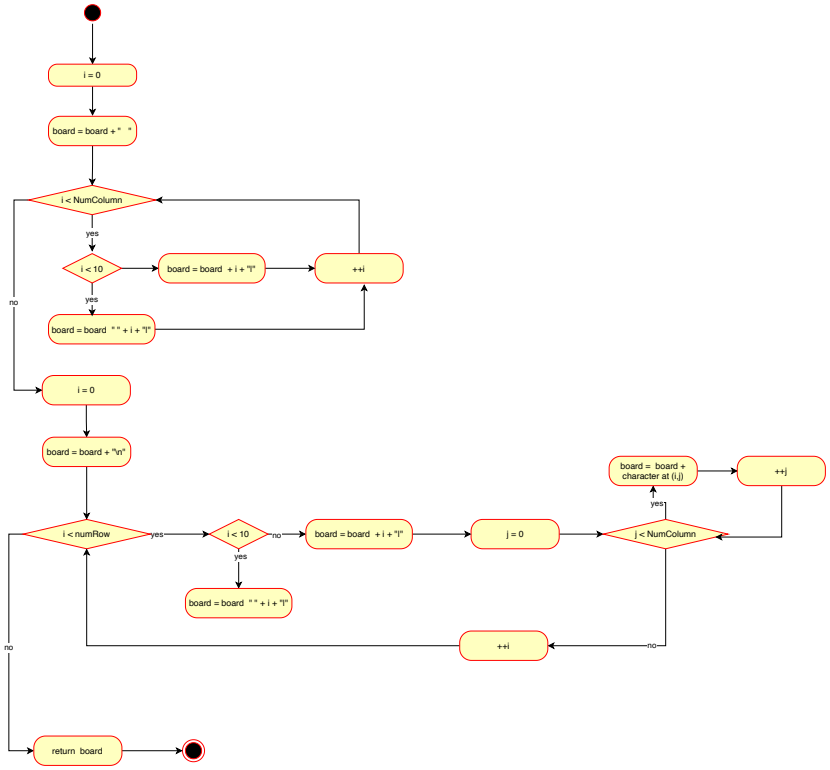


public int getNumToWin()



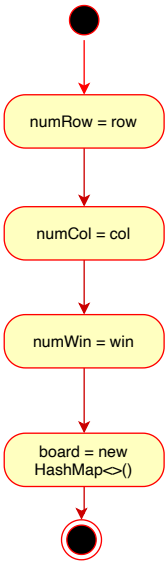
AbsGameBoard class

String toString()

