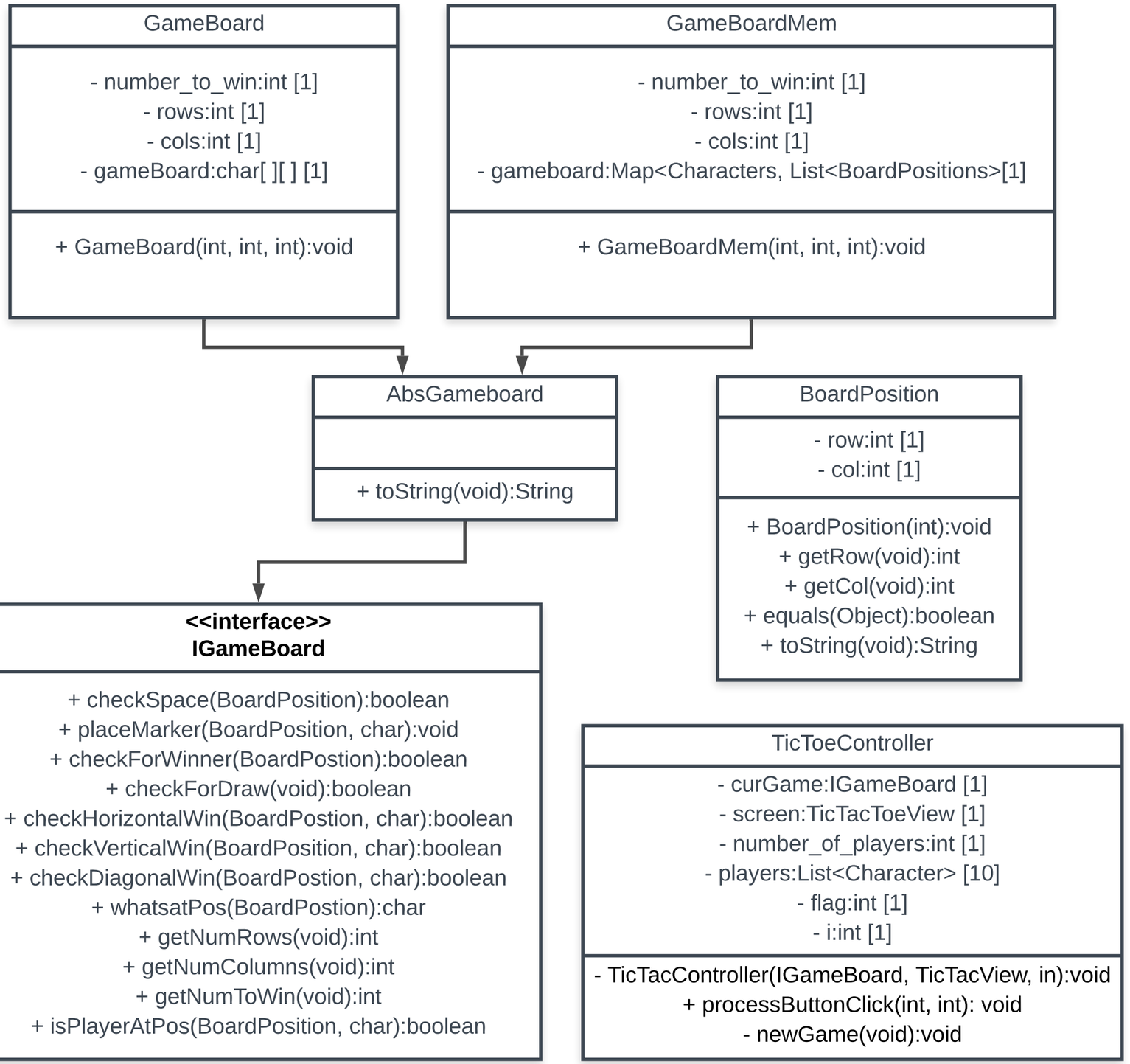
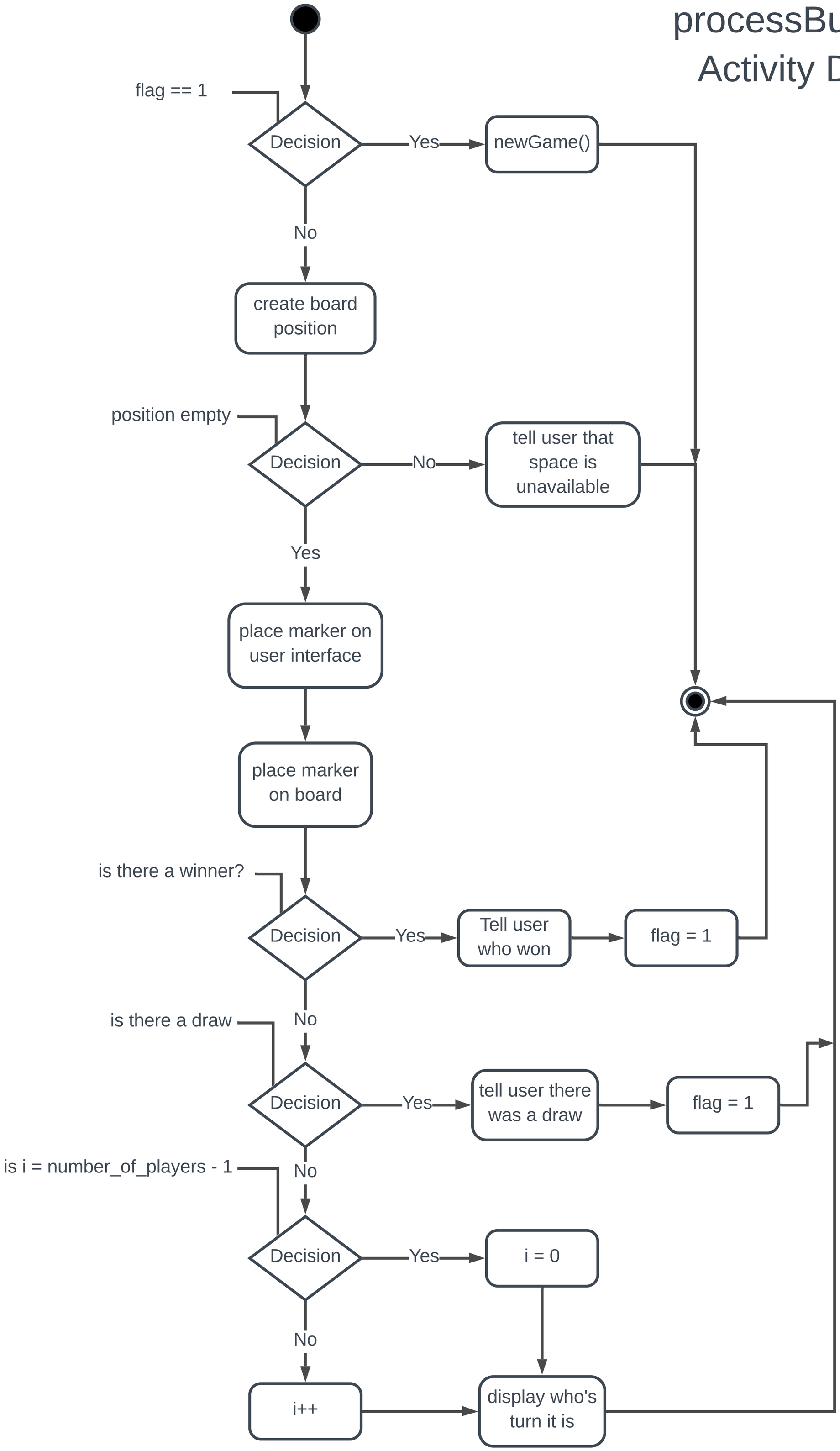
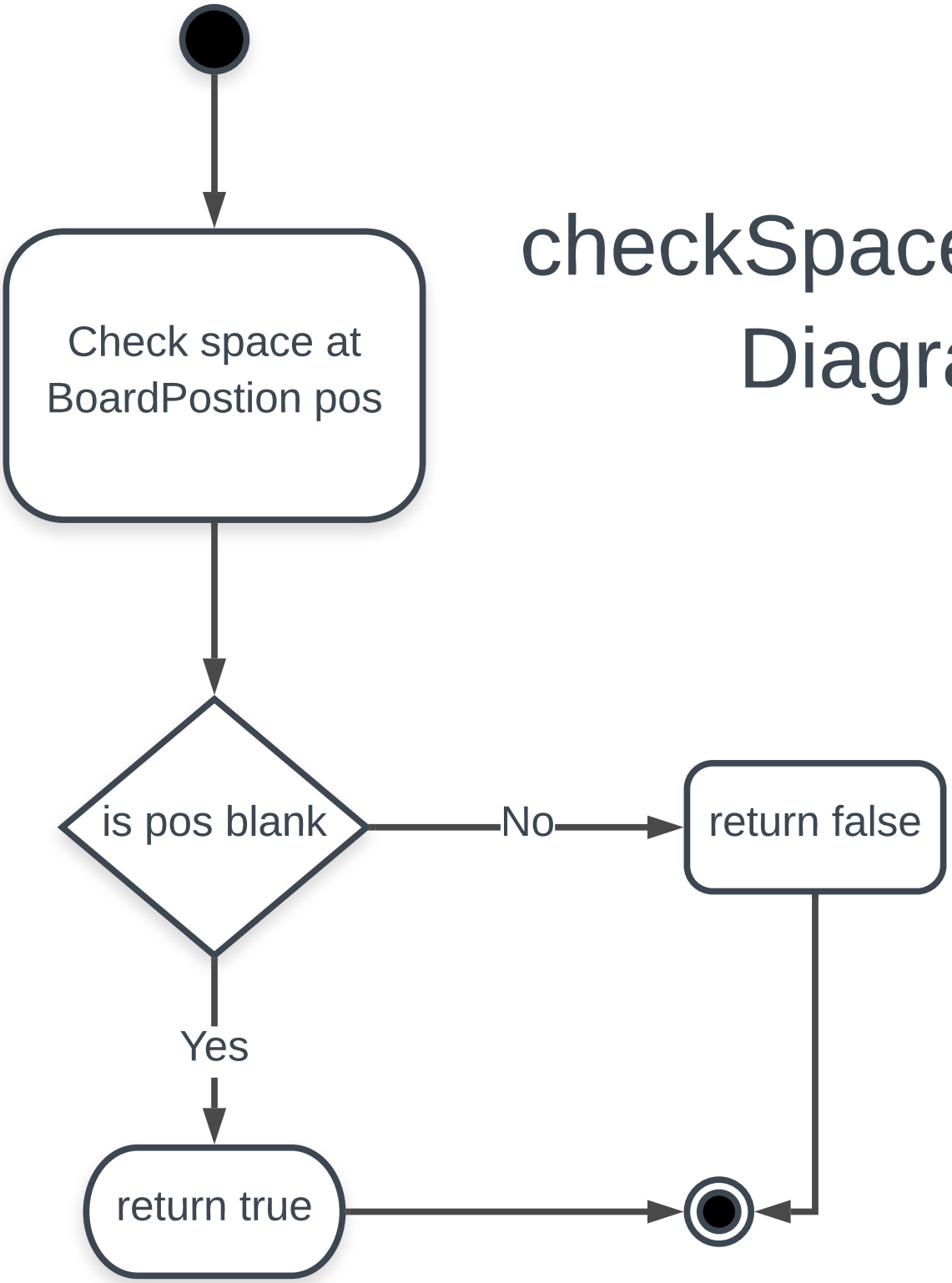
Class Diagrams

