Project 5

Jack Adams

11/29/2021

Jack Adams

Requirements Analysis

Functional Requirements

As a player, I get to choose the number of rows, columns, players, and number to win, so that the game specifications are to my liking.

As a player, I am notified if any of my selections in starting the game are invalid, so that I can enter valid game specifications.

As a player, I am notified of whose turn it is at all times during an active game, so that I know which player's turn it is.

As a player, I can select the cell of my marker, so that my marker is placed on the board.

As a player, I am notified of an invalid selection and given the chance to enter again, so that I may make a valid selection.

As a player, I can view the updated board after each move, so that I can see the current status of the game.

As a player, I am notified when the game is over, notified of the result, and the shown the resulting board, so that I know the game's result.

As a player, I can choose to play again, so that I can play a new game if desired.

As a player, I can again choose new game specifications at the start of a new game, so that I can play the game with different specifications from the previous game.

Non-Functional Requirements

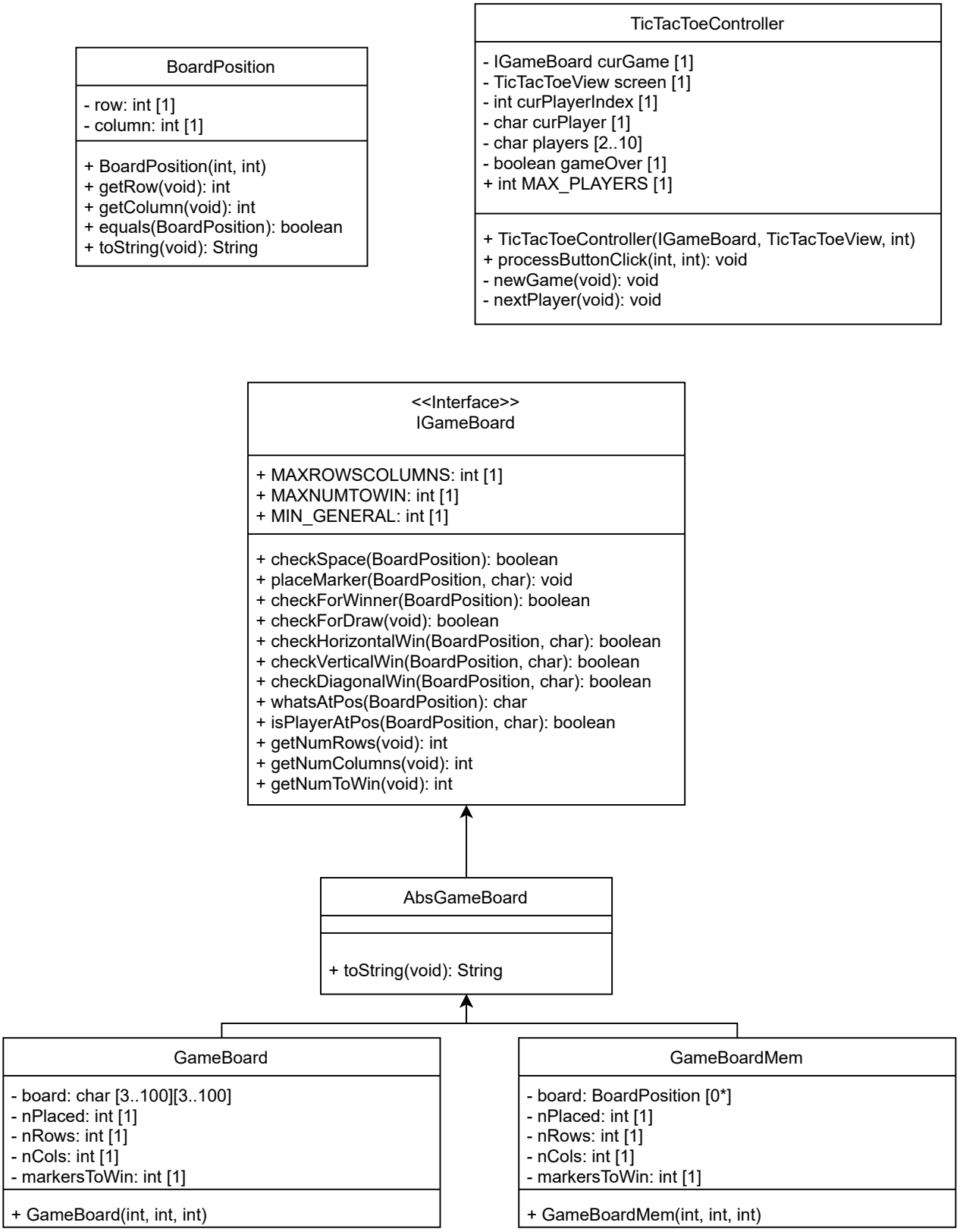
A pop-up window acts as the interface for gameplay.

The program must compile with Java JDK 11.

The program must run on IntelliJ by running the TicTacToeGame file.