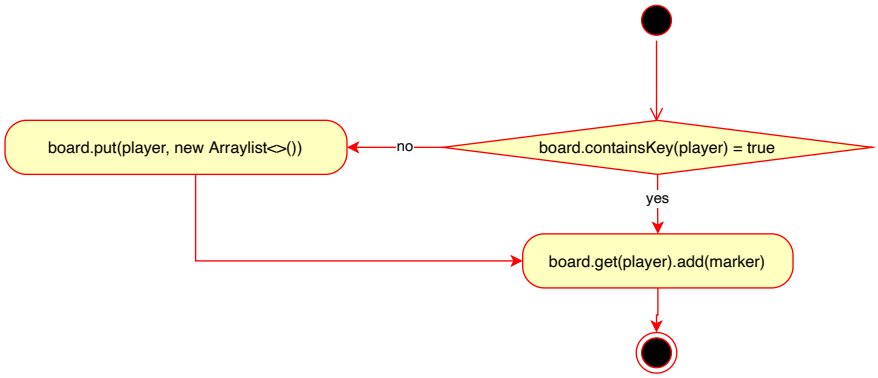


GameBoardMem class

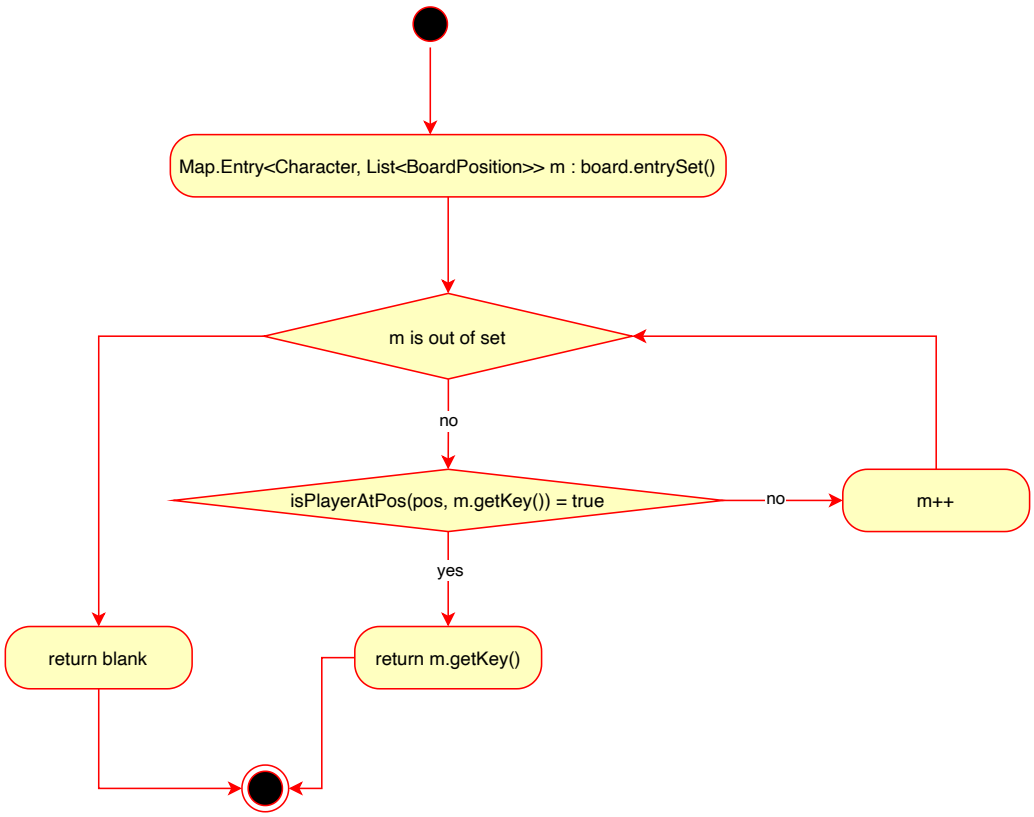
public GameBoardMem(int row, int col, int win)



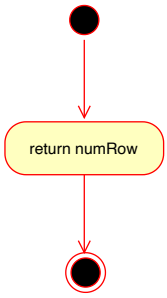
public void placeMarker(BoardPosition marker, char player)



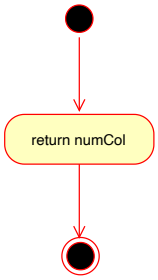
public char whatsAtPos(BoardPosition pos)



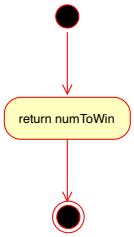
public int getNumRows()



public int getNumRows()



public int getNumToWin()



Test Cases

GameBoardMem(int row, int col, int win) and public GameBoard(int row, int col, int numWin)

Input	Output	Reanson

`row = 3, col =4,`
`numWin = 3`
	state:
	`  0| 1| 2| 3| 4|`
	`0|  |  |  |  |  |`
	`1|  |  |  |  |  |`
	`2|  |  |  |  |  |`
	`3|  |  |  |  |  |`
	getNumRow() = 3
		Test if constructor initializes number of row correctly
		
		Function Name:
		constructor_row

`row = 3, col =4,`
`numWin = 3`
	state:
	`  0| 1| 2| 3| 4|`
	`0|  |  |  |  |  |`
	`1|  |  |  |  |  |`
	`2|  |  |  |  |  |`
	`3|  |  |  |  |  |`
	getNumColumn() = 4
		Reason: Test if constructor initializes number of column correctly
		