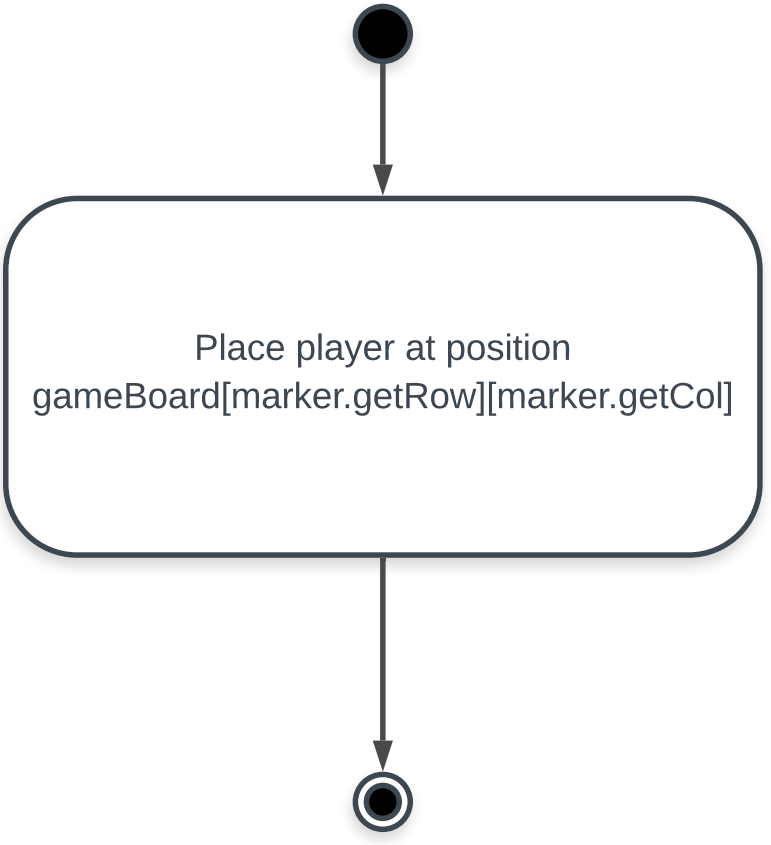


processButtonClick Activity Diagram



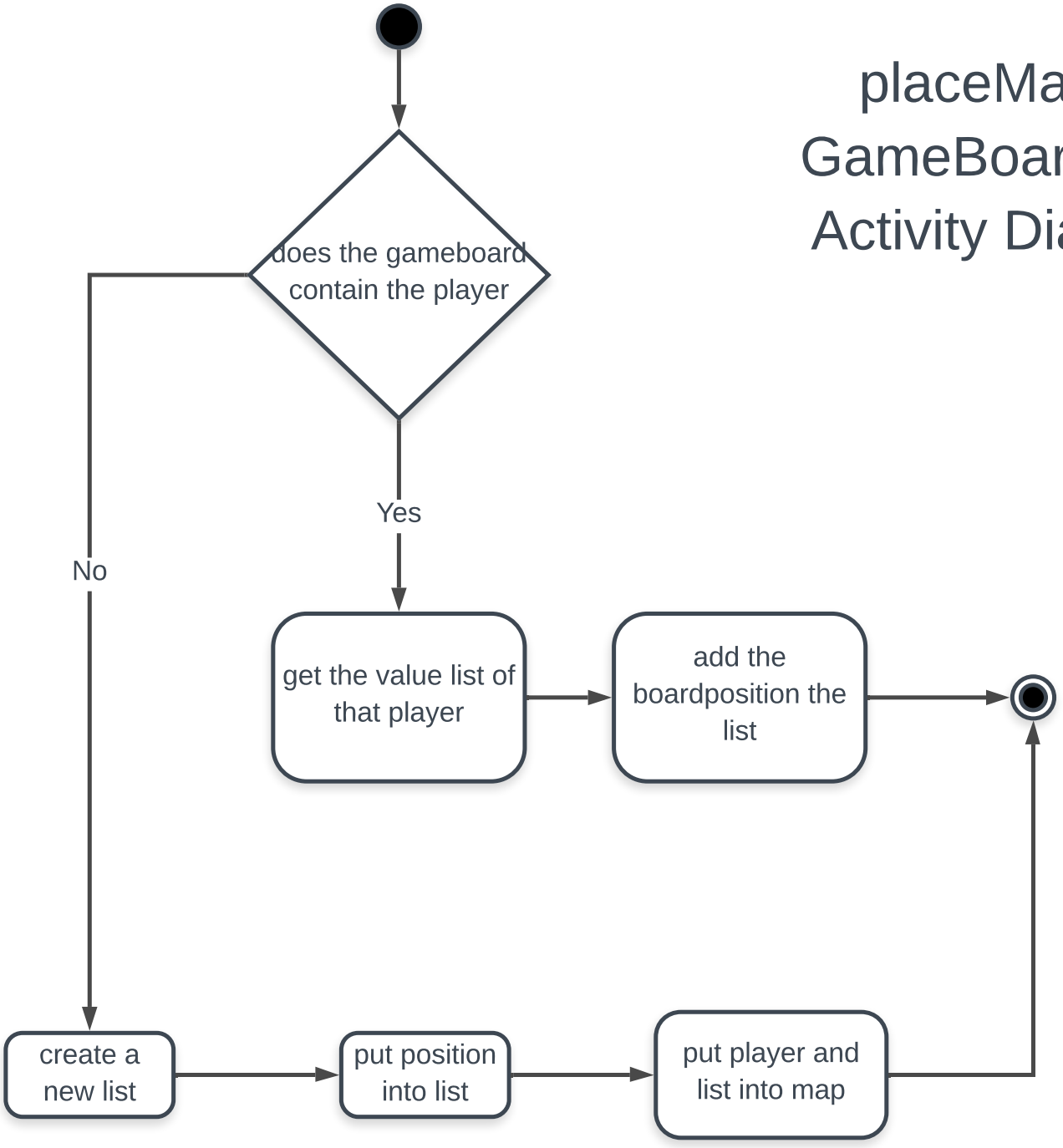
checkSpace Activity Diagram