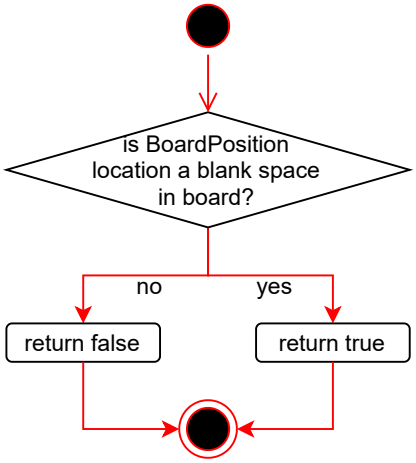
Jack Adams

Class Diagrams

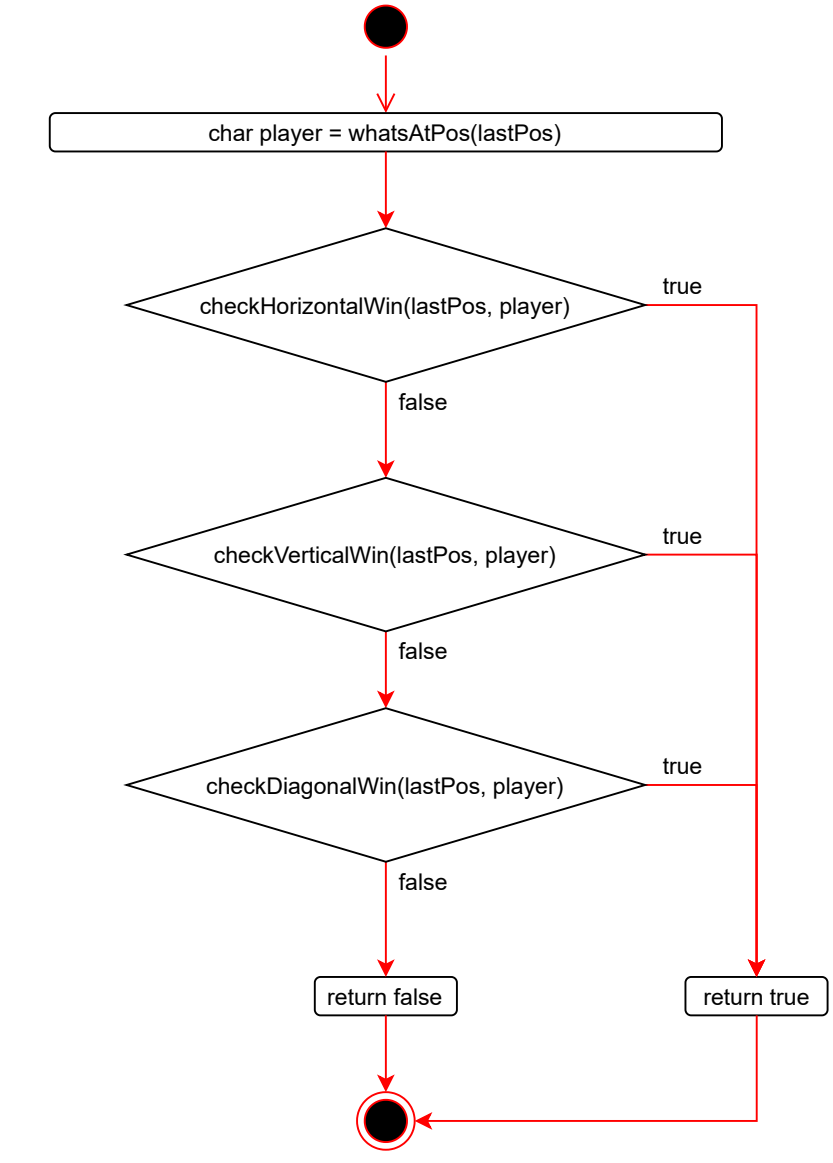


IGameBoard

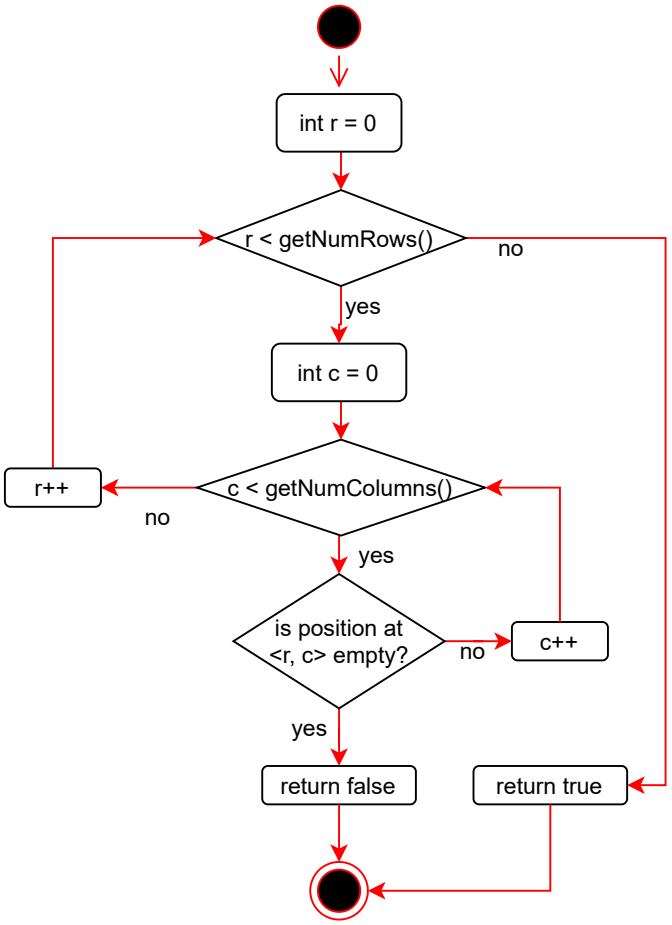
public boolean checkSpace(BoardPosition pos)



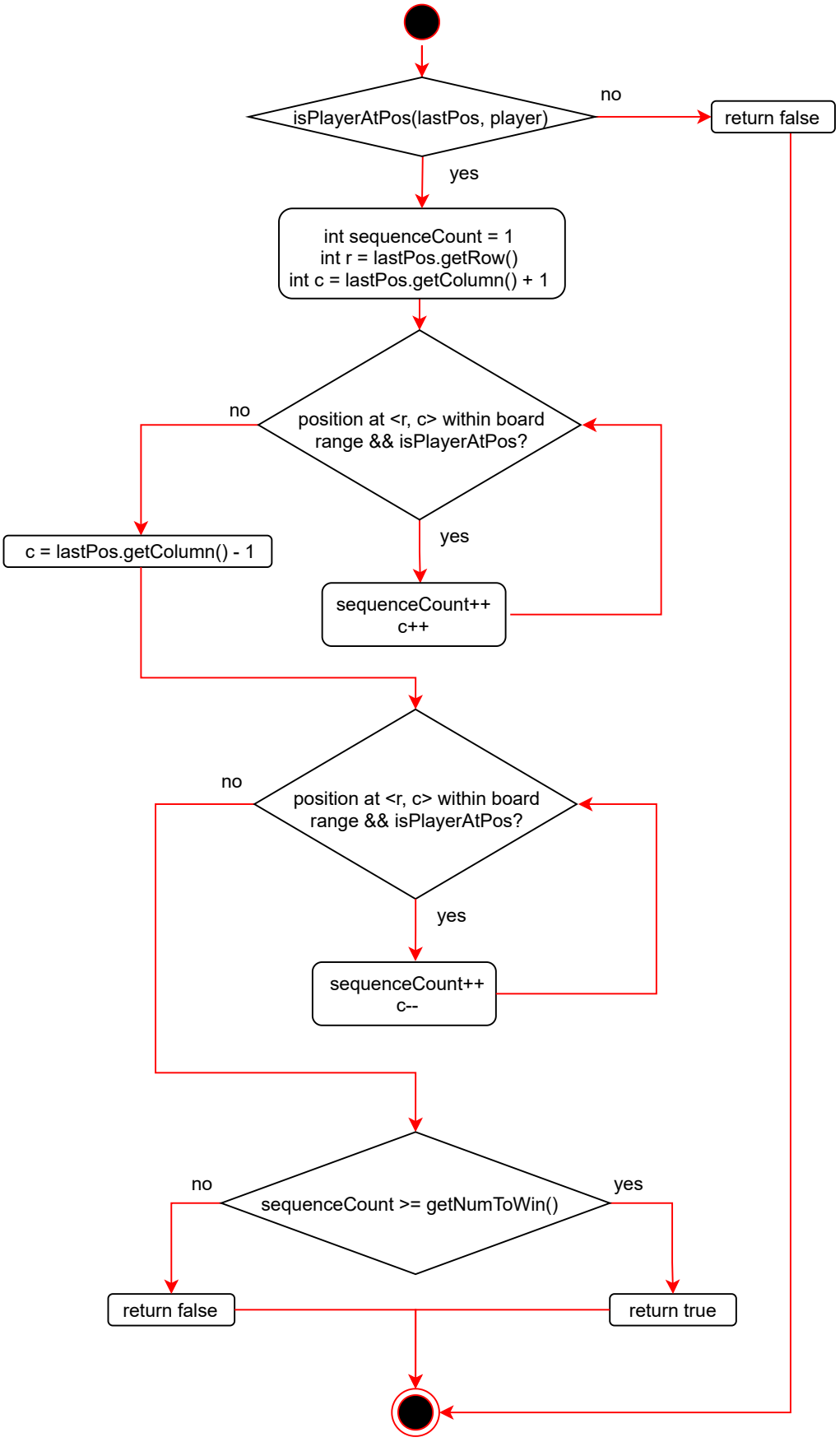
public boolean checkForWinner(BoardPosition lastPos)



