

CPSC 2150 Project Report

Owen Sullivan

Requirements Analysis

Functional Requirements:

1. As a player, I need to be able to start a new game so that I can play.
2. As a player, I need to be able to choose how many players should be able to play the game so that I can play with more than one other person if I wish.
3. As a player, I need to be able to be alerted if the number of players I have entered is invalid so that I can enter a valid input instead.
4. As a player, I need to be able to choose how what character should represent each player so that I can identify which player has played a token in a given position.
5. As a player, I need to be able to be alerted if the character I have chosen for a given player's token is already used by another player so that I can enter a different character instead.
6. As a player, I need to be able to choose how many rows the game board should have so that I can play whatever size board I wish.
7. As a player, I need to be able to be alerted if the number of rows I have entered is invalid so that I can enter a valid input instead.
8. As a player, I need to be able to choose how many columns the game board should have so that I can play whatever size board I wish.
9. As a player, I need to be able to be alerted if the number of columns I have entered is invalid so that I can enter a valid input instead.
10. As a player, I need to be able to choose how many tokens should be required in a row to win the game so that I can make the game more challenging.
11. As a player, I need to be able to be alerted if the number of tokens in a row I have entered is invalid so that I can enter a valid input instead.
12. As a player, I need to be able to choose whether I'd like to play a fast implementation of the game or a memory-efficient implementation of the game so that I can decide how my game experience should be.
13. As a player, I need to be able to be alerted if the character I have entered to select an implementation is invalid so that I can enter a valid input instead.
14. As a player, I need to be able to see which player's turn it is so that I can either take my turn or give control of the computer to my opponent.
15. As a player, I need to be able to see the game board so that I can tell what spaces have already been filled so that I can decide where to drop a token.
16. As a player, I need to be able to visually determine what column on the game board corresponds to what number column so that I can drop a token in the correct column.
17. As a player, I need to be able to enter the column number that I wish to drop a token in so that I can drop a token in the correct column.

18. As a player, I need to be able to tell if the result of a turn is the conclusion of the game or not so that I can determine whether to continue playing.
19. As a player, I need to be able to tell if a game has concluded due to a tie or due to either myself or my opponent winning so that I can determine if there is a winner.
20. As a player, I need to be able to opt to play the game again so that I can participate in a re-match.
21. As a player, I need to be able to be alerted if the character I have entered to select whether to play again is invalid so that I can enter a valid input instead.
22. As a player, I need to be able to be alerted if I have tried to drop a token in a column that does not exist so that I can instead drop a token in a column that does exist.
23. As a player, I need to be able to be alerted if I have tried to drop a token in a column that is already full so that I can instead drop a token in a column that is not already full.
24. As a player, I need to be able to drop tokens in such a way that they are arranged in a sequence of five horizontally so that I can win the game.
25. As a player, I need to be able to drop tokens in such a way that they are arranged in a sequence of five vertically so that I can win the game.
26. As a player, I need to be able to drop tokens in such a way that they are arranged in a sequence of five diagonally so that I can win the game.
27. As a player, I need to be able to tie the game by dropping a token in such a way that all the spaces on the game board are filled without either myself or my opponent winning so that I can prevent my opponent from winning.

Non-Functional Requirements

1. The game must be able to run on Unix and it must run as a command-line application.
2. The game must be written in Java.
3. The game must include a GameScreen class which will contain the program's main() method.
4. The GameScreen class must include methods that are responsible for getting user inputs for the number of players in a game, the tokens for each player in a game, the number of rows in on a game board, the number of columns on a game board, the number of consecutive tokens required to win a game, and the chosen implementation for a game.
5. The GameScreen class must include methods that are responsible for alternating turns between players, outputting whose turn is currently being played, accept input for the column that a player would like to drop a token in, and placing tokens in a player's specified column.
6. The GameScreen class must include a method that determines whether a player is trying to drop a token in a column that does not exist or is full, and that method must alert the player of the issue and ask them to try again.
7. The GameScreen class must include a method that checks whether the game has been completed by determining if there are any 5-in-a-row matches for a given player's token set or by determining that all available token slots have been filled.
8. The GameScreen class must include a method that outputs the results of a game if the game is concluded and asks the players if they'd like to play again (indicated by "Y" or "N"). If the players choose to play again, the program should display a blank game board and start from the beginning of the game.

9. The GameScreen class must include a method that prompts the player whose turn is next to enter a column number if the game has not concluded.
10. All program input and output must happen within the GameScreen class.
11. The BoardPosition class must keep track of a single space on the game board as accessed by the row number and column number. There must be variables for both row and column numbers.
12. The BoardPosition class must have only one constructor that takes two ints (row and column), and the constructor must be the only way to set the data in the class.
13. The BoardPosition class must have methods that return the row number and the column number.
14. The BoardPosition class must have an overridden equals() method that determines whether or not two BoardPosition objects are equal by checking if both their row and column values are equal.
15. The BoardPosition class must have an overridden toString() method that creates a string in the format "<row>,<column>".
16. There must be an IGameBoard interface that abstractly represents the actual game board. All of the attributes must be public static final, public methods implemented by a class which implements IGameBoard, or default methods.
17. The IGameBoard interface must contain constant variables to represent the minimum and maximum values for the number of rows, number of columns, and number of consecutive tokens required to win.
18. The IGameBoard interface must contain a default method called checkIfFree that determines if a specified column can accept another token.
19. The IGameBoard interface must contain a method called placeToken, which updates the game board with the specified player's token in the specified column, that will be implemented by a class that implements IGameBoard.
20. The IGameBoard interface must contain a default method called checkForWin that determines whether the last token placed in the specified column resulted in the game concluding with a win.
21. The IGameBoard interface must contain a default method called checkTie that determines whether all of the spaces on the game board have been filled.
22. The IGameBoard interface must contain a default method called checkHorizWin that determines whether the last token placed in the specified BoardPosition resulted in the winning number of tokens in a horizontal row for the specified player.
23. The IGameBoard interface must contain a default method called checkVertWin that determines whether the last token placed in the specified BoardPosition resulted in the winning number of tokens in a vertical column for the specified player.
24. The IGameBoard interface must contain a default method called checkDiagWin that determines whether the last token placed in the specified BoardPosition resulted in the winning number of tokens diagonally for the specified player.
25. The IGameBoard interface must contain a method called whatsAtPos, which determines what player's token (if any) is found at the specified position, that will be implemented by a class that implements IGameBoard.
26. The IGameBoard interface must contain a default method called isPlayerAtPos that determines whether the specified player's token is at the specified position.

27. The IGameBoard interface must contain a method called getNumRows, which returns the number of rows on the game board, that will be implemented by a class that implements IGameBoard.
28. The IGameBoard interface must contain a method called getNumColumns, which returns the number of columns on the game board, that will be implemented by a class that implements IGameBoard.
29. The IGameBoard interface must contain a method called getNumToWin, which returns the number of consecutive tokens required to win, that will be implemented by a class that implements IGameBoard.
30. There must be an abstract AbsGameBoard class that implements the IGameBoard interface and provides an overridden toString method which returns a string representing the current state of the game board.
31. There must be a GameBoard class that implements the IGameBoard interface and extends the AbsGameBoard class. All of the attributes in the class must be private or static and final.
32. The game board within the GameBoard class must be represented by a 2D array where [0][0] is the bottom left corner of the board and [8][6] is the top right.
33. The values of the 2D array within the GameBoard class must contain a ' ' (space) by default, until a player drops a token in that position.
34. When a player drops a token in a given space, that space within the 2D array within the GameBoard class should change to either an 'X' or an 'O' depending on which player drops the token.
35. A constructor for the GameBoard class should be provided with parameters for the number of rows, number of columns, and number of consecutive tokens required to win.
36. There must be a GameBoardMem class that implements the IGameBoard interface and extends the AbsGameBoard class. All of the attributes in the class must be private or static and final.
37. The game board within the GameBoardMem class must be represented by a map where the keys are type Character and the values are type List<BoardPosition>.
38. The key/value pairs of the map within the GameBoardMem class must only exist if a player has dropped a token at a position.
39. When a player drops a token in a given space, the map within the GameBoardMem class must be updated with the new key (if the key does not already exist within the map) and the new BoardPosition value corresponding to the player's token and the space that has been filled.
40. A constructor for the GameBoardMem class should be provided with parameters for the number of rows, number of columns, and number of consecutive tokens required to win.
41. The GameBoardMem class must override the default IGameBoard implementation of the isPlayerAtPos method in such a way that works for the game board represented as a map and does not rely on whatsAtPos.

Deployment Instructions

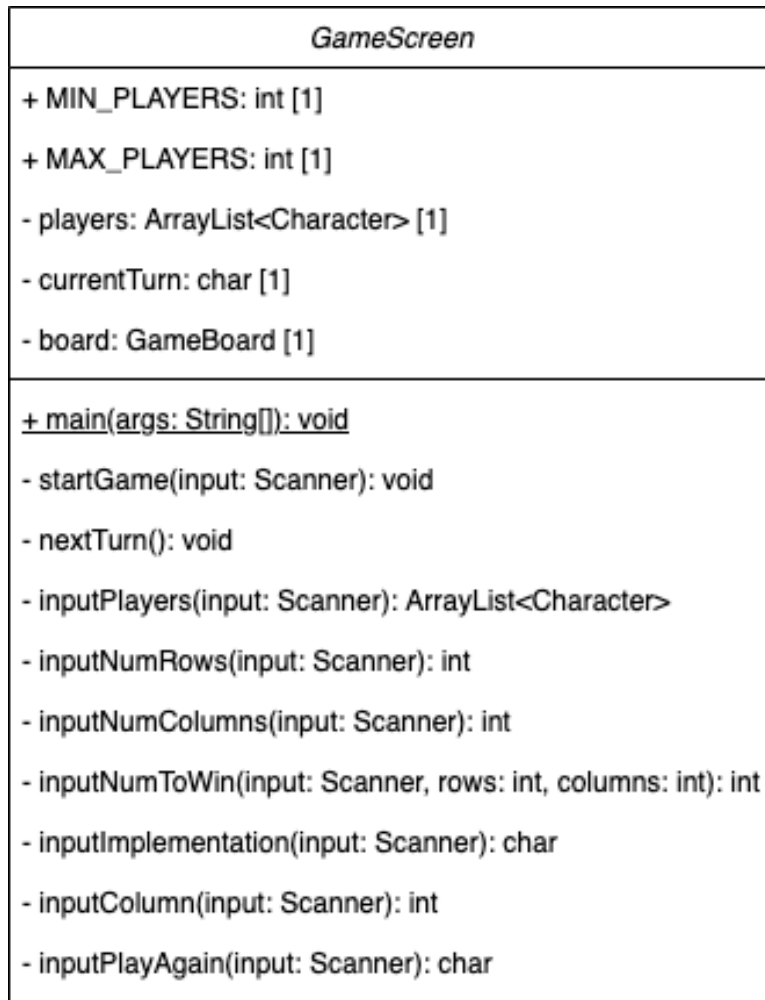
Details in Projects 2-5.