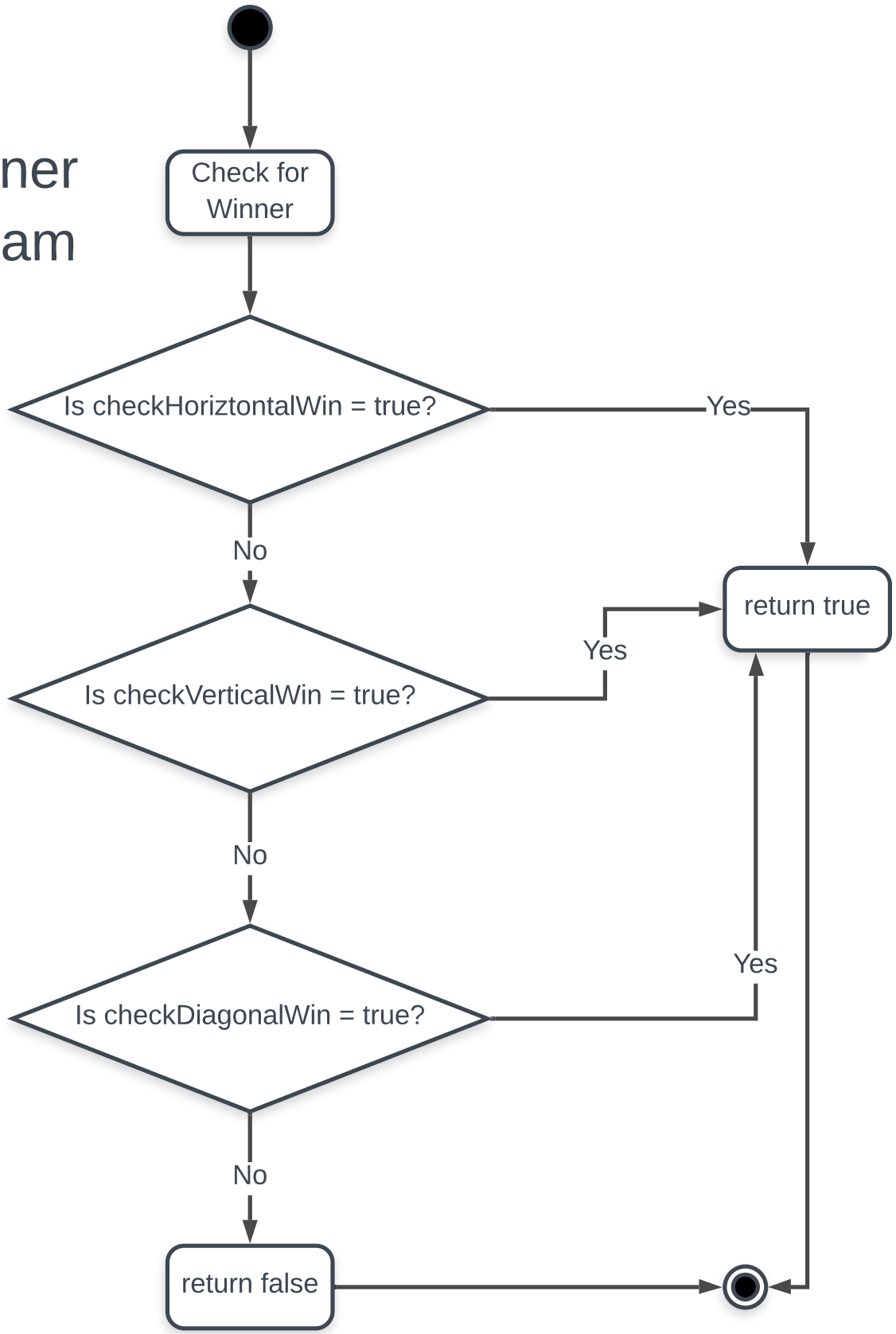


placeMarker Activity Diagram

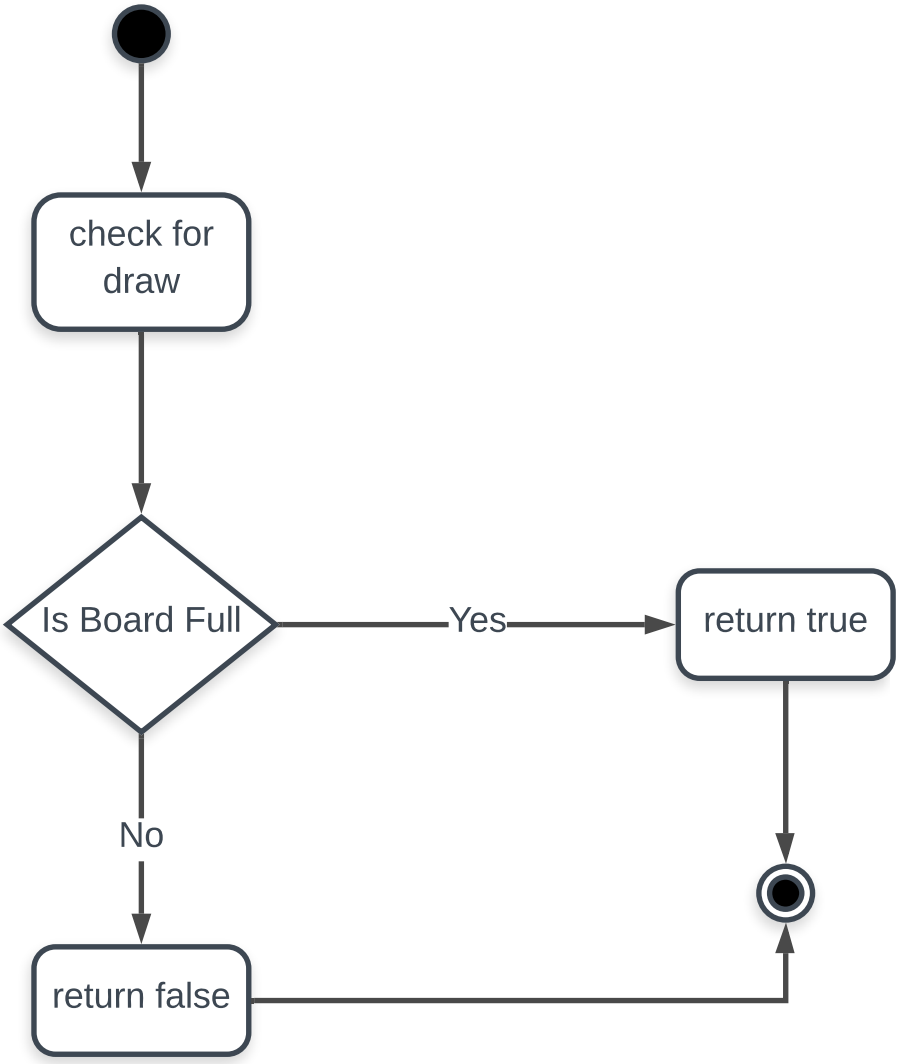
placeMarker GameBoardMem Activity Diagram