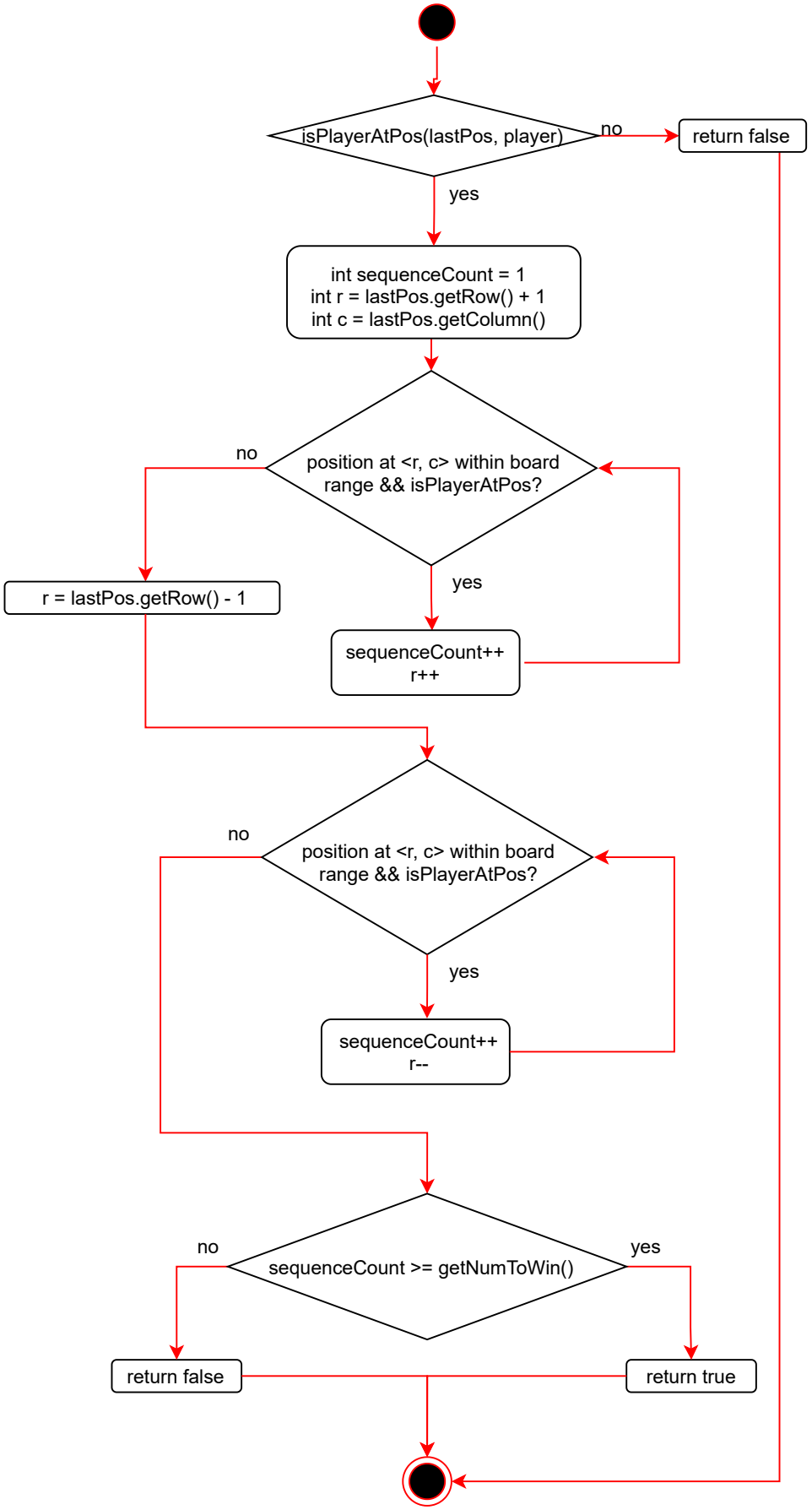
public boolean checkForDraw()



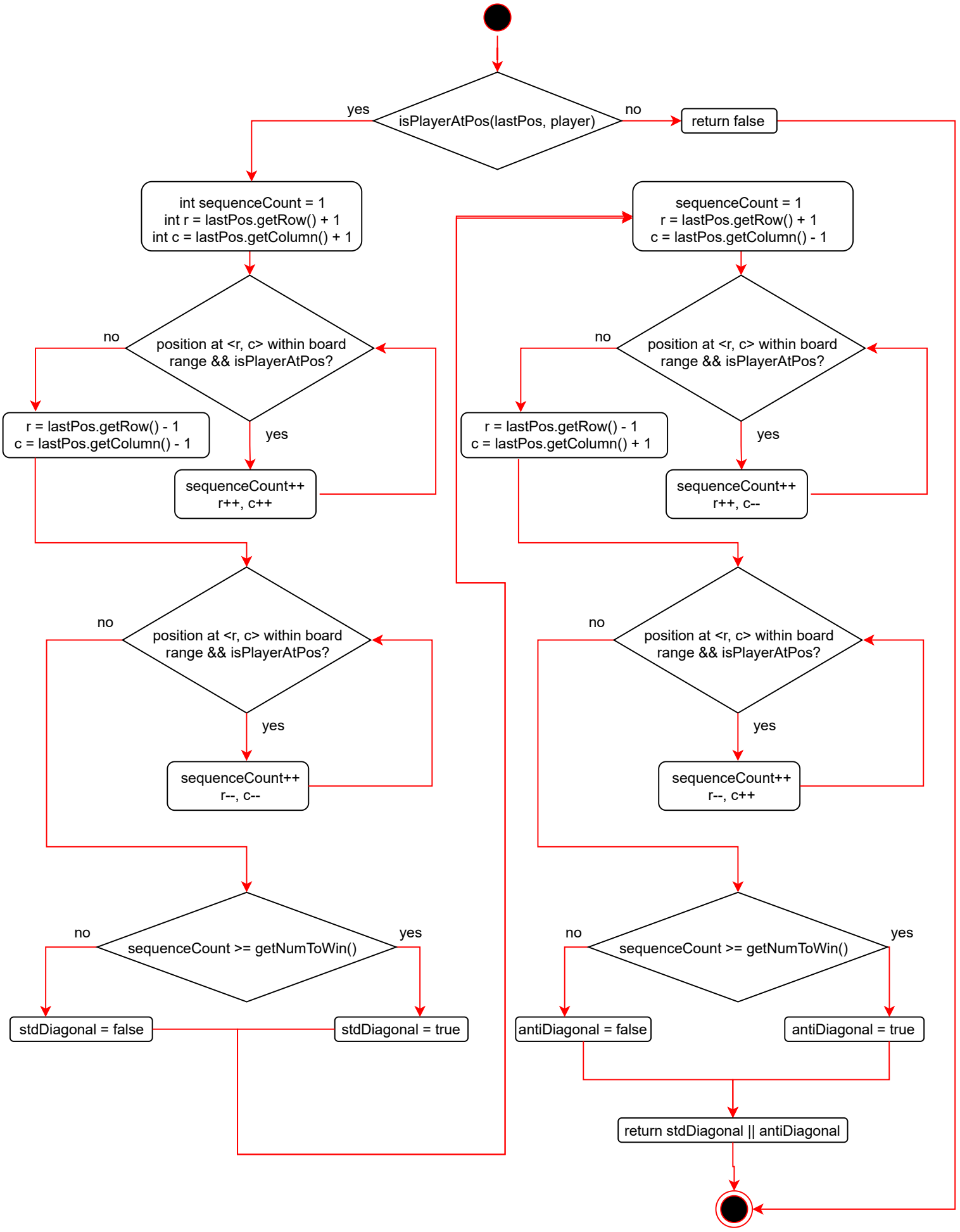
public boolean checkHorizontalWin(BoardPosition lastPos, char player)