		Function Name
		constructor_column

`row = 3, col =4,`
`numWin = 3`
	state:
	`  0| 1| 2| 3| 4|`
	`0|  |  |  |  |  |`
	`1|  |  |  |  |  |`
	`2|  |  |  |  |  |`
	`3|  |  |  |  |  |`
	getNumWin() = 3
		Reason: Test if constructor initializes number of numWin correctly
		
		Function Name
		constructor_rnumWin

default boolean checkSpace(BoardPosition pos)

Input	Output	Reanson

`  0| 1| 2| 3|`
`0|  |  |  |  |`
`1|  |  |  |  |`
`2|  |  |  |  |`
`3|  |  |  |  |`
 	
`pos.getRow = 6`
`pos.getColumn = 2`
	checkSpace = false
	
	State of the board in unchanged
		Position is out of the board
		
		Function Name
		
		checkSpace_row_out_of_bound

`  0| 1| 2| 3|`
`0|  |  |  |  |`
`1|  |  |  |  |`
`2|  |  |  |  |`
`3|  |  |  |  |`
`pos.getRow = 2`
`pos.getColumn = 1`
	checkSpace = true
	
	State of the board in unchanged
		Position is inside the board
		
		Function Name
		NamecheckSpace_inBound

`  0| 1| 2| 3|`
`0|  |  |  |  |`
`1|  |  |  |  |`
`2|  |X |  |  |`
`3|  |  |  |  |`
`pos.getRow = 2`
`pos.getColumn = 1`
	checkSpace = false
	
	State of the board in unchanged
		Position in inside the board but there exits another marker already
		
		Function Name
		checkSpace_inValid_position

default boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input	Output	Reanson

State: (numWin = 4) 0| 1| 2| 3| 4| 0| | | | | | 1|X |X |X |X | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 2 player = ‘X’	checkHorizontalWin = true State is unchanged	last marker in the middle, so it should scan left and right **Function Name** checkHorizontalWin_win_last_marker_middle
State: (numWin = 4) 0| 1| 2| 3| 4| 0|X |X |X | | | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 2 player = ‘X’	checkHorizontalWin = false State is unchanged	Check number of consecutive marks less than numWin **Function Name** checkHorizontalWin_not_enough_mark
State: (numWin = 4) 0| 1| 2| 3| 4| 0| | | | | | 1|X |X |X |X |X | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 2 player = ‘X’	checkHorizontalWin = true State is unchanged	the last position in the middle will make the number of consecutive marks larger than numWin. It still output true even the value is larger than numWin **Function Name** checkHorizontalWin_total_marker_larger_than_numWin
State: (numWin = 4) 0| 1| 2| 3| 4| 0| | | | | | 1|X |X |O |X |X | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 4 player = ‘X’	checkHorizontalWin = false State of the board in unchanged	There are enough number of consecutive marks, but still can not win because there is a different mark in the middle **Function Name** checkHorizontalWin_not_consecutive

default boolean checkVerticalWin(BoardPosition lastPos, char player)

Input	Output	Reanson
State: (numWin = 3) 0| 1| 2| 3| 4| 0| |X | | | | 1| |X | | | | 2| |X | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 1 player = ‘X’	checkVerticalWin = true State is unchanged	the last marker is in the middle, so it should scan both left and right **Function Name** checkVerticalWin_win_last_marker_middle
State: (numWin = 3) 0| 1| 2| 3| 4| 0| |X | | | | 1| |X | | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 1 player = ‘X’	checkVerticalWin = false State is unchanged	Check number of consecutive marks less than numWin **Function Name** checkVerticalWin_not_enough_mark
State: (numWin = 3) 0| 1| 2| 3| 4| 0| |X | | | | 1| |X | | | | 2| |X | | | | 3| |X | | | | 4| | | | | | pos.getRow = 2 pos.getColumn = 1 player = ‘X’	checkVerticalWin = true State is unchanged	the last position will make the number of consecutive marks larger than numWin. It still output true even the value is larger than numWin **Function Name** checkVerticalWin_total_marker_larger_than_numWin