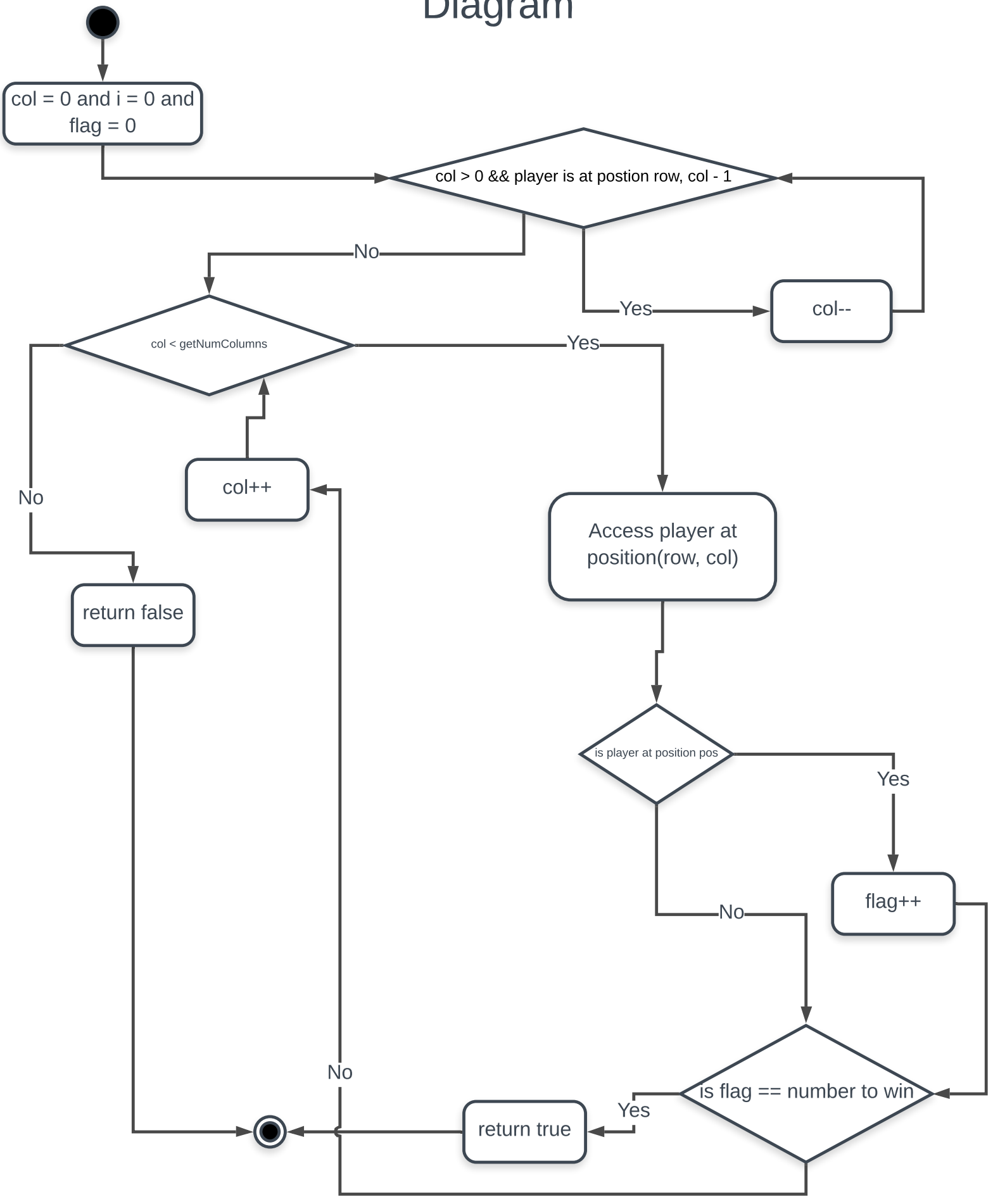
checkForWinner Activity Diagram



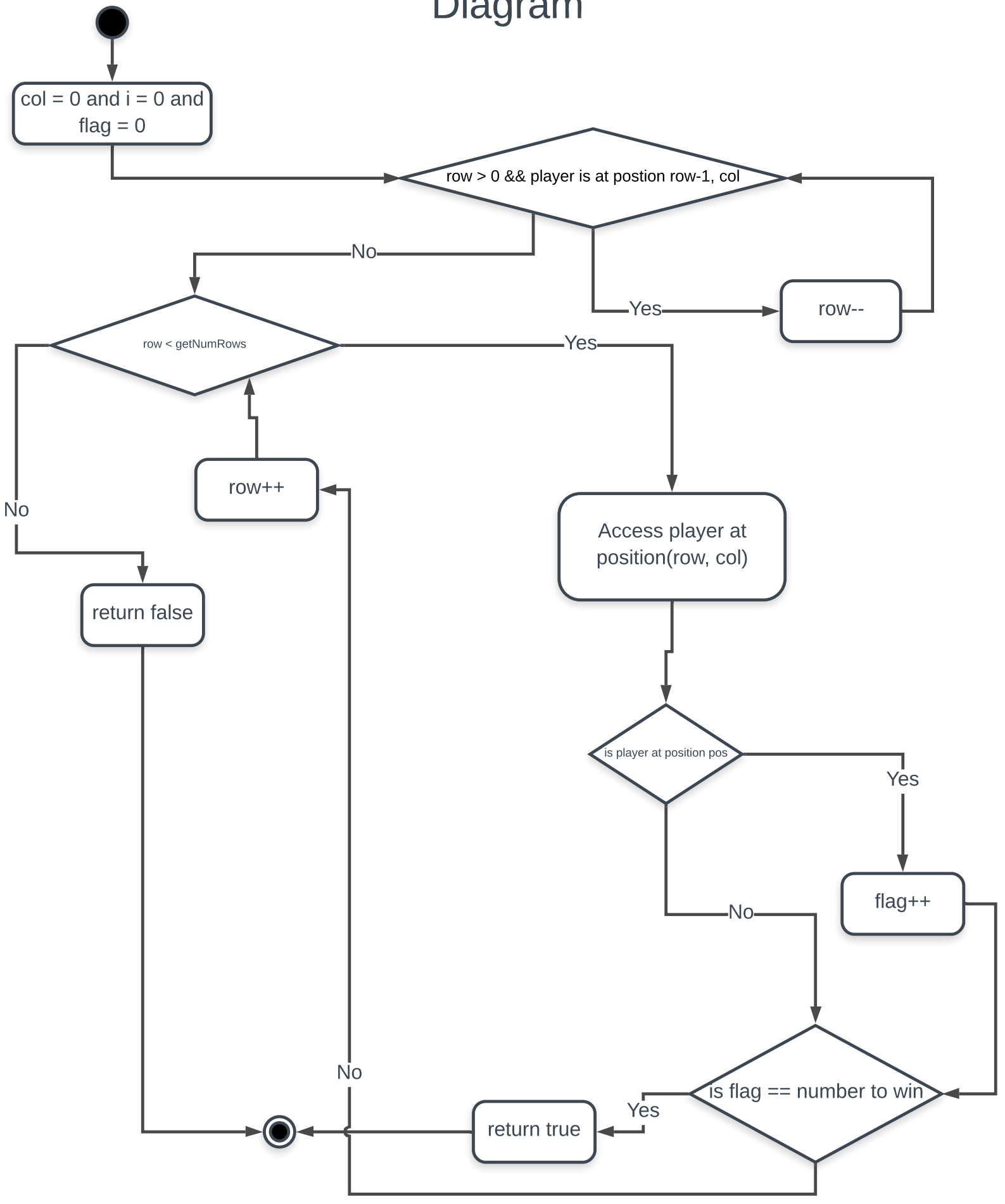
checkForDraw Activity Diagram



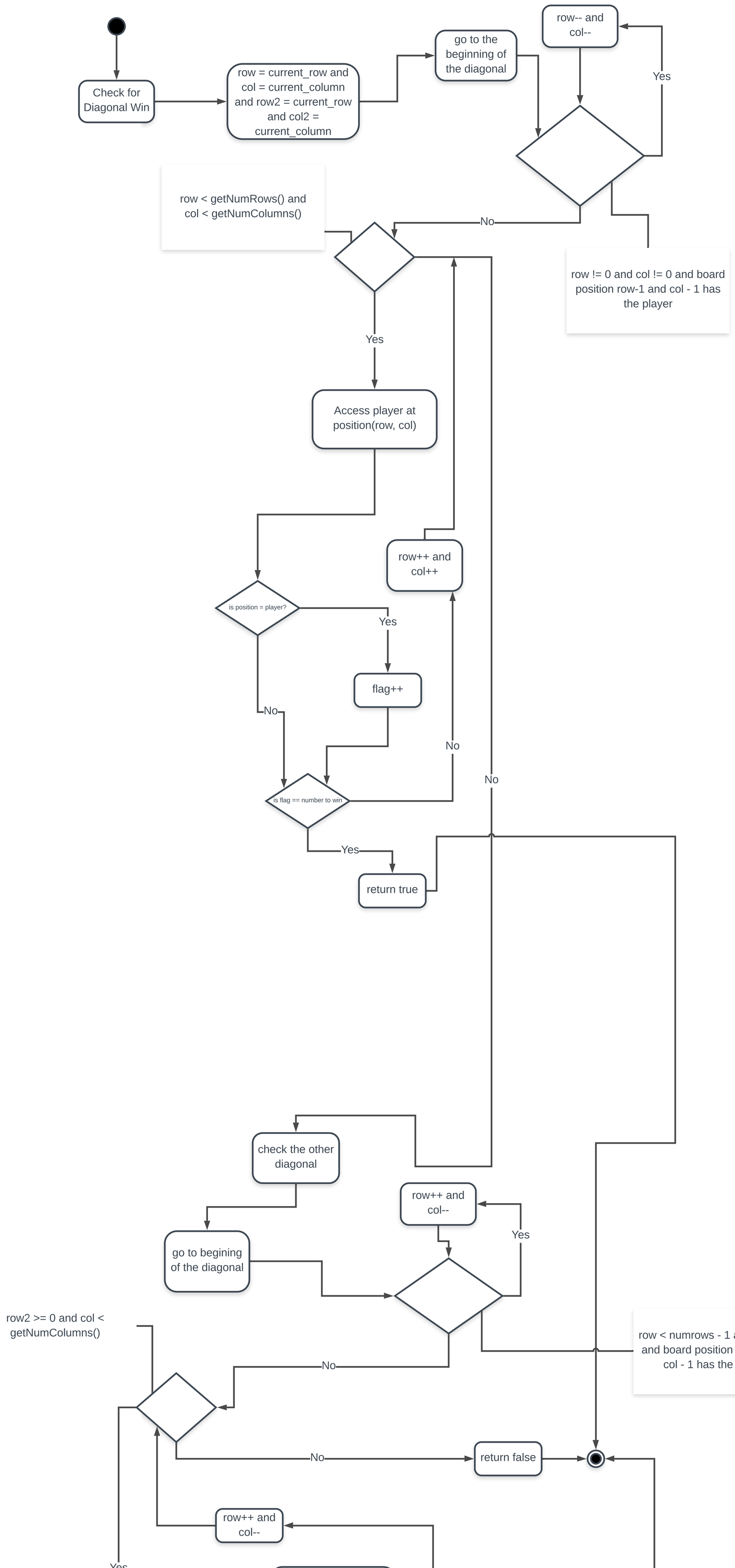