- To compile ExtendedConnectX, navigate to the project directory (ex: "ExtendedConnectX/src/") that contains the "cpssc2150" directory and the "makefile" file.
 - Run "make" or "make default" to compile the program.
- To run ExtendedConnectX, navigate to the project directory (ex: "ExtendedConnectX/src/") that contains the "cpssc2150" directory and the "makefile" file.
 - Run "make run" to run the compiled program.
 - If the program is not yet compiled or the source has changed since the last compilation, the program will re-compile before running.
- To remove the compiled ExtendedConnectX program files (.class), navigate to the project directory (ex: "ExtendedConnectX/src/") that contains the "cpssc2150" directory and the "makefile" file.
 - Run "make clean" to remove the compiled .class files.

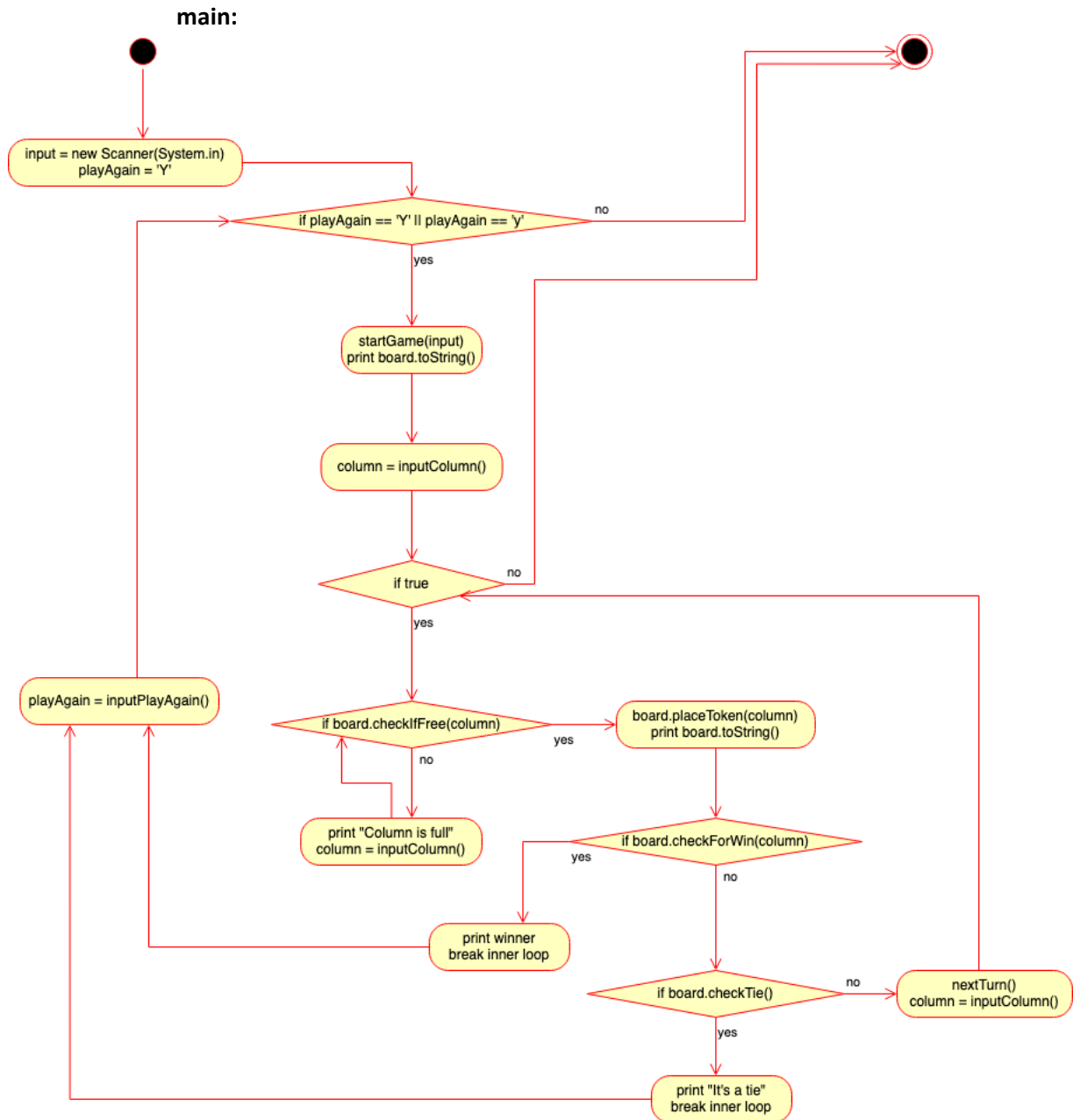
System Design

Class 1: GameScreen

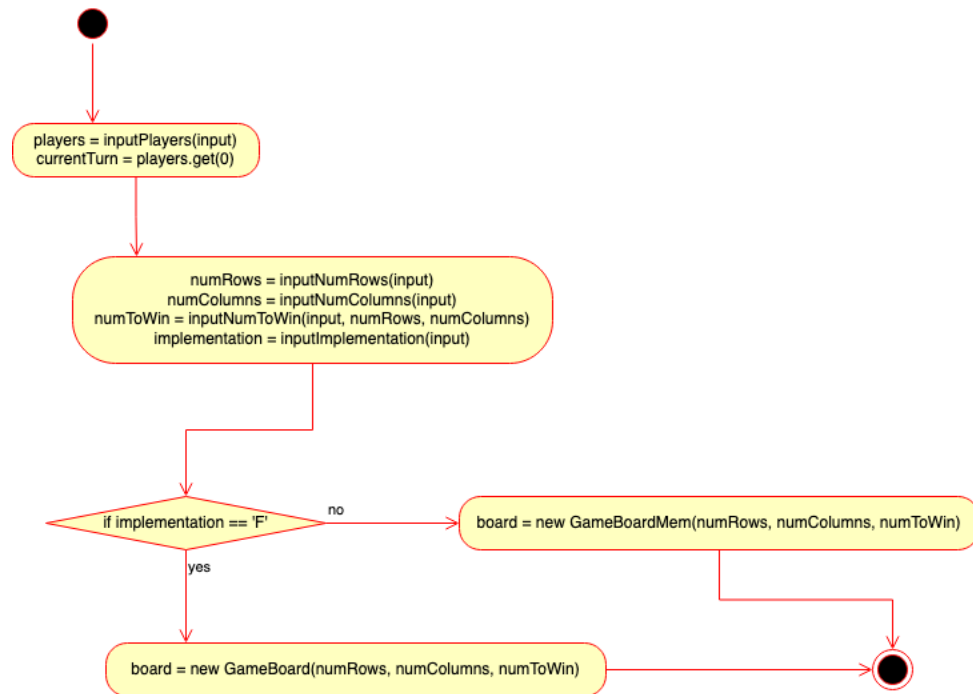
Class diagram



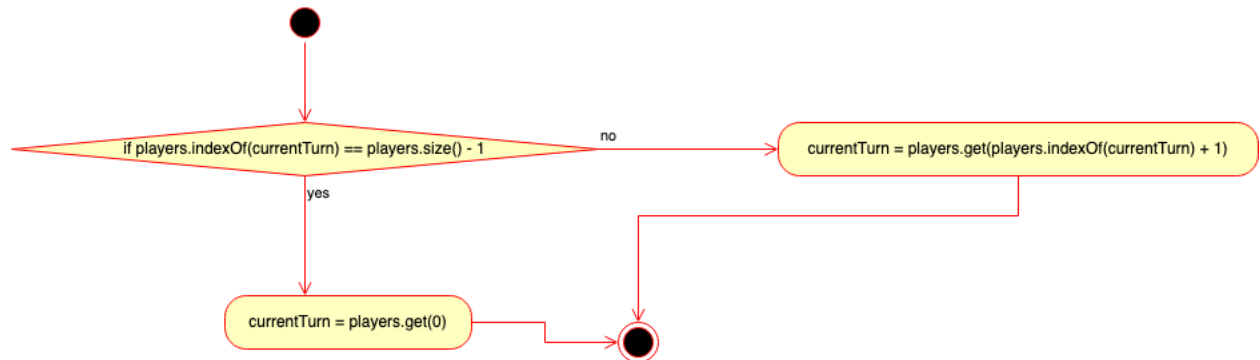
Activity diagrams



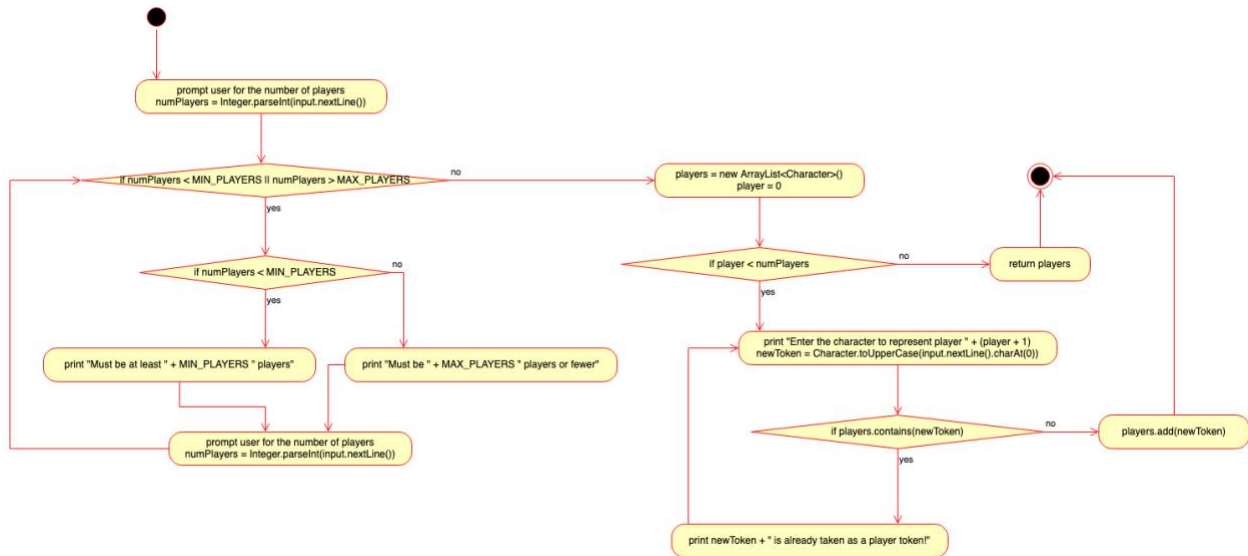
startGame:



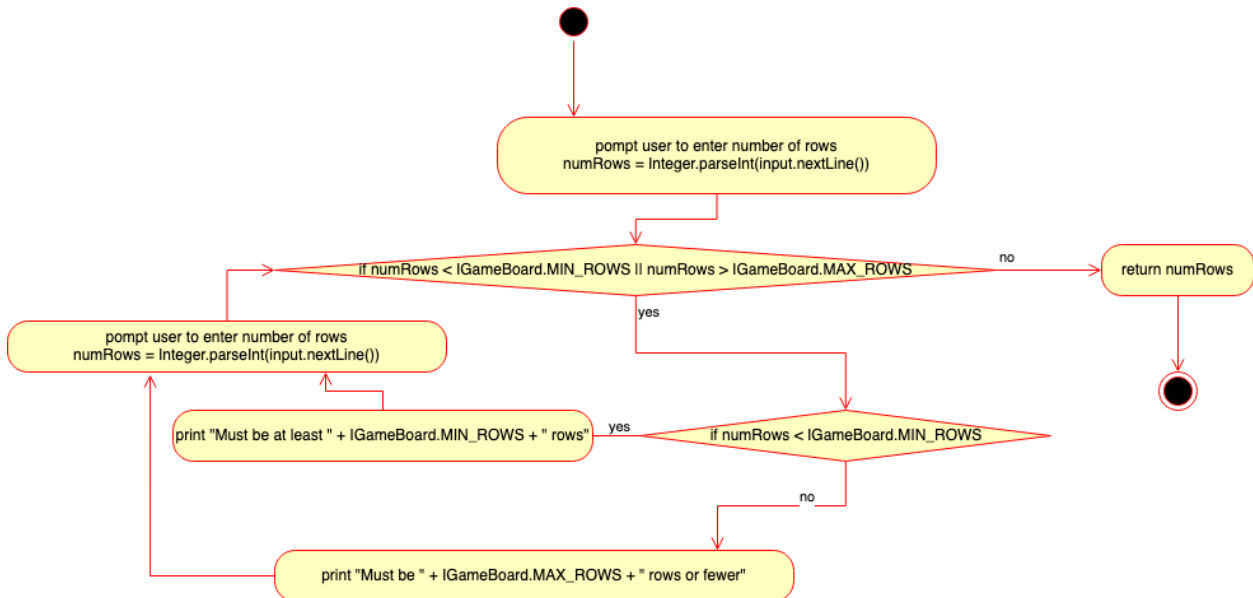
nextTurn:



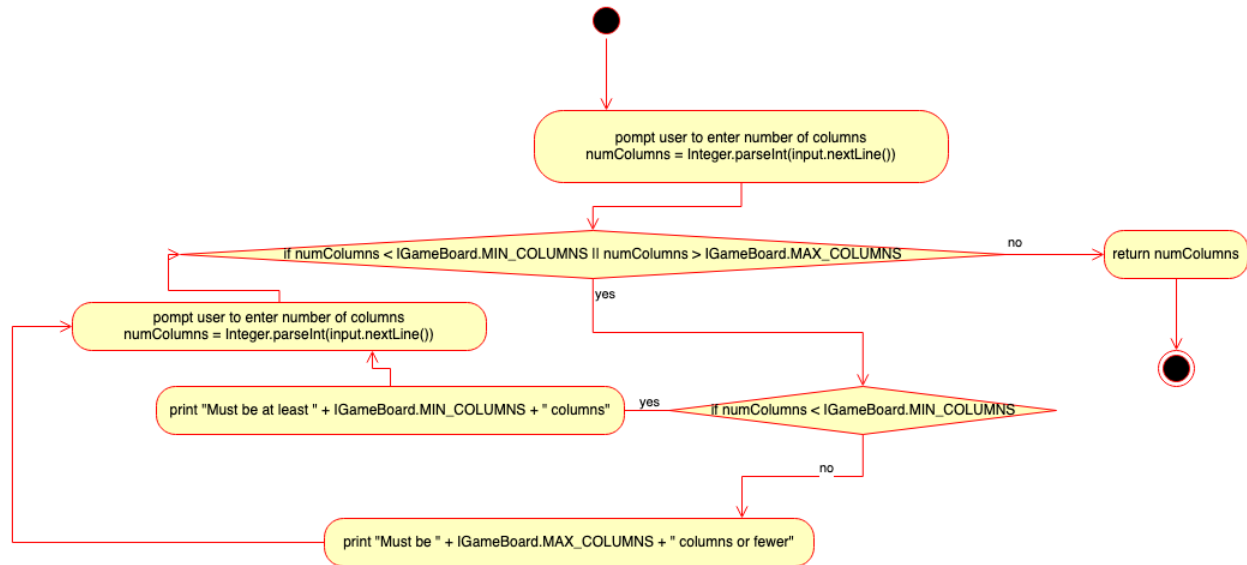
inputPlayers:



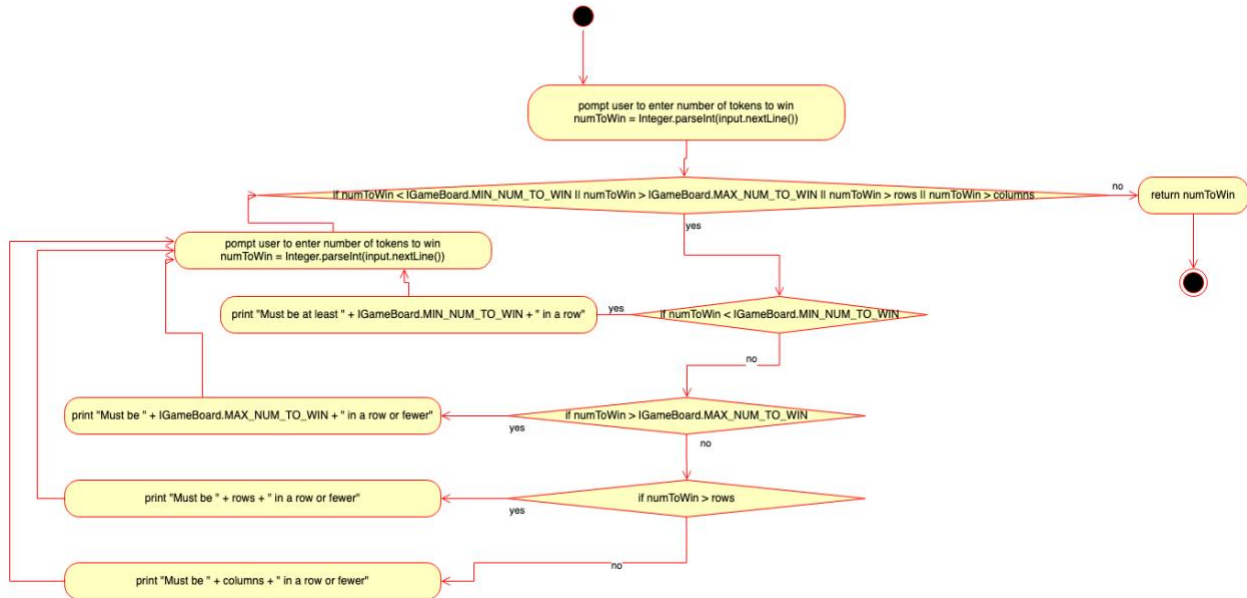
inputNumRows:



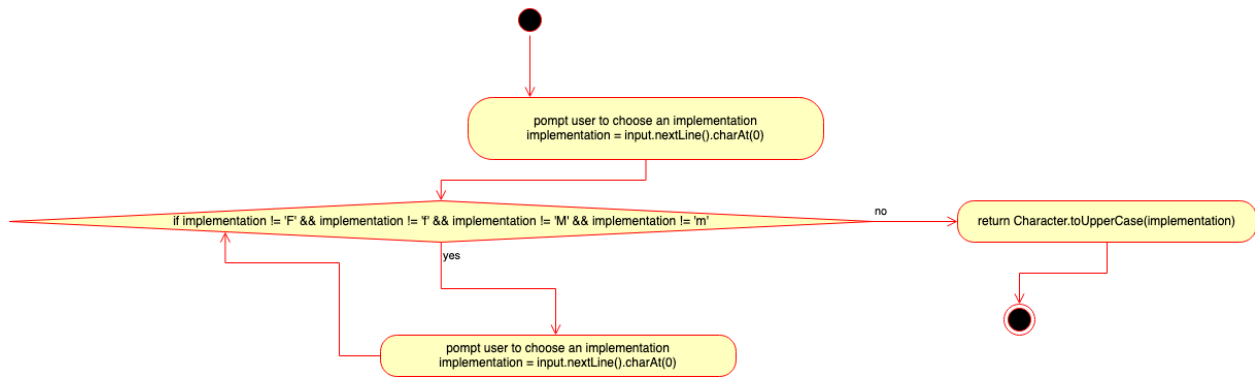
inputNumColumns:



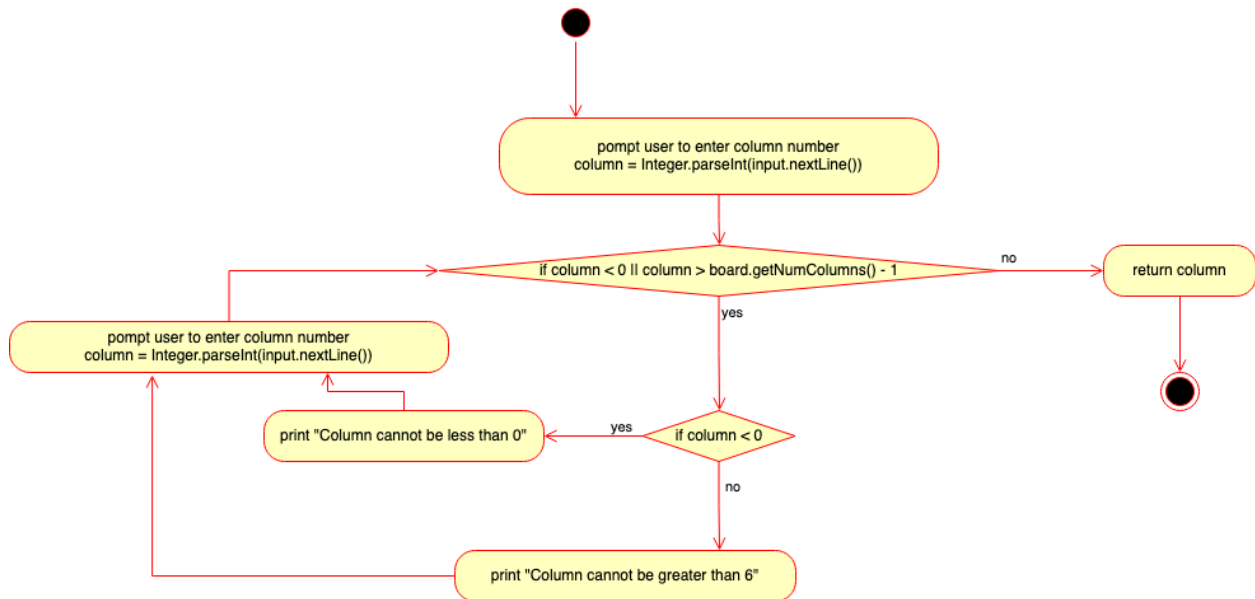
inputNumToWin:



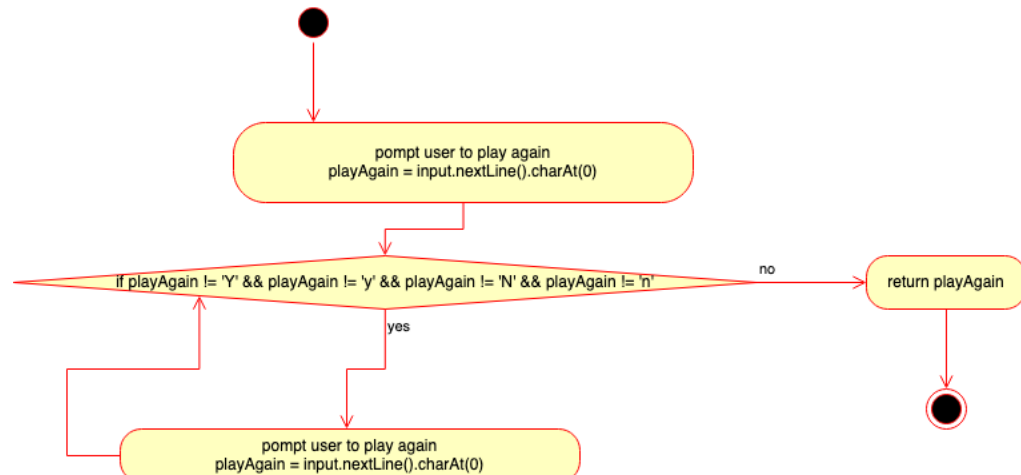
inputImplementation:



inputColumn:

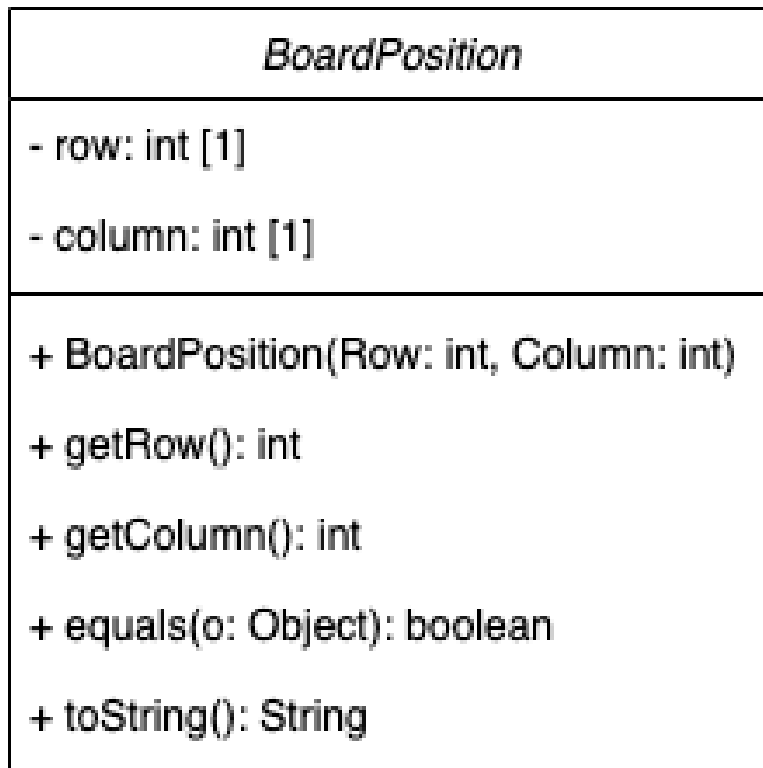


inputPlayAgain:



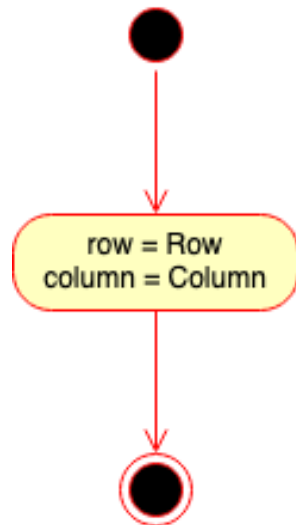
Class 2: BoardPosition

Class diagram

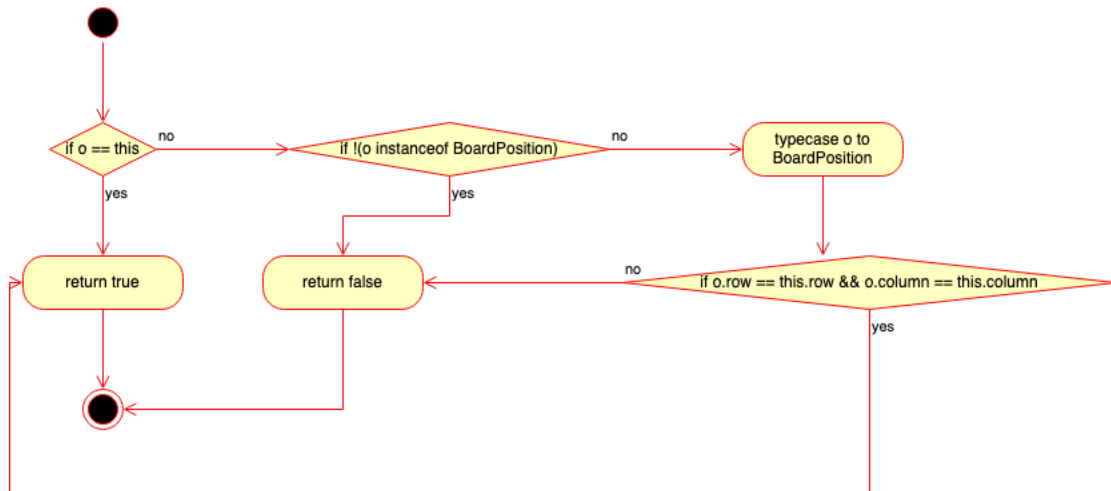


Activity diagrams

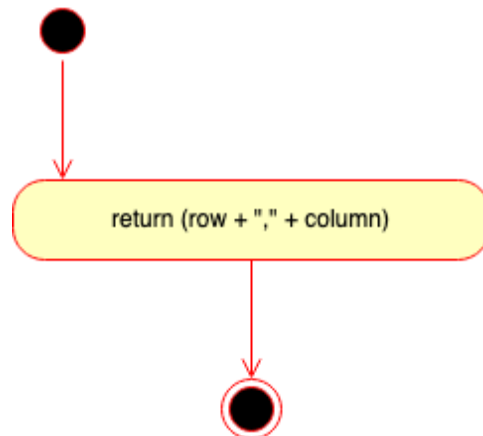
Constructor (BoardPosition):



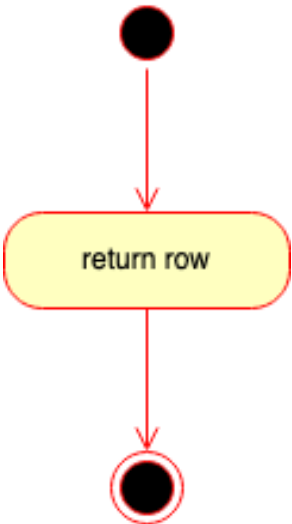
equals:



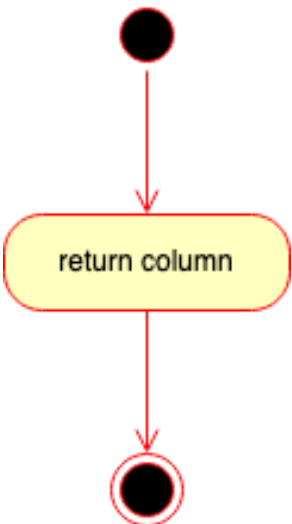
toString:



getRow:

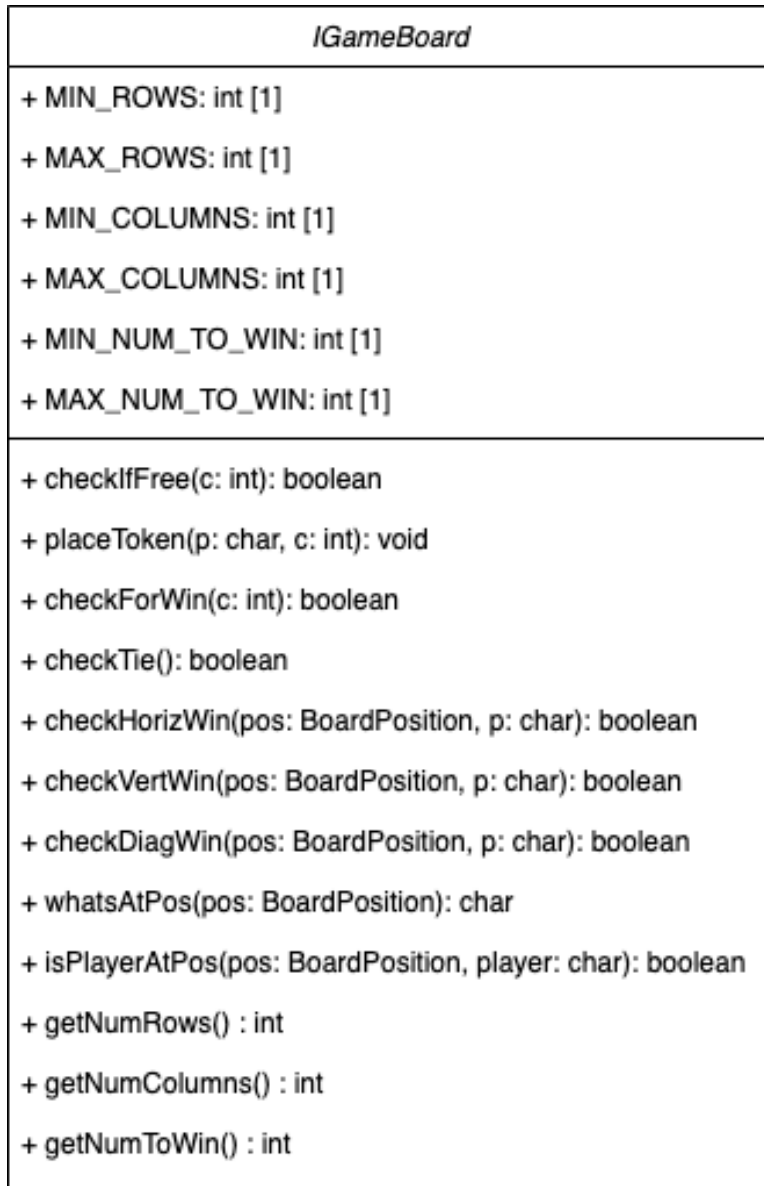


getColumn:



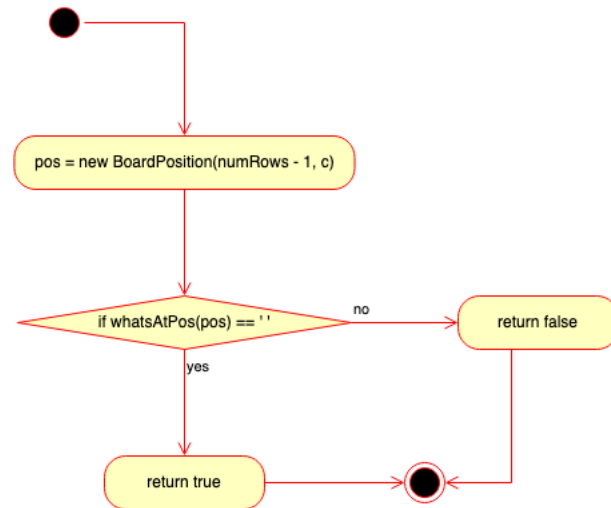
Interface 1: IGameBoard

Interface diagram

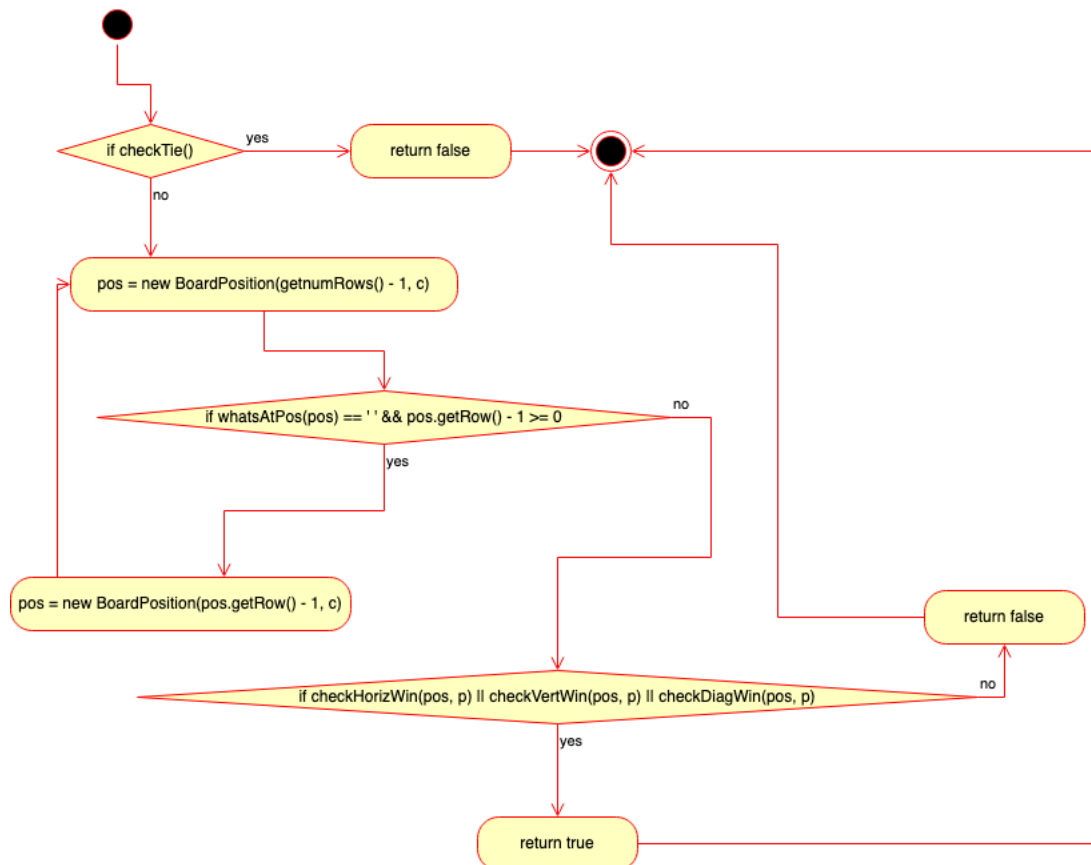


Activity diagrams

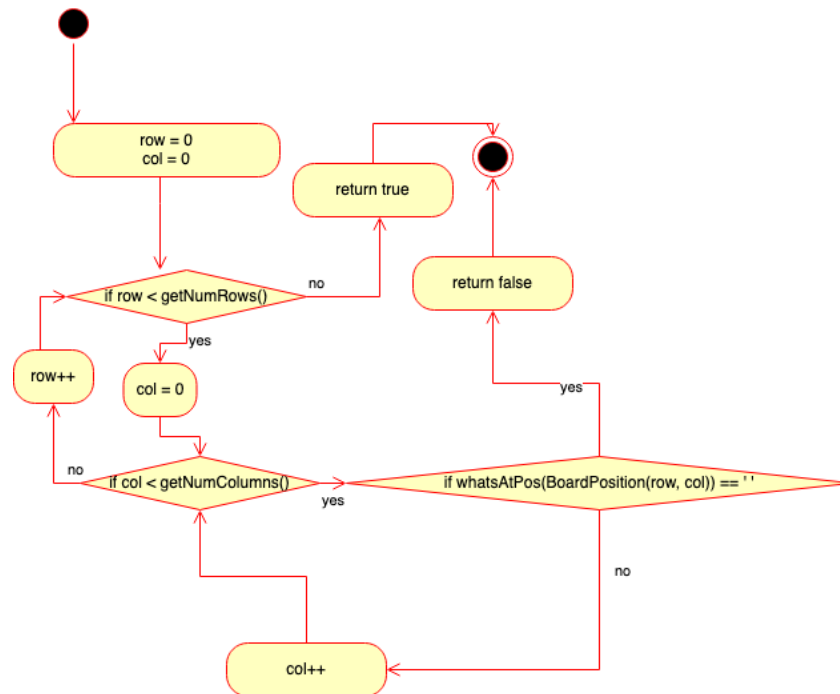
checkIfFree:



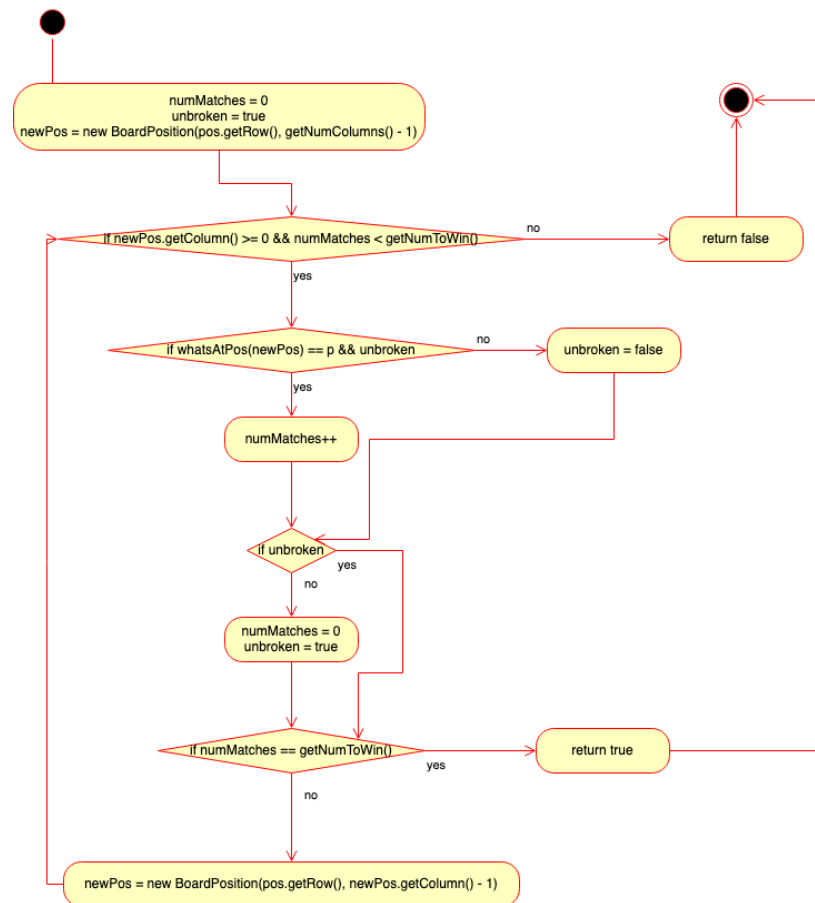
checkForWin:



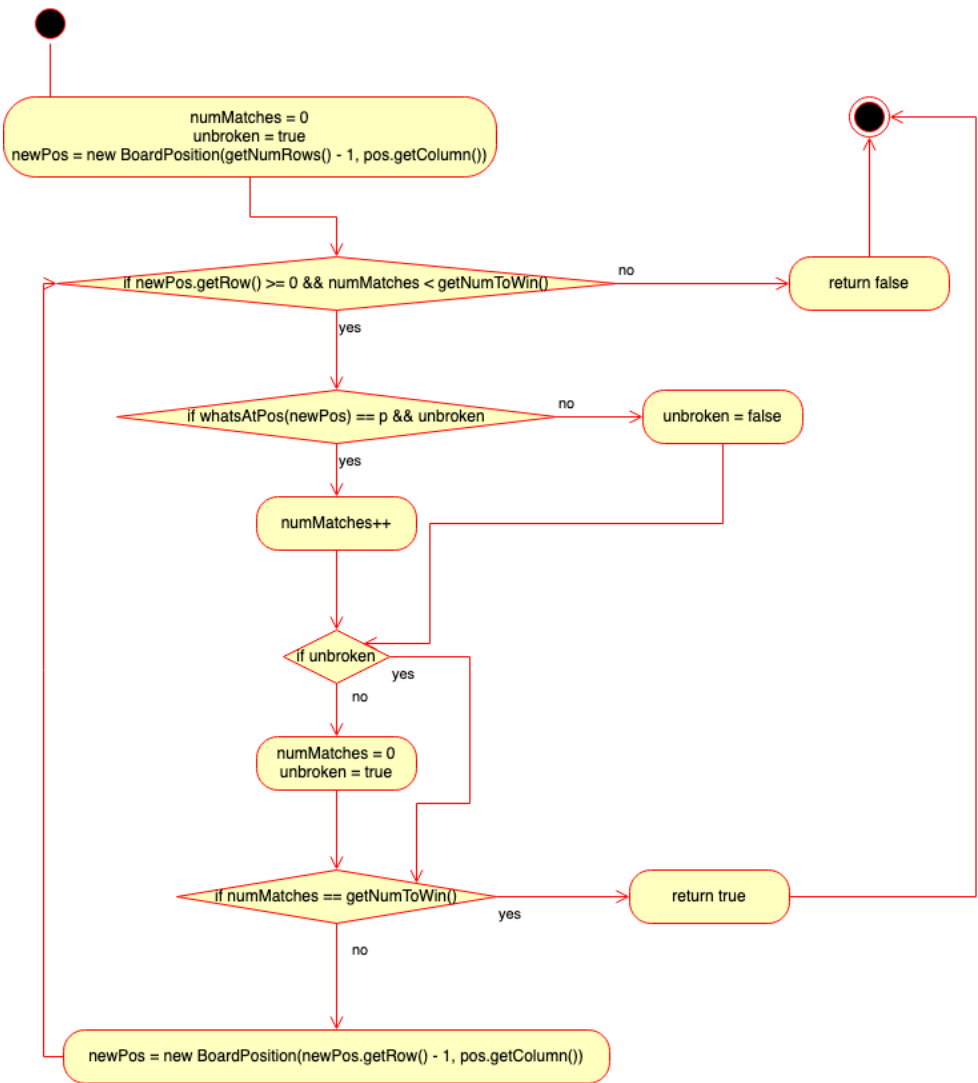
checkTie:



checkHorizWin:



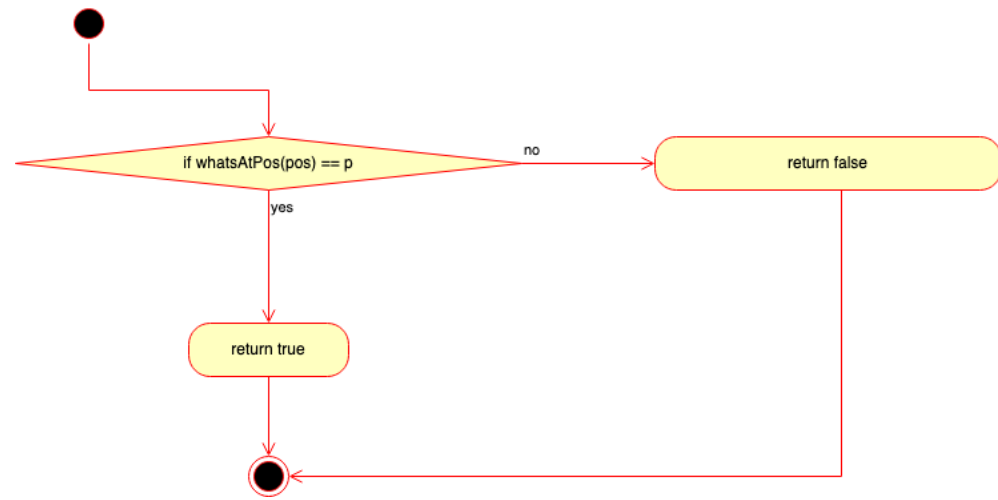
checkVertWin:



checkDiagWin:

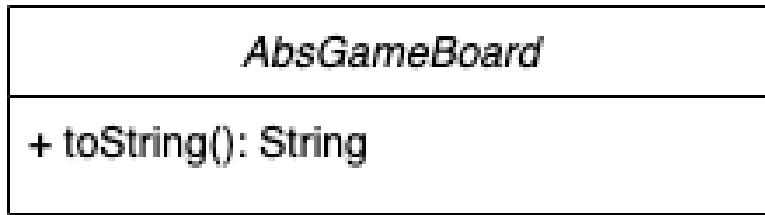


isPlayerAtPos:



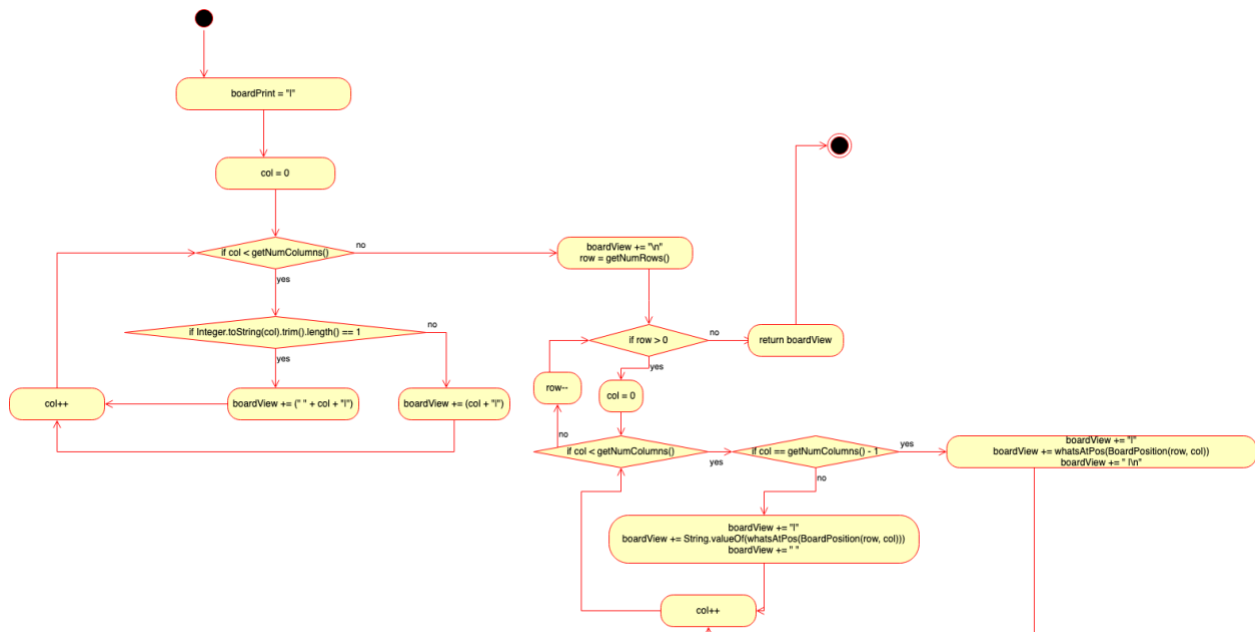
Class 3: AbsGameBoard

Class diagram



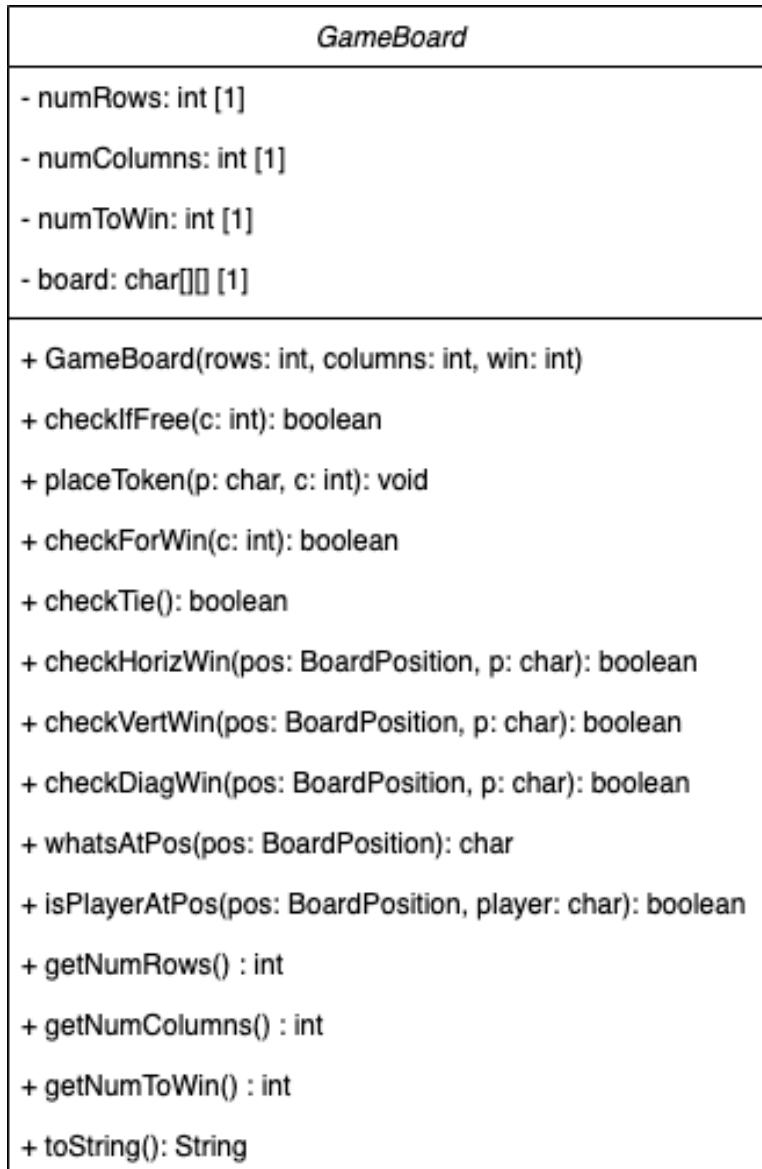
Activity diagrams

toString:



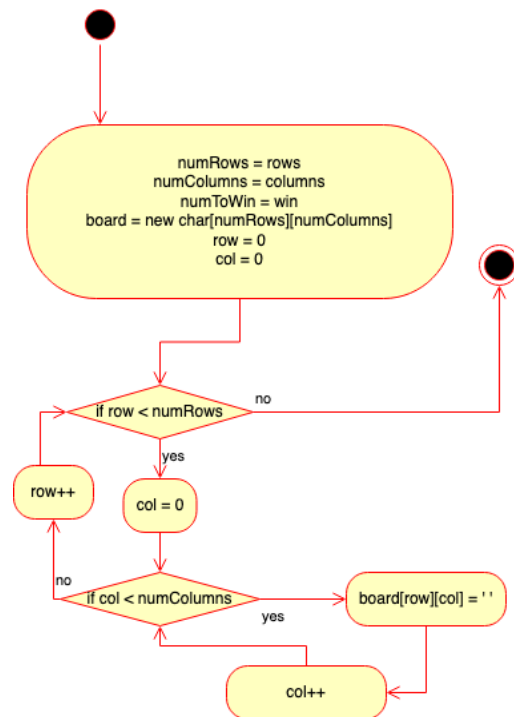
Class 4: GameBoard

Class diagram

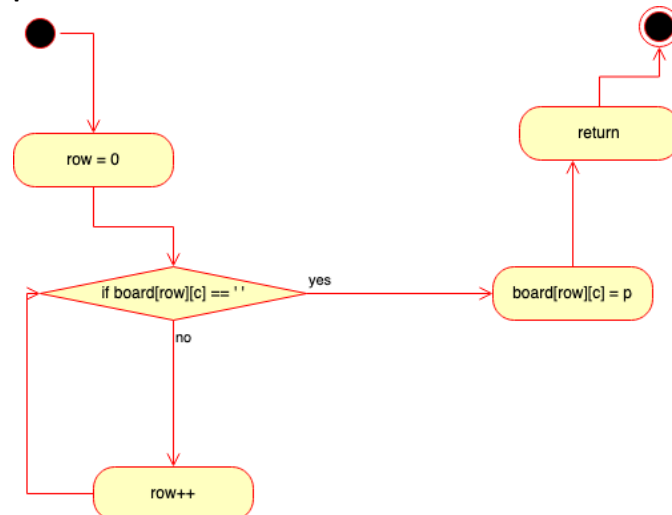


Activity diagrams

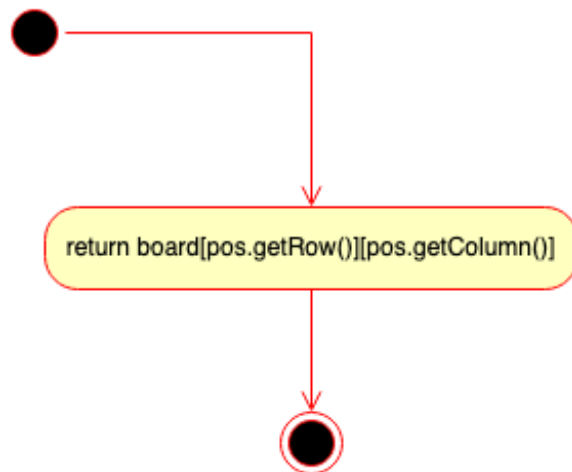
Constructor (GameBoard):