State: (numWin = 3) 0| 1| 2| 3| 4| 0| |X | | | | 1| |X | | | | 2| |O | | | | 3| |X | | | | 4| | | | | | pos.getRow = 3 pos.getColumn = 0 player = 'X'	checkVerticalWin = false State of the board in unchanged	There are enough number of consecutive marks, but still can not win because there is a different mark in the middle **Function Name** checkVerticalWin_not_consecutive

default boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reanson
State: (numWin = 3) 0| 1| 2| 3| 4| 0| |X | | | | 1| | |X | | | 2| | | |X | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 2 player = 'X'	checkDiagonalWin = true State is unchanged	Check left diagonal. The last marker is in the middle, so it should scan both bottom and top **Function Name** checkDiagonalWin_left_last_marker_middle
State: (numWin = 3) 0| 1| 2| 3| 4| 0| |X | | | | 1| | |X | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 2 player = 'X'	checkDiagonalWin = false State is unchanged	Left diagonal: check number of consecutive marks less than numWin **Function Name** checkDiagonalWin_left_not_enough_marks

State: (numWin = 3) 0| 1| 2| 3| 4| 0|X | | | | 1| |O | | | 2| | |X | | 3| | | |X | 4| | | | | pos.getRow = 3 pos.getColumn = 3 player = ‘X’	checkDiagonalWin = false State is unchanged	Left diagonal: There are enough number of consecutive marks, but still can not win because there is a different mark in the middle **Function Name** checkDiagonalWin_left_another_mark_in_middle
State: (numWin = 3) 0| 1| 2| 3| 4| 0| | |X | | | 1| |X | | | | 2|X | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 1 player = ‘X’	checkDiagonalWin = true State of the board in unchanged	Check right diagonal. The last marker is in the middle, so it should scan both bottom and top **Function Name** checkDiagonalWin_right_last_marker_middle
State: (numWin = 3) 0| 1| 2| 3| 4| 0| | |X | | | 1| | X | | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 1 player = ‘X’	checkDiagonalWin = false State of the board in unchanged	Check right diagonal. Check number of consecutive marks less than numWin **Function Name** checkDiagonalWin_right_not_enough_marks
State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | |X | | 1| | |O | | | 2| |X | | | | 3|X | | | | | 4| | | | | | pos.getRow = 3 pos.getColumn = 0 player = ‘X’	checkDiagonalWin = false State of the board in unchanged	Check right diagonal. There are enough number of consecutive marks, but still can not win because there is a different mark in the middle **Function Name** checkDiagonalWin_right_another_mark_in_middle

State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | |X | | 1| | |X | | | 2| |X | | | | 3|X | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 2 player = 'X'	checkDiagonalWin = true State of the board in unchanged	Check right diagonal. the last marker will make the number of consecutive marks larger than numWin. It still output true even the value is larger than numWin **Function Name** checkDiagonalWin_right_numMarks_larger_than_numWin

default boolean checkForDraw()

Input	Output	Reanson
State: (numWin = 3) 0| 1| 2| 3| 4| 0|A |B |C |D |E | 1|E |D |C |B |A | 2|C |A |B |D |E | 3|A |B |C |D |E | 4|E |C |D |A |D |	checkForDraw = true State is unchanged	check for full board **Function Name** checkForDraw_full_board
State: (numWin = 3) 0| 1| 2| 3| 4| 0|A |B | |D |E | 1|E | |C |B |A | 2|C |A |B |D | | | 3|A | |C |D |E | 4|E |C |D | |D |	checkForDraw = false State is unchanged	check for the board is not full yet. the function should catch the empty space. **Function Name** checkForDraw_not_full_board
State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | | | | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | | |	checkForDraw = false State is unchanged	check for the empty board.The function is still able to scan for an empty character even though there is not any mark placed yet **Function Name** checkForDraw_empty_board