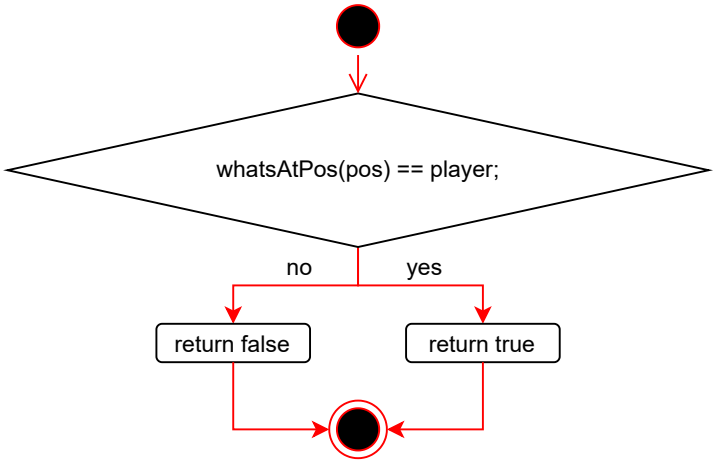
public boolean checkVerticalWin(BoardPosition lastPos, char player)



public boolean checkDiagonalWin(BoardPosition lastPos, char player)

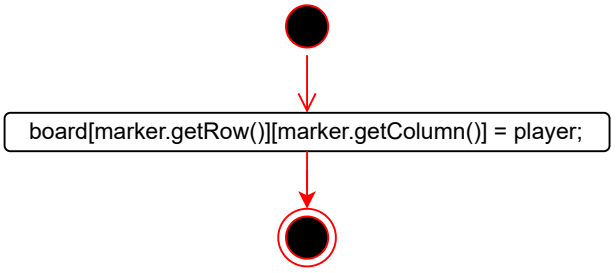


public boolean isPlayerAtPos(BoardPosition pos, char player)

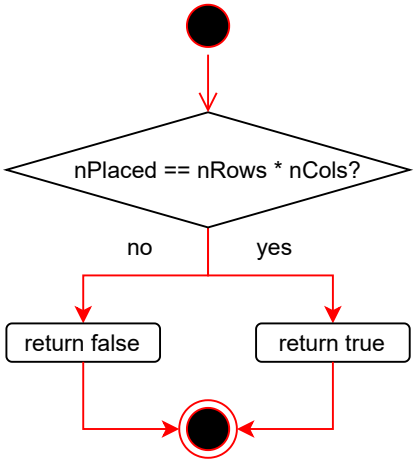


GameBoard

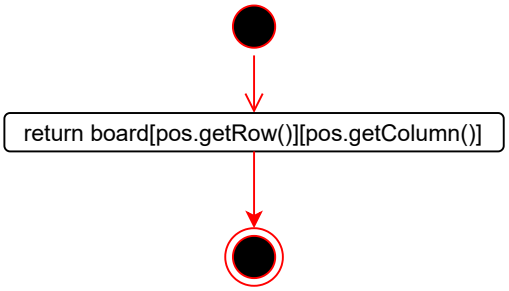
public void placeMarker(BoardPosition marker, char player)



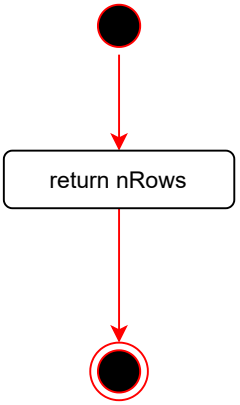
public boolean checkForDraw()



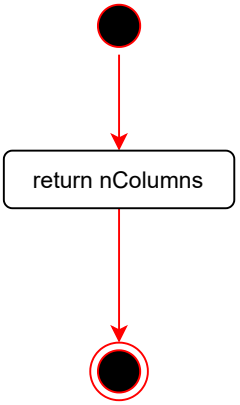
public char whatsAtPos(BoardPosition pos)



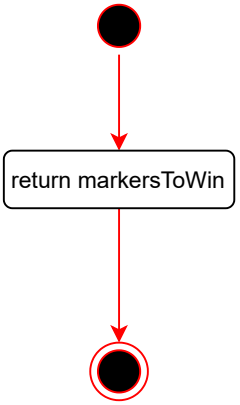
public int getNumRows()



public int getNumColumns()

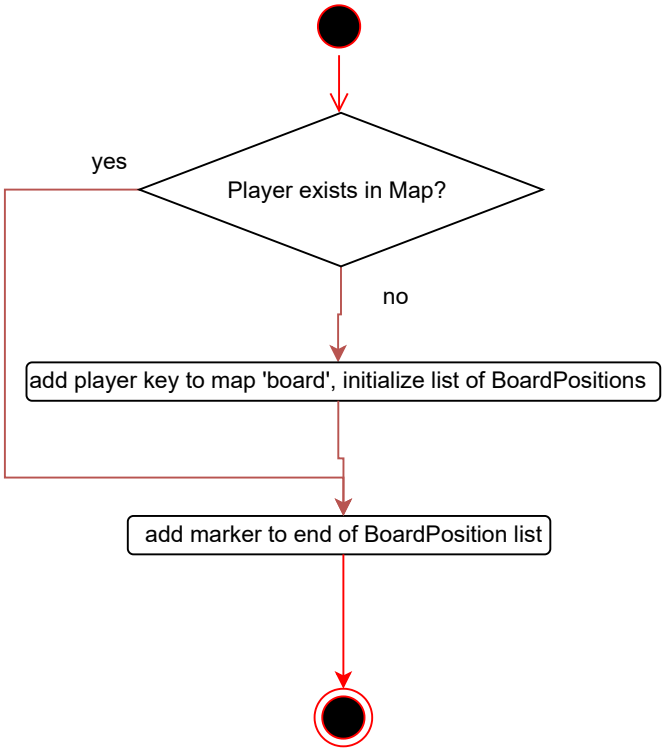


public int getNumToWin()

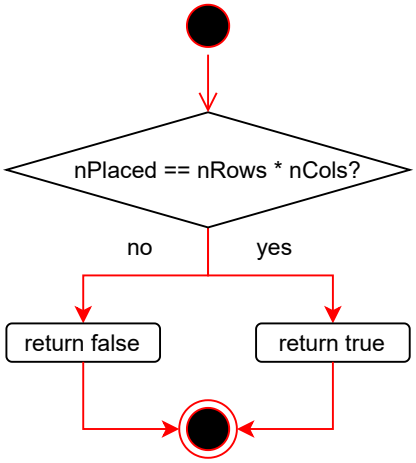


GameBoardMem

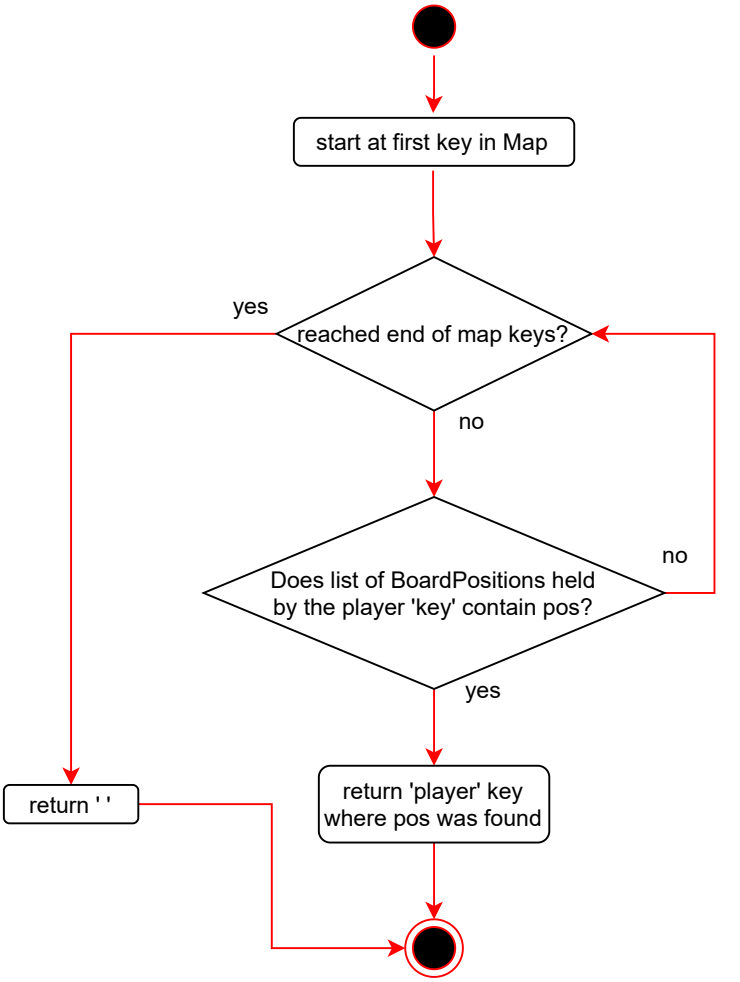
public void placeMarker(BoardPosition marker, char player)