checkHorizontallWin Activity Diagram

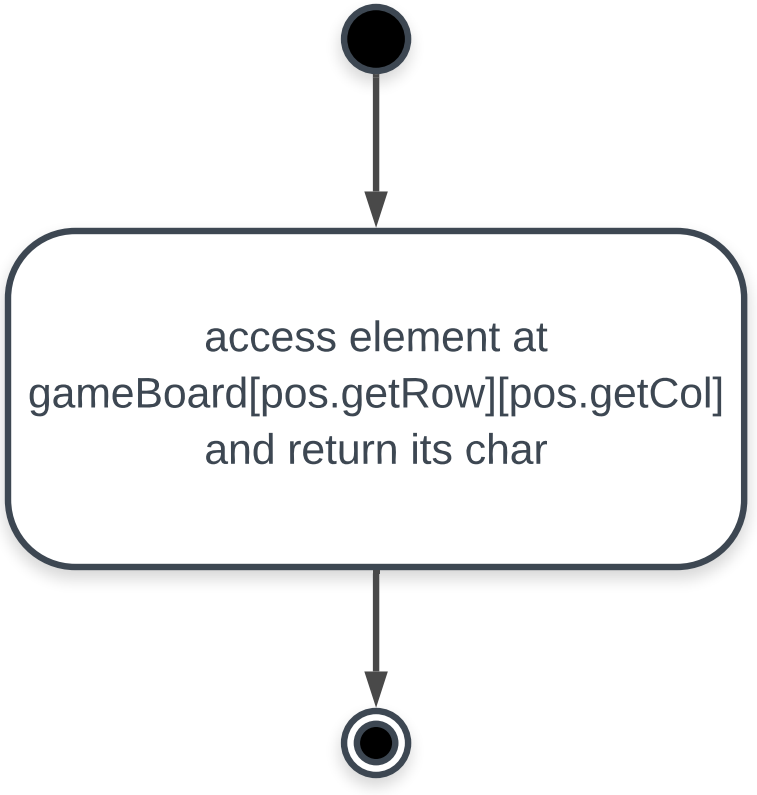


checkVerticalWin Activity Diagram



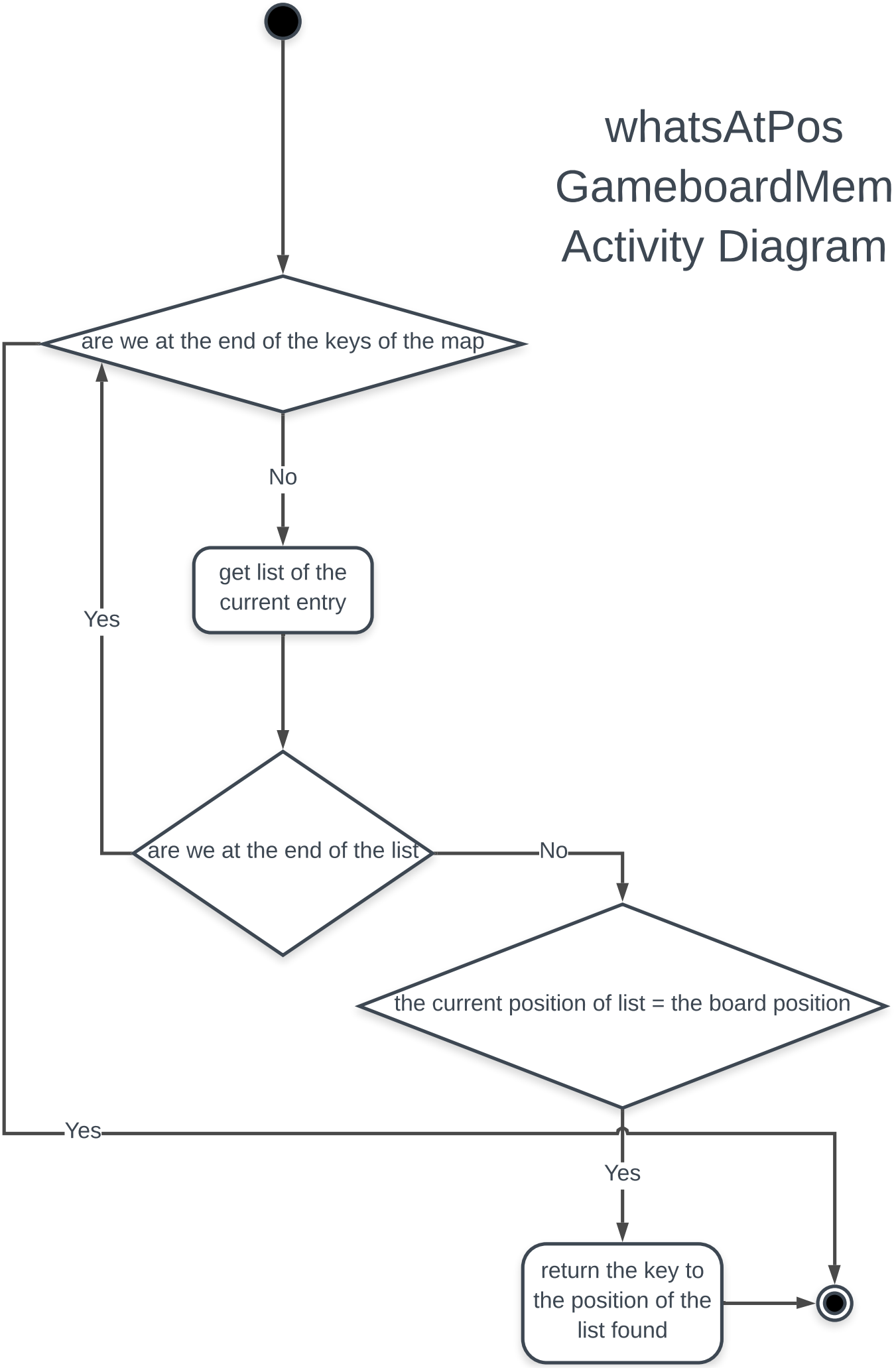