State: (numWin = 3) 0| 1| 2| 3| 4| 0|A |B |C |D |E | 1|E |D |C |B |A | 2|C |A |B |D |E | 3|A |B |C |D |D | 4|E |C |D |A | |	checkForDraw = false State of the board in unchanged	There is only one space left in the last position. the function should be able to scan to the last position to get that empty space **Function Name** checkForDraw_only_one_space_left_in_last_position

public char whatsAtPos(BoardPosition pos)

Input	Output	Reanson
State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | | | | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 1	whatsAtPos = ‘ ‘ State is unchanged	check the empty space **Function Name** whatsAtPos_empty_board
State: (numWin = 3) 0| 1| 2| 3| 4| 0|A |B | |D |E | 1|E | |C |B |A | 2|C |A |B |D | | 3|A | |C |D |E | 4|E |C |D | |D | pos.getRow = 1 pos.getColumn = 1	whatsAtPos= ‘ ‘ State is unchanged	check if function still recognizes the empty space after placing some marker **Function Name** whatsAtPos_empty_character_with_some_existing_characters

State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | | | | 1| |X | | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 1 pos.getColumn = 1	whatsAtPos = ‘X’ State is unchanged	check what is at a position right after placing a marker at that position **Function Name** whatsAtPos_after_place_a_marker
State: (numWin = 3) 0| 1| 2| 3| 4| 0|A |B | |D |E | 1|E | |C |B |A | 2|C |A |B |D | | 3|A | |C |D |E | 4|E |C |D | |D | pos.getRow = 0 pos.getColumn = 0	whatsAtPos = ‘A’ State of the board in unchanged	After placing a lot of character, check if the function still can get the character at the beginning **Function Name** whatsAtPos_previous_position
State: (numWin = 3) 0| 1| 2| 3| 4| 0|A |B |C |D |E | 1|E |D |C |B |A | 2|C |A |B |D |E | 3|A |B |C |D |E | 4|E |C |D |A |D | pos.getRow = 1 pos.getColumn = 1	whatsAtPos = player[i][j] or whatAtPos != ‘ ‘ State of the board in unchanged	use 2D for loop to scan board, the function should not catch any empty space. **Function Name** whatsAtPos_full_board

public void placeMarker (BoardPosition marker, char player)

Input	Output	Reanson

State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | | | | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 0 pos.getColumn = 0 player = 'X'	State 0| 1| 2| 3| 4| 0|X | | | | | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | | |	place marker to the top left conner to test the bound **Function Name** placeMarker_top_left
State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | | | | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 0 pos.getColumn = 4 player = 'X'	State 0| 1| 2| 3| 4| 0| | | | |X | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | | |	place marker to the top right conner to test the bound **Function Name** placeMarker_top_right
State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | | | | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 4 pos.getColumn = 0 player = 'X'	State 0| 1| 2| 3| 4| 0| | | | | | 1| | | | | | 2| | | | | | 3| | | | | | 4|X | | | | |	place marker to the bottom left conner to test the bound **Function Name** placeMarker_bottom_left
State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | | | | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | | | pos.getRow = 4 pos.getColumn = 4 player = 'X'	State 0| 1| 2| 3| 4| 0| | | | | | 1| | | | | | 2| | | | | | 3| | | | | | 4| | | | |X |	place marker to the bottom right conner to test the bound **Function Name** placeMarker_bottom_right

State: (numWin = 3) 0| 1| 2| 3| 4| 0| | | | | | 1| | |Y | | | 2| | | | | | 3| | | |X | | 4| | | | | | pos.getRow = 3 pos.getColumn = 2 player = ‘X’	State 0| 1| 2| 3| 4| 0| | | | | | 1| | |Y | | | 2| | | | | | 3| | |X |X | | 4| | | | | |	try to place a marker second time. **Function Name** placeMarker_second_time