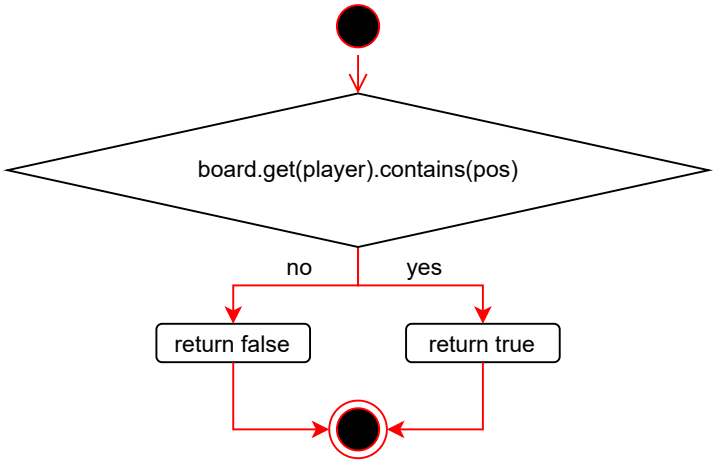
public boolean checkForDraw()



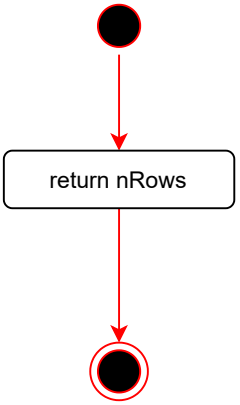
public char whatsAtPos(BoardPosition pos)



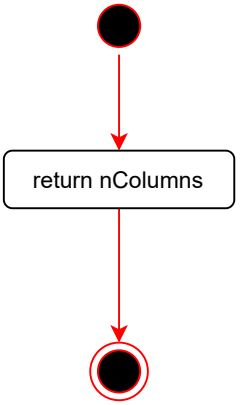
public boolean isPlayerAtPos(BoardPosition pos, char player)



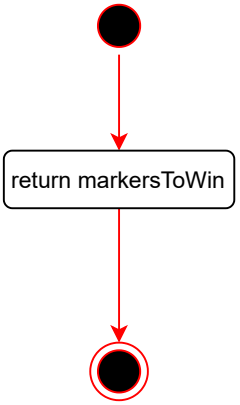
public int getNumRows()



public int getNumColumns()

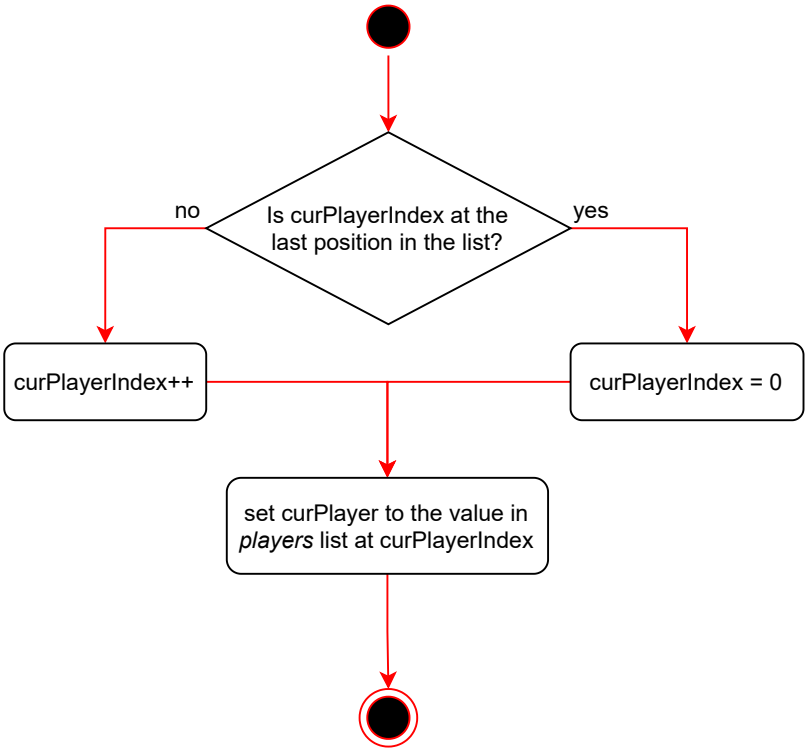


public int getNumToWin()