checkDiagonalWin Activity Diagram

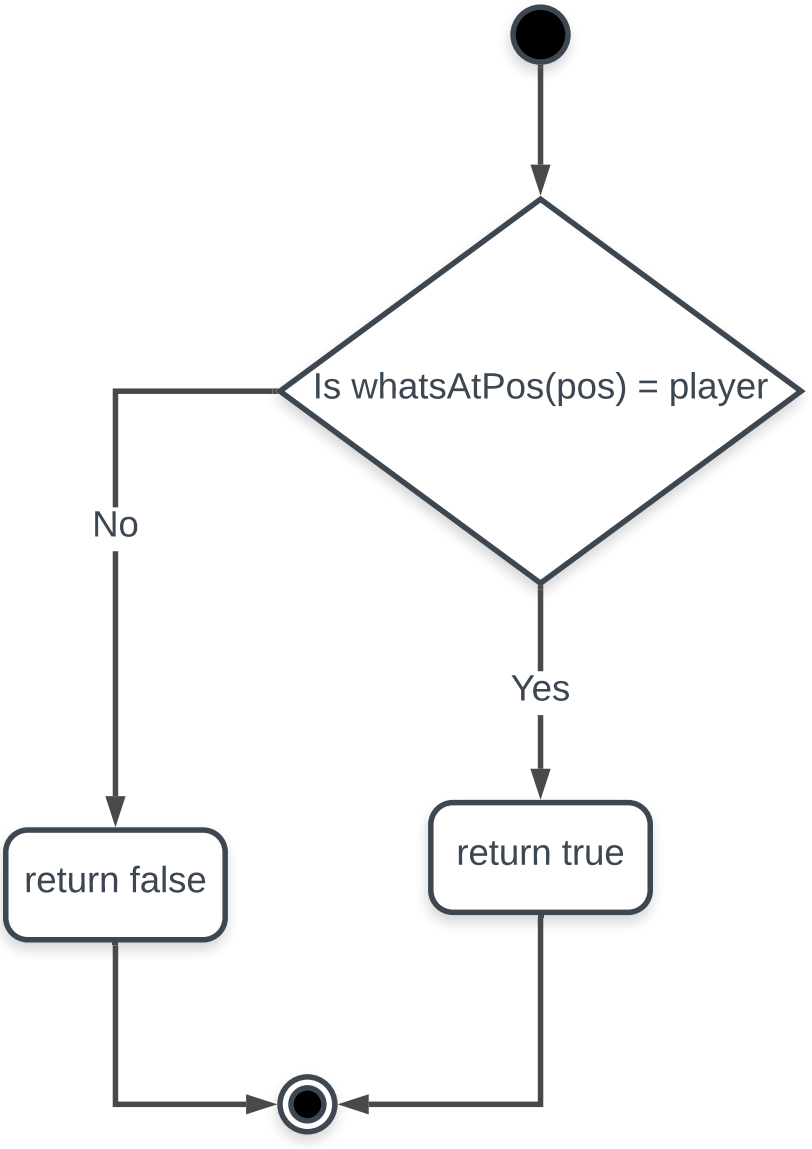




whatsAtPos Activity Diagram

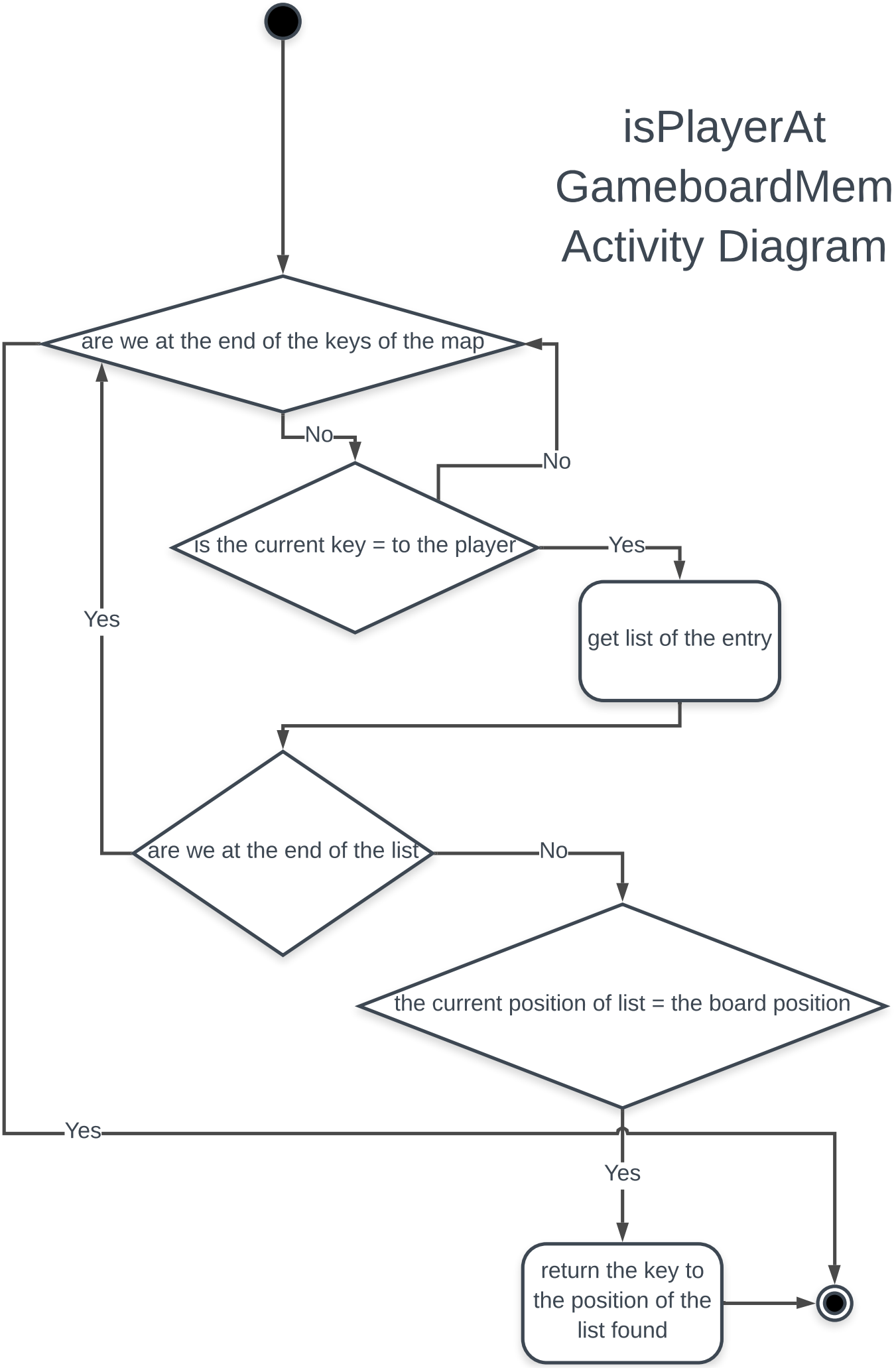
whatsAtPos GameboardMem Activity Diagram