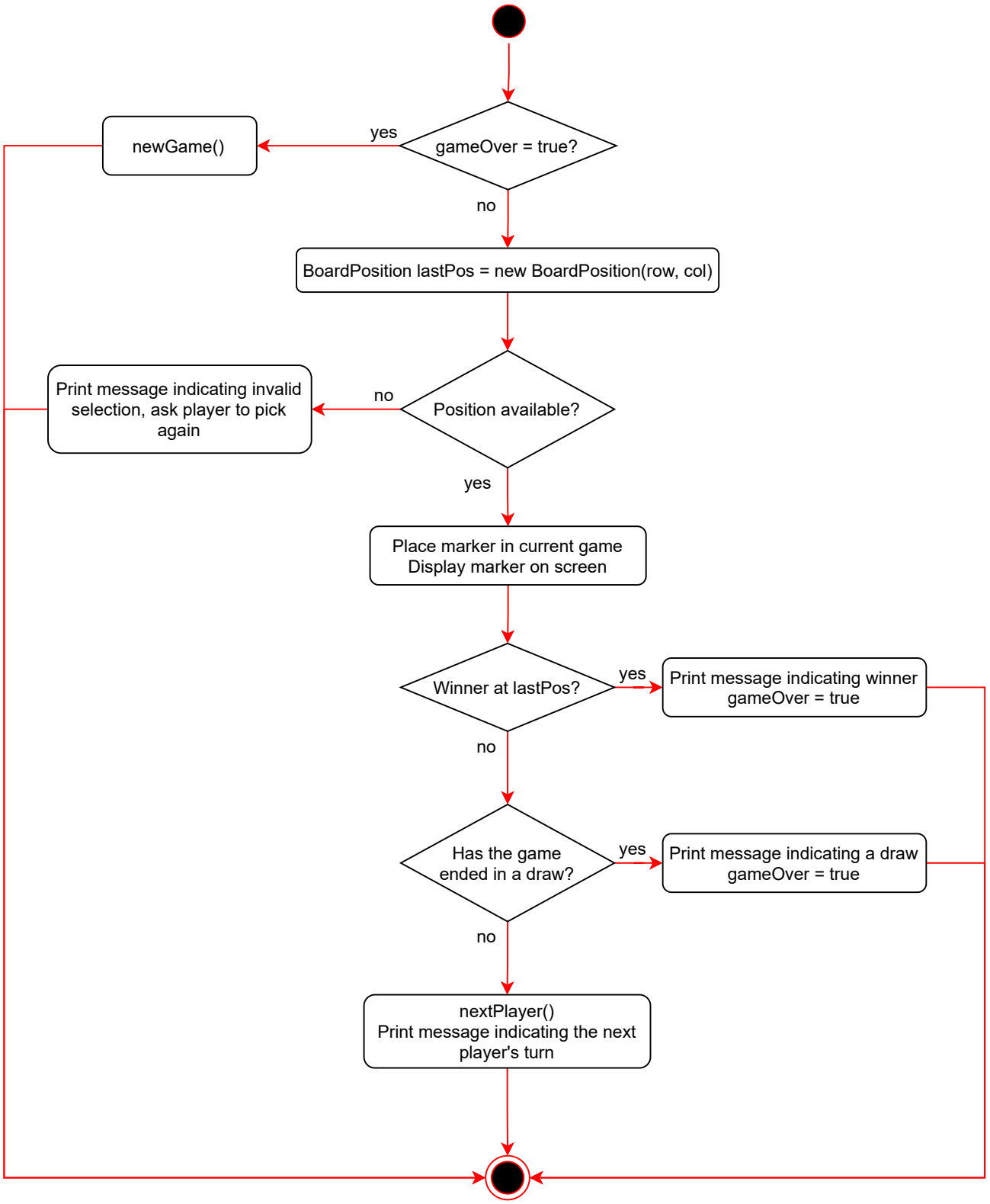


TicTacToeController

private void nextPlayer()



public void processButtonClick(int row, int col)



Testing

Constructor(int r, int c, int m) [GameBoard/GameBoardMem]

Input: State: r = 3 c = 3 m = 3 board is uninitialized	**Output:** State: (number to win = 3) <table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr></table>	**Reason:** This test case is unique and distinct because the GameBoard constructor is called using the minimum allowable values for all parameters. **Function Name:** testConstructor_min

Constructor(int r, int c, int m) [GameBoard/GameBoardMem]

Input: State: r = 100 c = 100 m = 25 board is uninitialized	**Output:** State: (number to win = 3) [empty board with 100 rows, 100 cols]	**Reason:** This test case is unique and distinct because the GameBoard constructor is called using the maximum allowable values for all parameters. **Function Name:** testConstructor_max

Constructor(int r, int c, int m) [GameBoard/GameBoardMem]

Input: State: r = 45 c = 82 m = 34 board is uninitialized	**Output:** State: (number to win = 3) [empty board with 100 rows, 100 cols]	**Reason:** This test case is unique and distinct because the GameBoard constructor is called using random middle allowable values for all parameters. **Function Name:** testConstructor_rand

boolean checkSpace(BoardPosition pos)

Input: State: (number to win = 4) <table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table> pos.getRow() = 4 pos.getColumn() = 4	**Output:** checkSpace = true state of the board is unchanged	**Reason:** This test case is unique and distinct because I am checking the availability of an empty position within the bounds of the board. **Function Name:** testCheckSpace_empty

boolean checkSpace(BoardPosition pos)

Input:	**Output:**	**Reason:**
State: (number to win = 4) 0 1 2 3 4 0 1 2 3 4 X pos.getRow() = 4 pos.getColumn() = 4	checkSpace = false state of the board is unchanged	This test case is unique and distinct because I am checking the availability of a position occupied by player 'X' within the bounds of the board. **Function Name:** testCheckSpace_occupied

boolean checkSpace(BoardPosition pos)

Input:	**Output:**	**Reason:**
State: (number to win = 4) 0 1 2 3 4 0 1 2 3 4 pos.getRow() = 5 pos.getColumn() = 5	checkSpace = false state of the board is unchanged	This test case is unique and distinct because I am checking the availability of a position outside the bounds of the board. **Function Name:** testCheckSpace_outOfBounds

boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input:	**Output:**	**Reason:**
State: (number to win = 4) 0 1 2 3 4 0 1 X X X X 2 3 4 lastPos.getRow() = 1 lastPos.getColumn() = 0 player = 'X'	checkHorizontalWin = true state of the board is unchanged	This test case is unique and distinct because the last 'X' was placed at the left end of the four consecutive 'X's, so the function needs to count 'X's only on the right. **Function Name:** testHorizWin_fromLeftEnd

boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input: State: (number to win = 4) 0 1 2 3 4 0 1 X X X X 2 3 4 lastPos.getRow() = 1 lastPos.getColumn() = 3 player = 'X'	**Output:** checkHorizontalWin = true state of the board is unchanged	**Reason:** This test case is unique and distinct because the last 'X' was placed at the right end of the four consecutive 'X's, so the function needs to count 'X's only on the left. **Function Name:** testHorizWin_fromRightEnd

boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input: State: (number to win = 4) 0 1 2 3 4 0 1 X X X X 2 3 4 lastPos.getRow() = 1 lastPos.getColumn() = 2 player = 'X'	**Output:** checkHorizontalWin = true state of the board is unchanged	**Reason:** This test case is unique and distinct because the last 'X' was placed in the middle of the four consecutive 'X's, so the function needs to count 'X's on the right and left. **Function Name:** testHorizWin_fromMiddle

boolean checkHorizontalWin(BoardPosition lastPos, char player)

Input: State: (number to win = 4) 0 1 2 3 4 0 1 O X X X 2 3 4 lastPos.getRow() = 1 lastPos.getColumn() = 3 player = 'X'	**Output:** checkHorizontalWin = false state of the board is unchanged	**Reason:** This test case is unique and distinct because the last 'X' placed was placed on the right end of 4 consecutive placed markers. This would be a win if the left-most char in the sequence was 'X', but it is a different char so the function needs to not count that position in the check and return false. **Function Name:** testHorizWin_noWin_otherPosIsDiffChar

boolean checkVerticalWin(BoardPosition lastPos, char player)

Input:	**Output:**	**Reason:**
State: (number to win = 4) | | 0 | 1 | 2 | 3 | 4 | | 0 | | | | | | | 1 | | | | X | | | 2 | | | | X | | | 3 | | | | X | | | 4 | | | | X | | lastPos.getRow() = 1 lastPos.getColumn() = 3 player = 'X'	checkVerticalWin = true state of the board is unchanged	This test case is unique and distinct because the last 'X' was placed at the top end of the four consecutive 'X's, so the function needs only to count 'X's below. **Function Name:** testVertWin_fromTopEnd

boolean checkVerticalWin(BoardPosition lastPos, char player)

Input:	**Output:**	**Reason:**
State: (number to win = 4) | | 0 | 1 | 2 | 3 | 4 | | 0 | | | | | | | 1 | | | | X | | | 2 | | | | X | | | 3 | | | | X | | | 4 | | | | X | | lastPos.getRow() = 4 lastPos.getColumn() = 3 player = 'X'	checkVerticalWin = true state of the board is unchanged	This test case is unique and distinct because the last 'X' was placed at the bottom end of the four consecutive 'X's, so the function needs only to count 'X's above. **Function Name:** testVertWin_fromBottomEnd

boolean checkVerticalWin(BoardPosition lastPos, char player)

Input:	**Output:**	**Reason:**
State: (number to win = 4) | | 0 | 1 | 2 | 3 | 4 | | 0 | | | | | | | 1 | | | | X | | | 2 | | | | X | | | 3 | | | | X | | | 4 | | | | X | | lastPos.getRow() = 3 lastPos.getColumn() = 3 player = 'X'	checkVerticalWin = true state of the board is unchanged	This test case is unique and distinct because the last 'X' was placed in the middle of the four consecutive 'X's, so the function needs to count 'X's above and below. **Function Name:** testVertWin_fromMiddle

boolean checkVerticalWin(BoardPosition lastPos, char player)

Input:

State: (number to win = 4)

	0	1	2	3	4
0					
1				O	
2				X	
3				X	
4				X	

lastPos.getRow() = 4
lastPos.getColumn() = 3
player = 'X'

Output:

checkVerticalWin = false

state of the board is unchanged

Reason:
This test case is unique and distinct because the last 'X' placed was placed on the bottom end of 4 consecutive placed markers. This would be a win if the top-most char in the sequence was 'X', but it is a different char so the function needs to not count that position in the check and return false.

Function Name:
testVertWin_noWin_otherPosIs DiffChar

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:

State: (number to win = 4)

	0	1	2	3	4
0		X			
1			X		
2				X	
3					X
4					

lastPos.getRow() = 0
lastPos.getColumn() = 1
player = 'X'

Output:

checkDiagonalWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last 'X' was placed at the front of four consecutive 'X's in a standard (right-down) sequence, so the function needs only to count 'X's ahead (right-down).

Function Name:
testDiagWin_StdDiag_fromFront

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:

State: (number to win = 4)

	0	1	2	3	4
0		X			
1			X		
2				X	
3					X
4					

lastPos.getRow() = 3
lastPos.getColumn() = 4
player = 'X'

Output:

checkDiagonalWin = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the last 'X' was placed at the end of four consecutive 'X's in a standard (right-down) sequence, so the function needs only to count 'X's behind (left-up).

Function Name:
testDiagWin_StdDiag_fromEnd

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input: State: (number to win = 4) | | 0 | 1 | 2 | 3 | 4 | | --- | --- | --- | --- | --- | --- | | 0 | | | | | X | | 1 | | | | X | | | 2 | | | X | | | | 3 | | X | | | | | 4 | | | | | | lastPos.getRow() = 0 lastPos.getColumn() = 4 player = 'X'	**Output:** checkDiagonalWin = true state of the board is unchanged	**Reason:** This test case is unique and distinct because the last 'X' was placed at the front of four consecutive 'X's in an anti (left-down) sequence, so the function needs only to count 'X's ahead (left-down). **Function Name:** testDiagWin_AntiDiag_fromFront

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input: State: (number to win = 4) | | 0 | 1 | 2 | 3 | 4 | | --- | --- | --- | --- | --- | --- | | 0 | | | | | X | | 1 | | | | X | | | 2 | | | X | | | | 3 | | X | | | | | 4 | | | | | | lastPos.getRow() = 3 lastPos.getColumn() = 1 player = 'X'	**Output:** checkDiagonalWin = true state of the board is unchanged	**Reason:** This test case is unique and distinct because the last 'X' was placed at the end of four consecutive 'X's in an anti (left-down) sequence, so the function needs only to count 'X's ahead (right-up). **Function Name:** testDiagWin_AntiDiag_fromEnd

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input: State: (number to win = 5) | | 0 | 1 | 2 | 3 | 4 | | --- | --- | --- | --- | --- | --- | | 0 | O | | | | | | 1 | | X | | | | | 2 | | | X | | | | 3 | | | | X | | | 4 | | | | | X | lastPos.getRow() = 4 lastPos.getColumn() = 4 player = 'X'	**Output:** checkDiagonalWin = false state of the board is unchanged	**Reason:** This test case is unique and distinct because the last 'X' placed was placed at the end of 5 consecutive placed markers in a standard diagonal pattern. This would result in a win if the front char in the sequence was 'X', but it is a different char so the function needs to not count that position in the check and return false. **Function Name:** testDiagWin_StdDiag_noWin_otherPosIsDiffChar

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:

State: (number to win = 5)

	0	1	2	3	4
0					O
1				X	
2			X		
3		X			
4	X				

lastPos.getRow() = 4
lastPos.getColumn() = 0
player = 'X'

Output:

checkDiagonalWin = false

state of the board is unchanged

Reason:
This test case is unique and distinct because the last 'X' placed was placed at the end of 5 consecutive placed markers in an anti diagonal pattern. This would result in a win if the front char in the sequence was 'X', but it is a different char so the function needs to not count that position in the check and return false.

Function Name:
testDiagWin_AntiDiag_noWin_otherPosIsDiffChar

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input:

State: (number to win = 5)

	0	1	2	3	4
0	X				X
1		X		X	
2			O		
3		X		X	
4	X				X

lastPos.getRow() = 2
lastPos.getColumn() = 2
player = 'O'

Output:

checkDiagonalWin = false

state of the board is unchanged

Reason:
This test case is unique and distinct because the last player was placed at the center of two sequences of 'X's in a would-be win position in both directions, but the char placed does not match the sequence. The function needs to recognize that the player parameter does not match the chars in the sequences.

Function Name:
testDiagWin_noWin_lastPosIsDiffChar

boolean checkForDraw()

Input:

State: (number to win = 5)

	0	1	2	3	4
0					
1					
2					
3					
4					

Output:

checkForDraw = false

state of the board is unchanged

Reason:
This test case is unique and distinct because the board is empty, so the function needs to determine that the board is not full and return false.

Function Name:
testDraw_noDraw_emptyBoard

boolean checkForDraw()

Input:

State: (number to win = 4)

	0	1	2	3	4
0		X	X	X	X
1	X	X	X	X	X
2	X	X	X	X	X
3	X	X	X	X	X
4	X	X	X	X	X

Output:

checkForDraw = false

state of the board is unchanged

Reason:
This test case is unique and distinct because the board is full except for the first position on the board, so the function needs to determine that the board is not full and return false.

Function Name:
testDraw_noDraw_1LessThanFullBoard_firstEmpty

NOTE: the checkForDraw function is designed to return false if there are remaining spaces on the board, or true if there aren't any [the board is full]. It is not concerned with whether or not there is a winning sequence on the board.

boolean checkForDraw()

Input:

State: (number to win = 4)

	0	1	2	3	4
0	X	X	X	X	X
1	X	X	X	X	X
2	X	X	X	X	X
3	X	X	X	X	X
4	X	X	X	X	

Output:

checkForDraw = false

state of the board is unchanged

Reason:
This test case is unique and distinct because the board is full except for the last position on the board, so the function needs to determine that the board is not full and return false.

Function Name:
testDraw_noDraw_1LessThanFullBoard_lastEmpty

NOTE: the checkForDraw function is designed to return false if there are remaining spaces on the board, or true if there aren't any [the board is full]. It is not concerned with whether or not there is a winning sequence on the board.

boolean checkForDraw()

Input:

State: (number to win = 4)

	0	1	2	3	4
0	X	X	X	X	X
1	X	X	X	X	X
2	X	X	X	X	X
3	X	X	X	X	X
4	X	X	X	X	X

Output:

checkForDraw = true

state of the board is unchanged

Reason:
This test case is unique and distinct because the board is full, so the function needs to determine that the board is full and return true.

Function Name:
testDraw_fullBoard

char whatsAtPos(BoardPosition pos)

Input:

State: (number to win = 4)

	0	1	2	3	4
0					
1					
2					
3					
4					

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

Output:

whatsAtPos = ‘ ’

state of the board is unchanged

Reason:
This test case is unique and distinct because the board is empty, so the function should return the char ‘ ‘.

Function Name:
testWhatsAtPos_emptyBoard

char whatsAtPos(BoardPosition pos)

Input:

State: (number to win = 4)

	0	1	2	3	4
0	A				
1					
2					
3					
4					

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

Output:

whatsAtPos = ‘A’

state of the board is unchanged

Reason:
This test case is unique and distinct because player ‘A’ is at parameter ‘pos’, which is at the first cell on the board, and there are no other players on the board, so the function needs to return the char ‘A’.