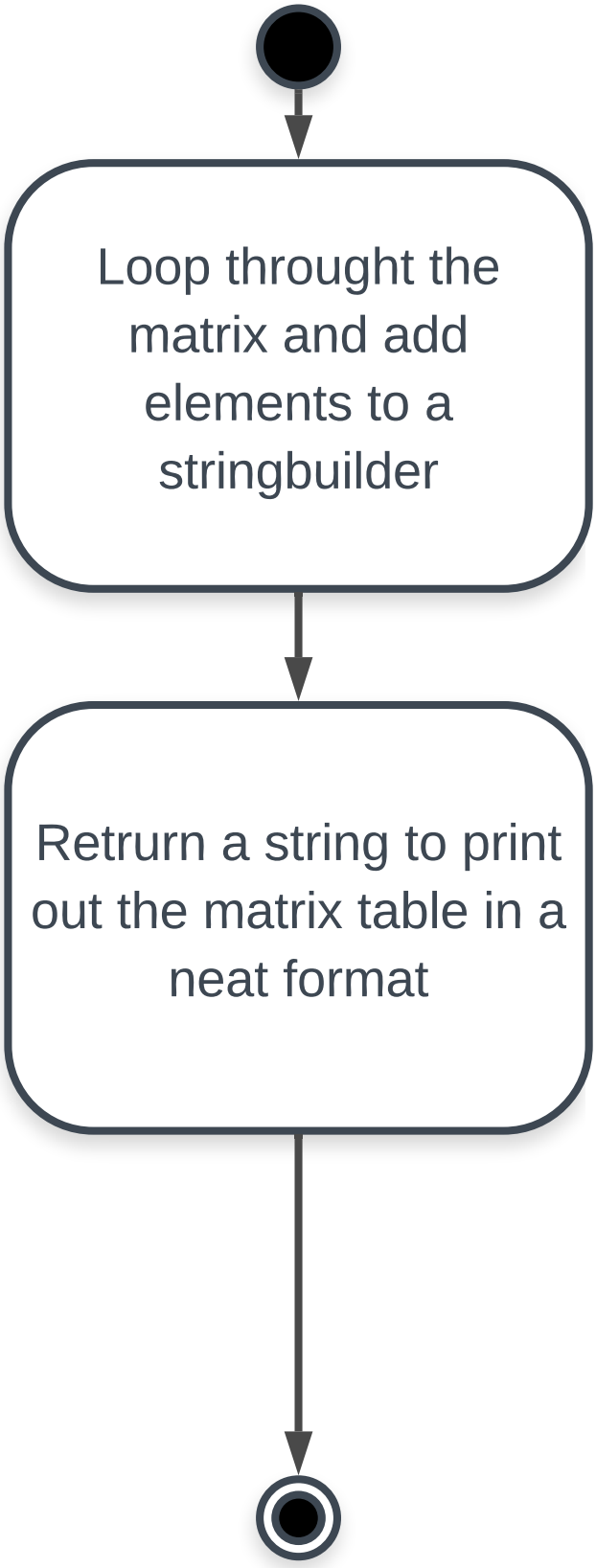


isPlayerAt
Activity Diagram

isPlayerAt GameboardMem Activity Diagram



toString Activity Diagram



Requirements Analysis

Functional Requirements

- As a user, I can place my marker on the board, so I can try to win
- As a user, I can put my marker in a empty spot, so as not to erase someone elses marker.
- As a user, I can pick a row and column within the bounds of the matrix, so as to not go out of bounds
- As a user, I can place my markers in a horizontal pattern, so I can win horizontally
- As a user, I can place my markers in a vertical pattern, so I can win vertically
- As a user, I can place my markers in a diagonal pattern from top-left to bottom-right, so I can win diagonally.
- As a user, I can place my markers in a diagonal pattern from bottom-left to top-right, so I can win diagonally
- As a user, I can choose to play a new game, so I can play over again from the beginning
- As a user, I can have my game end in a tie, so neither me or my opponent win
- As a user, I can choose the number of rows, so I can dictate the amount of rows I want
- As a user, I can choose the number of columns, so I can dictate the amount of columns I want
- As a user, I can choose how many I need in a row to win, so I can dictate the number in row to win
- As a user, I can choose how many players that I can have, so I can play with multiple people

Non-Functional Requirments

- system must work in intellij
- must be written in c++
- must have at least 2 players
- can have a max of 10 characters
- 0,0 is the top left of the board
- the game is written in Java
- the game has a user interface

Test Documentation

void GameBoard(int rows, int cols, int num_to_win)

Input	Output	Reason:
State: rows=3 cols=3 num_to_win=3	State of board is unchanged	The reason this is a distinct case is because the constructor is tested constructing the board with the minimum size. Function Name: test_Constructor_min_game_board_size

void GameBoard(int rows, int cols, int num_to_win)

Input	Output	Reason:
State: 100x100 board with nothing in it rows=100 cols=100 num_to_win=3	State of board is unchanged	The reason this is a distinct case is because we test the constructor with a board that is the maximum size Function Name: test_Constructor_max_game_board_size

void GameBoard(int rows, int cols, int num_to_win)

Input	Output	Reason:
State: rows=3 cols=5 num_to_win=3	State of board is unchanged	The reason this test case is unique is because we test constructing a board with an uneven number of rows and columns Function Name: test_Constructor_uneven_game_board_size

boolean checkSpace(BoardPosition pos)

Input	Output	Reason:
State: pos.getRow=3 pos.getColumn=4	checkSpace = true State of board is unchanged	The reason this test case is unique is because we are checking a space that is empty Function Name: test_checkSpace_empty_space

boolean checkSpace(BoardPosition pos)

Input	Output	Reason:
State: pos.getRow=3 pos.getColumn=1	checkSpace = false State of board is unchanged	The reason this test case is unique is because we are testing a space that already has something in it Function Name: test_checkSpace_space_already_has_character_in_it

	0	1	2	3	4
0					
1					
2					
3		X			
4					

boolean checkSpace(BoardPosition pos)

Input	Output	Reason:
State: pos.getRow=6 pos.getColumn=5	checkSpace = false State of board is unchanged	The reason this test case is unique is because we are testing a space that is out of bounds Function Name: test_checkSpace_space_out_of_bounds

	0	1	2	3	4
0					
1					
2					
3					
4					

boolean checkHorizontalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) pos.getRow=0 pos.getColumn=2 player=X	checkHorizontalWin = true State of board is unchanged	The reason this test case is unique is because we are checking a horizontal win in the top left corner of the board which has to check all the way to the minimum column number. Function Name: test_checkHorizontalWin_top_left_corner

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

boolean checkHorizontalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) pos.getRow=2 pos.getColumn=3 player=X	checkHorizontalWin = true State of board is unchanged	The reason this test case is unique is because we are testing for a win in the middle of the board where we don't have to check all the way to the edges. Function Name: test_checkHorizontalWin_middle_of_board

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

boolean checkHorizontalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) pos.getRow=4 pos.getColumn=4 player=X	checkHorizontalWin = true State of board is unchanged	The reason this test case is unique is because we are testing for a win in the bottom right corner where we are checking all the way to the maximum column size. Function Name: test_checkHorizontalWin_bottom_right_of_board

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

boolean checkHorizontalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) pos.getRow=2 pos.getColumn=1 player=X	checkHorizontalWin = true State of board is unchanged	The reason this test case is unique is because we are testing for a win in were a marker is placed in front of two of the same markers so it doesn't check left and it only has to check right. Function Name: test_checkHorizontalWin_place_at_beginning_of_2markers

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

boolean checkHorizontalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) pos.getRow=2 pos.getColumn=2 player=X	checkHorizontalWin = true State of board is unchanged	The reason this test case is unique is because we are testing for a win in were a marker is placed in the middle of 2 of the same markers so it has to check left and right. Function Name: test_checkHorizontalWin_place_middle_of_2markers

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

boolean checkVerticalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) pos.getRow=2 pos.getColumn=0 player=X	checkVerticalWin = true State of board is unchanged	The reason this test case is unique is because we are checking a vertical win in the top left corner of the board which has to check all the way to the minimum row number. Function Name: test_checkVerticalWin_top_left_corner

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

boolean checkVerticalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) pos.getRow=3 pos.getColumn=2 player=X	checkVerticalWin = true State of board is unchanged	The reason this test case is unique is because we are testing for a win in the middle of the board where we don't have to check all the way to the edges. Function Name: test_checkVerticalWin_middle_of_board

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

boolean checkVerticalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | 3 | 4 | | 0 | | | | | | | 1 | | | | | | | 2 | | | | | X | | 3 | | | | | X | | 4 | | | | | X | pos.getRow=4 pos.getColumn=4 player=X	checkVerticalWin = true State of board is unchanged	The reason this test case is unique is because we are testing for a win in the bottom right corner where we are checking all the way to the maximum row size. Function Name: test_checkVerticalWin_bottom_right_of_board

boolean checkVerticalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | 3 | 4 | | 0 | | | | | | | 1 | | X | | | | | 2 | | X | | | | | 3 | | X | | | | | 4 | | | | | | pos.getRow=1 pos.getColumn=1 player=X	checkVerticalWin = true State of board is unchanged	The reason this test case is unique is because we are testing for a win in were a marker is placed in front of two of the same markers so it doesn't check up and it only has to check down. Function Name: test_checkVerticalWin_place_at_beginning_of_2markers

boolean checkVerticalWin(BoardPosition lastPos, char

Input	Output	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | 3 | 4 | | 0 | | | | | | | 1 | | X | | | | | 2 | | X | | | | | 3 | | X | | | | | 4 | | | | | | pos.getRow=2 pos.getColumn=1 player=X	checkVerticalWin = true State of board is unchanged	The reason this test case is unique is because we are testing for a win in were a marker is placed in the middle of 2 of the same markers so it has to check up and down. Function Name: test_checkVerticalWin_place_middle_of_2markers

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason:
State: (number to win = 3) pos.getRow=2 pos.getColumn=2 player=X	checkDiagonalWin = true State of board is unchanged	This test case is unique because we are checking the outer bounds of the bottom left of the board. We will have to check all the way to the max row number and the minimum column number Function Name: test_checkDiagonalWin_bottomleft_to_topright_at_bottom_left

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

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason:
State: (number to win = 3) pos.getRow=0 pos.getColumn=4 player=X	checkDiagonalWin = true State of board is unchanged	The test case is unique because we are checking the outer bounds of the top right of the board. We have to check all the way to the max column number and minimum row number. Function Name: test_checkDiagonalWin_bottomleft_to_topright_at_top_right

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

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason:
State: (number to win = 3) pos.getRow=4 pos.getColumn=4 player=X	checkDiagonalWin = true State of board is unchanged	This test case is unique because we are testing the outer bounds of bottom right of the board. We have to check all the way to the max row and column number. Function Name: test_checkDiagonalWin_topleft_to_bottomright_bottom_right

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

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | 3 | 4 | | 0 | X | | | | | | 1 | | X | | | | | 2 | | | X | | | | 3 | | | | | | | 4 | | | | | | pos.getRow=0 pos.getColumn=0 player=X	checkDiagonalWin = true State of board is unchanged	This test case is unique because we are checking the outer bounds of the top left of the board. We have to check all the way to the minimum row and column number. Function Name: test_checkDiagonalWin_topleft_to_bottomright_top_left

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | 3 | 4 | | 0 | | | | | | | 1 | | | | | | | 2 | X | | | | | | 3 | | X | | | | | 4 | | | X | | | pos.getRow=4 pos.getColumn=2 player=X	checkDiagonalWin = true State of board is unchanged	This test case is unique because we are checking for a win in a smaller part of the board at the bottom left. We aren't checking for a win in the larger diagonal areas. Function Name: test_checkDiagonalWin_bottom_left_small_diagonal

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | 3 | 4 | | 0 | | | | | | | 1 | | | | | | | 2 | | | | | X | | 3 | | | | X | | | 4 | | | X | | | pos.getRow=2 pos.getColumn=4 player=X	checkDiagonalWin = true State of board is unchanged	This test case is unique because we are checking for a win in a smaller part of the board at the bottom right and we are going from bottom left to top right. Function Name: test_checkDiagonalWin_bottom_right_small_diagonal

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason:
State: (number to win = 3) pos.getRow=1 pos.getColumn=1 player=X	checkDiagonalWin = true State of board is unchanged	This test case is unique because the last marker we place is in the middle of two of the same markers so we have to check both directions of the diagonal in the top left of the board. This is an edge case. Function Name: test_checkDiagonalWin_top_left_place_marker_in_middle

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

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Input	Output	Reason:
State: (number to win = 3) pos.getRow=1 pos.getColumn=3 player=X	checkDiagonalWin = true State of board is unchanged	This test case is unique because we are placing the last marker in the middle of two of the same markers in the top right of the board. We will have to check in both directions. Where we are checking for the win is also why this test case is unique. Function Name: test_checkDiagonalWin_top_right_place_marker_in_middle

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

boolean checkForDrawWin()

Input	Output	Reason:
State: (number to win = 3)	checkForDraw = true State of board is unchanged	This test case is unique because we are checking for a draw when the board is full and there is not a winner. Function Name: test_checkForDraw_board_is_full_without_win

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

boolean checkForDrawWin()

Input	Output	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | | 0 | X | | | | 1 | | | | | 2 | | | |	checkForDraw = false State of board is unchanged	This test case is unique because we are testing to make sure after we place just one marker that we are not told that there is a draw. Function Name: test_checkForDraw_board_place_only_one_marker

boolean checkForDrawWin()

Input	Output	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | | 0 | X | O | X | | 1 | O | X | O | | 2 | O | X | |	checkForDraw = false State of board is unchanged	This test case is unique because we are testing to make sure that even though the board is one spot away from being full without a winner that we do not get told that there was a draw. Function Name: test_checkForDraw_board_place_only_one_marker

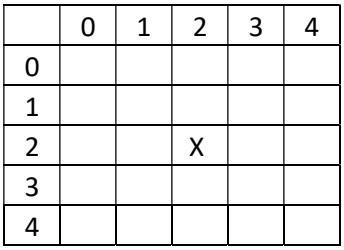
boolean checkForDrawWin()

Input	Output	Reason:
State: (number to win = 3) | | 0 | 1 | 2 | | 0 | X | O | X | | 1 | O | X | | | 2 | | | |	checkForDraw = false State of board is unchanged	This test case is unique because we are testing to make sure that even though the board is half full that we are not told that there is a draw Function Name: test_checkForDraw_board_place_only_one_marker

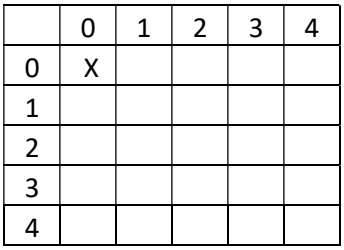
char whatsAtPos(BoardPosition pos)

Input	Output	Reason:
State: | | 0 | 1 | 2 | 3 | 4 | | 0 | | | | | | | 1 | | | | | | | 2 | | | | | | | 3 | | | | | | | 4 | | | | | | pos.getRow=2 pos.getColumn=2	whatsAtPos= empty space State of board is unchanged	This test case is unique because we are testing to make sure that whatsAtPos will return an empty space if there is not anything in that position. Function Name: test_whatsAtPos_empty_space

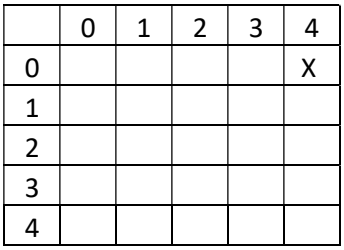
char whatsAtPos(BoardPosition pos)

Input State:  pos.getRow=2 pos.getColumn=2	Output whatsAtPos= X State of board is unchanged	Reason: The reason this test case is unique is because we are testing to see that whatsAtPos will return the correct marker is it's in the middle of board. Function Name: test_whatsAtPos_marker_in_middle_of_board

char whatsAtPos(BoardPosition pos)

Input State:  pos.getRow=0 pos.getColumn=0	Output whatsAtPos= X State of board is unchanged	Reason: The reason this test case is unique is because we are testing to that whatsAtPos will return the correct marker if the marker is in the top left corner. It is at the edge of the board so that is why it unique. Function Name: test_whatsAtPos_marker_top_left_corner

char whatsAtPos(BoardPosition pos)

Input State:  pos.getRow=0 pos.getColumn=4	Output whatsAtPos= X State of board is unchanged	Reason: The reason this test case is unique is because we are testing to that whatsAtPos will return the correct marker if the marker is in the top right corner. It is at the edge of the board so that is why it unique. Function Name: test_whatsAtPos_marker_top_right_corner

char whatsAtPos(BoardPosition pos)

Input State: pos.getRow=4 pos.getColumn=0	Output whatsAtPos= X State of board is unchanged	Reason: The reason this test case is unique is because we are testing to that whatsAtPos will return the correct marker if the marker is in the bottom left corner. It is at the edge of the board so that is why it unique. Function Name: test_whatsAtPos_marker_bottom_left_corner

	0	1	2	3	4
0					
1					
2					
3					
4	X				

char whatsAtPos(BoardPosition pos)

Input State: pos.getRow=4 pos.getColumn=0	Output whatsAtPos= X State of board is unchanged	Reason: The reason this test case is unique is because we are testing to that whatsAtPos will return the correct marker if the marker is in the bottom right corner. It is at the edge of the board so that is why it unique. Function Name: test_whatsAtPos_marker_bottom_right_corner

	0	1	2	3	4
0					
1					
2					
3					
4					X

char whatsAtPos(BoardPosition pos)

Input State: pos.getRow=0 pos.getColumn=2	Output whatsAtPos= O State of board is unchanged	Reason: The reason this test case is unique is because we are testing to that whatsAtPos will return the correct marker if there are different markers on the board. Function Name: test_whatsAtPos_O_Marker

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

void placeMarker(BoardPosition pos, char player)

Input
State:

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

player=O
pos.getRow=2
pos.getColumn=2

Output
State:

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

Reason:
The reason this test case is unique is because we are placing a marker which has not been placed on the board yet

Function Name:
test_placeMarker_place_new_player

void placeMarker(BoardPosition pos, char player)

Input
State:

	0	1	2	3	4
0					
1					
2					
3					
4					

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

Output
State:

	0	1	2	3	4
0	X				
1					
2					
3					
4					

Reason:
The reason this test case is unique is because we are testing placing a marker in the top left corner of the board which is an edge case.

Function Name:
test_placeMarker_place_top_left_corner

void placeMarker(BoardPosition pos, char player)

Input
State:

	0	1	2	3	4
0					
1					
2					
3					
4					

player=X
pos.getRow=4
pos.getColumn=0

Output
State:

	0	1	2	3	4
0					
1					
2					
3					
4	X				

Reason:
The reason this test case is unique is because we are testing placing a marker in the bottom left corner which is an edge case.

Function Name:
test_placeMarker_place_bottom_left_corner

void placeMarker(BoardPosition pos, char player)

Input State:	Output State:	Reason:
<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> player=X pos.getRow=4 pos.getColumn=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>X</td></tr></table>	The reason this test case is unique is because we are testing placing a marker in the bottom right corner of the board which is an edge case. Function Name: test_placeMarker_place_bottom_right_corner

void placeMarker(BoardPosition pos, char player)

Input State:	Output State:	Reason:
<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> player=X pos.getRow=0 pos.getColumn=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>X</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>	The reason this test case is unique is because we are testing placing a marker in the top right corner of the board which is an edge case. Function Name: test_placeMarker_place_top_right_corner