Function Name:
testWhatsAtPos_atPlacedChar

char whatsAtPos(BoardPosition pos)

Input:

State: (number to win = 4)

	0	1	2	3	4
0	A				
1					
2					
3					
4					

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

Output:

whatsAtPos = ‘ ’

state of the board is unchanged

Reason:
This test case is unique and distinct because player ‘A’ has been placed on the board, but is not at parameter ‘pos’, and there are no other players on the board, so the function needs to return the char ‘ ’.

Function Name:
testWhatsAtPos_notAtPlacedChar

char whatsAtPos(BoardPosition pos)

Input:

State: (number to win = 4)

	0	1	2	3	4
0	A				
1					
2					
3					
4					B

pos.getRow() = 4
pos.getColumn() = 4

Output:

whatsAtPos = ‘B’

state of the board is unchanged

Reason:
This test case is unique and distinct because players ‘A’ and ‘B’ have been placed on the board, and ‘B’ is at parameter ‘pos’, so the function needs to return the char ‘B’.

Function Name:
testWhatsAtPos_2PlacedChars

char whatsAtPos(BoardPosition pos)

Input:

State: (number to win = 4)

	0	1	2	3	4
0	A	B	C	D	E
1	F	G	H	I	J
2	K	L	M	N	O
3	P	Q	R	S	T
4	U	V	W	X	Y

pos.getRow() = 2
pos.getColumn() = 2

Output:

whatsAtPos = ‘M’

state of the board is unchanged

Reason:
This test case is unique and distinct because a different player has been placed in each position on the board. The function should return the player found at ‘pos’, which is ‘M’.

Function Name:
testWhatsAtPos_fullBoard

boolean isPlayerAtPos(BoardPosition pos, char player)

Input:

State: (number to win = 4)

	0	1	2	3	4
0					
1					
2					
3					
4					

pos.getRow() = 0
pos.getColumn() = 0
player = ‘A’

Output:

isPlayerAtPos = false

state of the board is unchanged

Reason:
This test case is unique and distinct because the board is empty, so the function should return false.

Function Name:
testIsPlayerAtPos_emptyBoard

boolean isPlayerAtPos(BoardPosition pos, char player)

Input:	**Output:**	**Reason:**
State: (number to win = 4) | | 0 | 1 | 2 | 3 | 4 | | 0 | A | | | | | | 1 | | | | | | | 2 | | | | | | | 3 | | | | | | | 4 | | | | | | pos.getRow() = 0 pos.getColumn() = 0 player = 'A'	isPlayerAtPos = true state of the board is unchanged	This test case is unique and distinct because player 'A' is at parameter 'pos', which is at the first cell on the board, and there are no other players on the board, so the function needs to return true. **Function Name:** testIsPlayerAtPos_atPlacedChar

boolean isPlayerAtPos(BoardPosition pos, char player)

Input:	**Output:**	**Reason:**
State: (number to win = 4) | | 0 | 1 | 2 | 3 | 4 | | 0 | A | | | | | | 1 | | | | | | | 2 | | | | | | | 3 | | | | | | | 4 | | | | | B | pos.getRow() = 0 pos.getColumn() = 0 player = 'B'	isPlayerAtPos = false state of the board is unchanged	This test case is unique and distinct because players 'A' and 'B' have been placed on the board, and the pos parameter holds the location of 'A', while the player passed is 'B', so the function should return false. **Function Name:** testIsPlayerAtPos_firstPos_secondPlayer

boolean isPlayerAtPos(BoardPosition pos, char player)

Input:	**Output:**	**Reason:**
State: (number to win = 4) | | 0 | 1 | 2 | 3 | 4 | | 0 | A | | | | | | 1 | | | | | | | 2 | | | | | | | 3 | | | | | | | 4 | | | | | B | pos.getRow() = 4 pos.getColumn() = 4 player = 'B'	isPlayerAtPos = true state of the board is unchanged	This test case is unique and distinct because players 'A' and 'B' have been placed on the board, and the pos parameter holds the location of 'B', and the player passed is 'B', so the function should return true. **Function Name:** testIsPlayerAtPos_secondPos_secondPlayer

boolean isPlayerAtPos(BoardPosition pos, char player)

Input:

State: (number to win = 4)

	0	1	2	3	4
0	A	B	C	D	E
1	F	G	H	I	J
2	K	L	M	N	O
3	P	Q	R	S	T
4	U	V	W	X	Y

pos.getRow() = 2
pos.getColumn() = 2
player = 'M'

Output:

isPlayerAtPos = true

state of the board is unchanged

Reason:
This test case is unique and distinct because a different player has been placed in each position on the board. The function should determine that the player found at 'pos' matches the player 'M' and return true.

Function Name:
testIsPlayerAtPos_fullBoard

void placeMarker(BoardPosition marker, char player)

Input:

State:

	0	1	2	3	4
0					
1					
2					
3					
4					

marker.getRow() = 0
marker.getColumn() = 0
player = 'A'

Output:

State:

	0	1	2	3	4
0	A				
1					
2					
3					
4					

Reason:
This test case is unique and distinct because it is placing the marker of a player not yet represented on the board, it is the first placement on the board, and it is at the minimum row and column positions. The function should update the board with the specified player at the specified marker.

Function Name:
testPlaceMarker_firstPlayer_firstPlacement_minRowCol

void placeMarker(BoardPosition marker, char player)

Input:

State:

	0	1	2	3	4
0					
1					
2					
3					
4					

marker.getRow() = 4
marker.getColumn() = 4
player = 'A'

Output:

State:

	0	1	2	3	4
0					
1					
2					
3					
4					A

Reason:
This test case is unique and distinct because it is placing the marker of a player not yet represented on the board, it is the first placement on the board, and it is at the maximum row and column positions. The function should update the board with the specified player at the specified marker.

Function Name:
testPlaceMarker_firstPlayer_firstPlacement_maxRowCol

void placeMarker(BoardPosition marker, char player)

Input:

State:

	0	1	2	3	4
0	A				
1					
2					
3					
4					

marker.getRow() = 0
marker.getColumn() = 1
player = ‘B’

Output:

State:

	0	1	2	3	4
0	A	B			
1					
2					
3					
4					

Reason:
This test case is unique and distinct because it is placing the marker of a player not yet represented on the board, and it is the second placement on the board. The function should update the board with the specified player at the specified marker.

Function Name:
testPlaceMarker_secondPlayer_firstPlacement

void placeMarker(BoardPosition marker, char player)

Input:

State:

	0	1	2	3	4
0	A	B			
1					
2					
3					
4					

marker.getRow() = 1
marker.getColumn() = 1
player = ‘A’

Output:

State:

	0	1	2	3	4
0	A	B			
1		A			
2					
3					
4					

Reason:
This test case is unique and distinct because it is placing the marker of the first player to be represented on the board, and it is the second placement for that player. The function should update the board with the specified player at the specified marker.

Function Name:
testPlaceMarker_firstPlayer_secondPlacement

void placeMarker(BoardPosition marker, char player)

Input:

State:

	0	1	2	3	4
0	A	B			
1		A			
2					
3					
4					

marker.getRow() = 1
marker.getColumn() = 0
player = ‘B’

Output:

State:

	0	1	2	3	4
0	A	B			
1	B	A			
2					
3					
4					

Reason:
This test case is unique and distinct because it is placing the marker of the second player to be represented on the board, and it is the second placement for that player. The function should update the board with the specified player at the specified marker.

Function Name:
testPlaceMarker_secondPlayer_secondPlacement