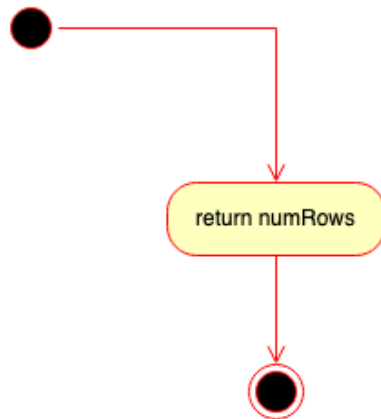
placeToken:



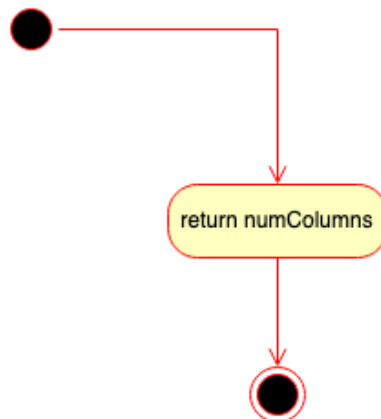
whatsAtPos:



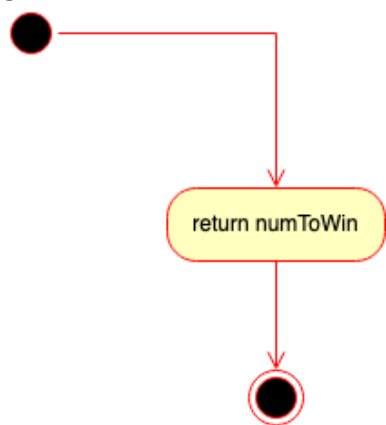
getNumRows:



getNumColumns:

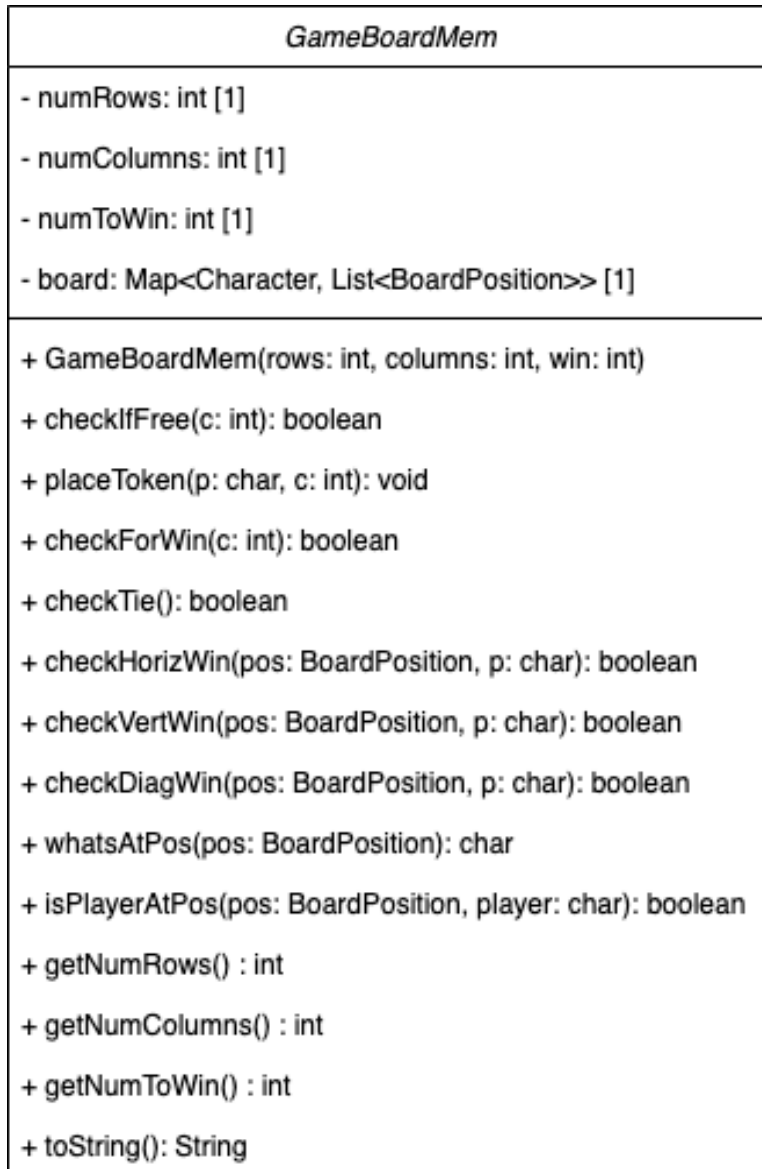


getNumToWin:



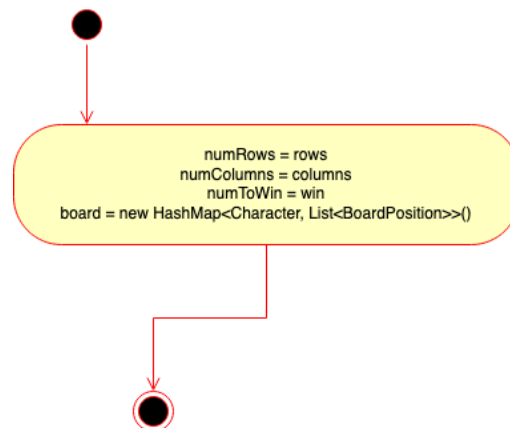
Class 4: GameBoardMem

Class diagram

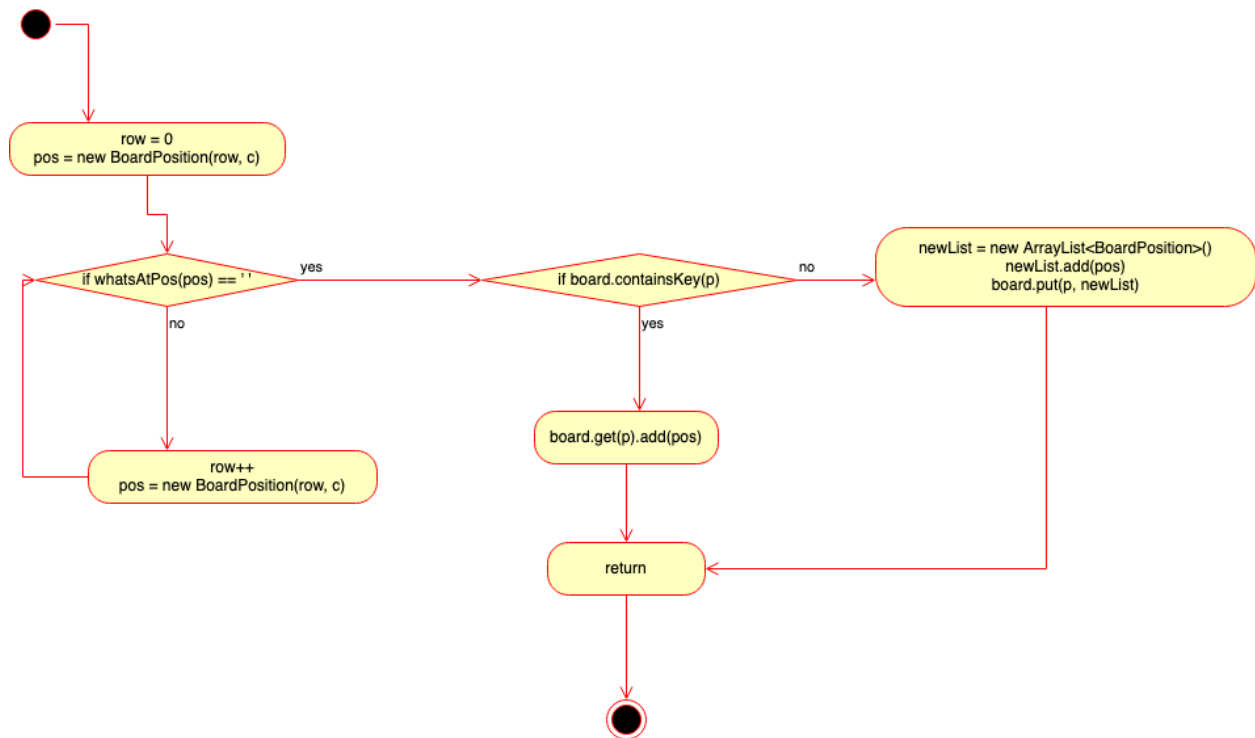


Activity diagrams

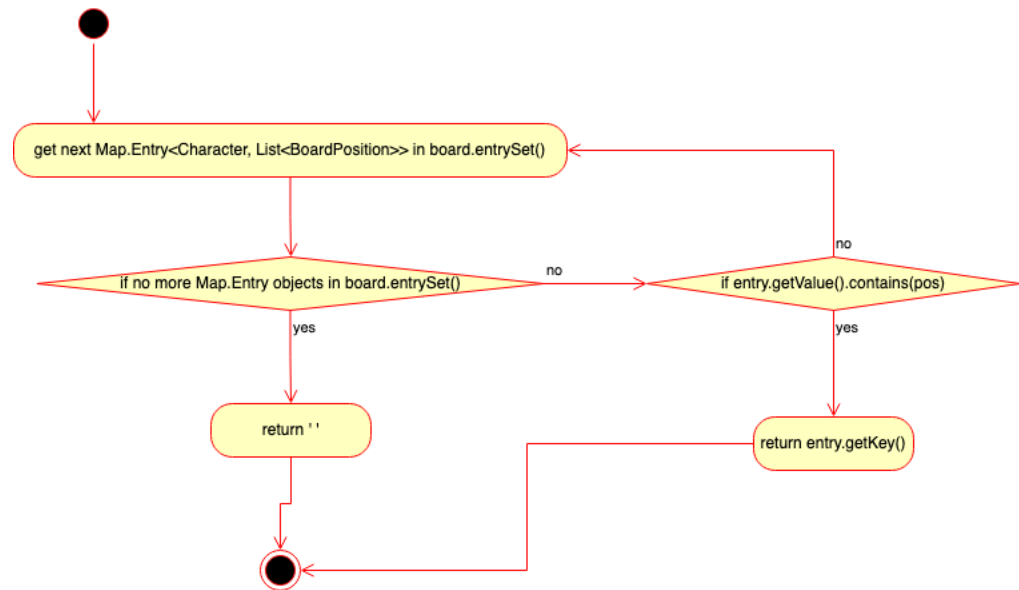
Constructor (GameBoardMem):



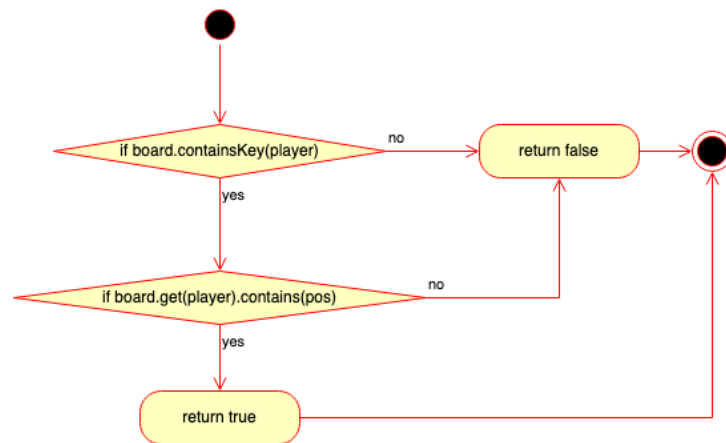
placeToken:



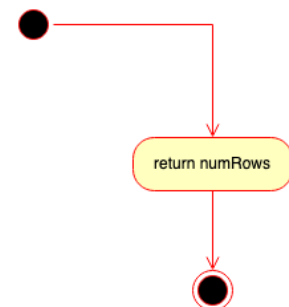
whatsAtPos:



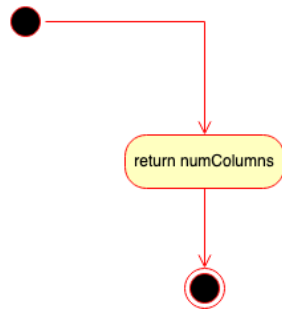
isPlayerAtPos:



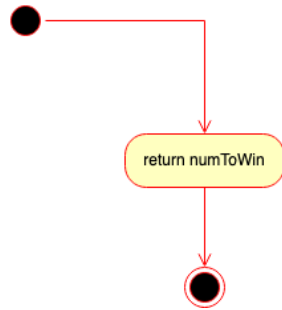
getNumRows:



getNumColumns:



getNumToWin:



Test Cases

GameBoard(int rows, int columns, int win)

GameBoardMem(int rows, int columns, int win)

Input: State: <table border="1" data-bbox="207 478 573 594"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </table> rows = IGameBoard.MIN_ROWS columns = IGameBoard.MIN_COLUMNS win = IGameBoard.MIN_NUM_TO_WIN										Output: Board is initialized with IGameBoard.MIN_ROWS rows, IGameBoard.MIN_COLUMNS columns, and IGameBoard.MIN_NUM_TO_WIN number to win State of the board is unchanged	Reason: This test case is unique and distinct because the GameBoard is being initialized to the minimum values specified in IGameBoard, which could be changed by the interface implementer. Function name: testConstructor_minimum_values
Input: State: 100x100 grid with uninitialized tokens (not shown due to table limits) rows = IGameBoard.MAX_ROWS columns = IGameBoard.MAX_COLUMNS win = IGameBoard.MAX_NUM_TO_WIN	Output: Board is initialized with IGameBoard.MAX_ROWS rows, IGameBoard.MAX_COLUMNS columns, and IGameBoard.MAX_NUM_TO_WIN number to win State of the board is unchanged	Reason: This test case is unique and distinct because the GameBoard is being initialized to the maximum values specified in IGameBoard, which could be changed by the interface implementer. Function name: testConstructor_maximum_values									
Input: State: [random]x[random] grid with uninitialized tokens (not shown due to inability to predict random number generation) rows = [random number between	Output: Board is initialized with a random number between IGameBoard.MIN_ROWS and IGameBoard.MAX_ROWS, columns is initialized with a random number between IGameBoard.MIN_COLUMNS and IGameBoard.MAX_COLUMNS, and the number to win is initialized with a random	Reason: This test case is unique and distinct because the GameBoard is being initialized to random values between the bounds specified in IGameBoard, which could be changed by the interface implementer, and it could theoretically test the constructor with any possible set of values. Function name: testConstructor_random_values									

IGameBoard.MIN_ROWS and IGameBoard.MAX_ROWS] columns = [random number between IGameBoard.MIN_COLUMNS and IGameBoard.MAX_COLUMNS] win = [random number between IGameBoard.MIN_NUM_TO_ WIN and IGameBoard.MAX_NUM_TO_ WIN]	number between IGameBoard.MIN_NUM_TO_ WIN and IGameBoard.MAX_NUM_TO_ _WIN. State of the board is unchanged	
--	--	--

```
boolean checkIfFree(int c)
```

Input: State: <table border="1" data-bbox="204 338 474 451"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </table> c = 0										Output: checkIfFree = true State of the board is unchanged	Reason: This test case is unique and distinct because checkIfFree is being tested when the column passed as a parameter is completely empty, which could highlight issues with checkIfFree if it is counting spaces as tokens for some reason. Function name: testCheckIfFree_column_empty
Input: State: <table border="1" data-bbox="204 693 474 806"> <tr><td>X</td><td></td><td></td></tr> <tr><td>X</td><td></td><td></td></tr> <tr><td>X</td><td></td><td></td></tr> </table> c = 0	X			X			X			Output: checkIfFree = false State of the board is unchanged	Reason: This test case is unique and distinct because checkIfFree is being tested when the column number passed as a parameter is completely full, which could highlight issues with checkIfFree if it is under-counting the number of tokens in a column for some reason. Function name: testCheckIfFree_column_full
X											
X											
X											
Input: State: <table border="1" data-bbox="204 1050 474 1163"> <tr><td></td><td></td><td></td></tr> <tr><td>X</td><td></td><td></td></tr> <tr><td>X</td><td></td><td></td></tr> </table> c = 0				X			X			Output: checkIfFree = true State of the board is unchanged	Reason: This test case is unique and distinct because checkIfFree is being tested when the column number passed as a parameter is only partially full, which could highlight issues with checkIfFree if it is over-counting the number of tokens in a column for some reason. Function name: testCheckIfFree_column_partially_full
X											
X											

boolean checkHorizWin(BoardPosition pos, char p)

Input: State: (number to win = 3) <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td></td></tr></table> pos.getRow = 0 pos.getCol = 2 p = 'X'													X	X	X		Output: checkHorizWin = true State of the board is unchanged	Reason: This test case is unique and distinct because checkHorizWin is being tested when there is an unbroken chain of the same token placed from the left-most column, with no other tokens placed before or after the chain, which could highlight problems with checkHorizWin if it is under-counting the number of unbroken tokens for some reason. Function name: testCheckHorizWin_unbroken_chain_left									
X	X	X																									
Input: State: (number to win = 3) <table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td>X</td><td>X</td><td></td></tr></table> pos.getRow = 0 pos.getCol = 3 p = 'X'																					O	X	X	X		Output: checkHorizWin = true State of the board is unchanged	Reason: This test case is unique and distinct because checkHorizWin is being tested when there is an unbroken chain of the same token placed after at least one token representing a different player, which could highlight problems with checkHorizWin if it is counting the other player's token for some reason. Function name: testCheckHorizWin_unbroken_chain_middle
O	X	X	X																								
Input: State: (number to win = 3) <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td>X</td><td>X</td></tr></table> pos.getRow = 0 pos.getCol = 1 p = 'X'														X	X	X	Output: checkHorizWin = true State of the board is unchanged	Reason: This test case is unique and distinct because checkHorizWin is being tested when there is an unbroken chain of the same token placed from the right-most column, with no other tokens placed before or after the chain, which could highlight problems with checkHorizWin if it is under-counting the number of unbroken tokens for some reason. Function name: testCheckHorizWin_unbroken_chain_right									
	X	X	X																								
Input: State: (number to win = 3) <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr></table>									Output: checkHorizWin = false State of the board is unchanged	Reason: This test case is unique and distinct because checkHorizWin is being tested when a token other than the first is placed before the last token in a chain, simulating what happens when another player intentionally cuts off a player's horizontal win ability,																	

<table><tr><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td>O</td><td>X</td><td></td></tr></table> pos.getRow = 0 pos.getCol = 2 p = 'X'					X	O	X			which could highlight problems with checkHorizWin if it is counting the other player's token for some reason. Function name: testCheckHorizWin_broken_chain
X	O	X								

```
boolean checkVertWin(BoardPosition pos, char p)
```

Input: State: (number to win = 3) <table><tr><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr></table> pos.getRow = 2 pos.getColumn = 0 p = 'X'					X				X				X				Output: checkVertWin = true State of the board is unchanged	Reason: This test case is unique and distinct because checkVertWin is being tested when there is an unbroken chain of the same token terminated at the bottom-most row, with no other tokens placed before or after the chain, which could highlight problems with checkVertWin if it is under-counting the number of unbroken tokens for some reason. Function name: testCheckVertWin_unbroken_chain_bottom				
X																						
X																						
X																						
Input: State: (number to win = 3) <table><tr><td>O</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td></tr></table> pos.getRow = 3 pos.getColumn = 0 p = 'X'	O				X				X				X				O				Output: checkVertWin = true State of the board is unchanged	Reason: This test case is unique and distinct because checkVertWin is being tested when there is an unbroken chain of the same token placed after at least one token representing a different player and before another token representing a different player, which is different than placing the winning token at the very top of the column, which could highlight problems with checkHorizWin if it is counting the other player's token for some reason. This unit test is valid for checkVertWin because it functions separately from GameScreen's normal checks after each token placement, and instead works even with other tokens placed out of order. Function name: testCheckVertWin_unbroken_chain_middle
O																						
X																						
X																						
X																						
O																						
Input: State: (number to win = 3) <table><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td></tr></table> pos.getRow = 3 pos.getColumn = 0 p = 'X'	X				X				X				O				Output: checkVertWin = true State of the board is unchanged	Reason: This test case is unique and distinct because checkVertWin is being tested when there is an unbroken chain of the same token terminated at the top-most row, with no other tokens placed before or after the chain, which could highlight problems with checkVertWin if it is under-counting the number of unbroken tokens for some reason. Function name: testCheckVertWin_unbroken_chain_top				
X																						
X																						
X																						
O																						

Input: State: (number to win = 3) <table border="1"><tr><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td></tr></table> pos.getRow = 2 pos.getColumn = 0 p = 'X'					X				O				X				Output: checkVertWin = false State of the board is unchanged	Reason: This test case is unique and distinct because checkVertWin is being tested when a token other than the first is placed before the last token in a chain, simulating what happens when another player intentionally cuts off a player’s vertical win ability, which could highlight problems with checkVertWin if it is counting the other player’s token for some reason. Function name: testCheckVertWin_broken_chain
X																		
O																		
X																		

boolean checkDiagWin(BoardPosition pos, char p)

Input: State: (number to win = 3) <table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table> pos.getRow = 1 pos.getCol = 1 p = 'X'										Output: checkDiagWin = false State of the board is unchanged	Reason: This test is unique and distinct because it tests checkDiagWin when the GameBoard is completely empty, which could highlight issues with checkDiagWin if it is having trouble working without any tokens available for some reason. Function name: testCheckDiagWin_empty_board							
Input: State: (number to win = 3) <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td></tr><tr><td></td><td>X</td><td>O</td><td></td></tr><tr><td>X</td><td>O</td><td>O</td><td></td></tr></table> pos.getRow = 2 pos.getCol = 2 p = 'X'							X			X	O		X	O	O		Output: checkDiagWin = true State of the board is unchanged	Reason: This test is unique and distinct because it tests checkDiagWin when an unbroken sequence of a player's token is found diagonally from left to right moving upwards, which could highlight issues with checkDiagWin if it is not properly counting tokens for some reason. Function name: testCheckDiagWin_unbroken_left_right_up
		X																
	X	O																
X	O	O																
Input: State: (number to win = 3) <table><tr><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td><td></td></tr><tr><td>X</td><td>O</td><td>X</td><td></td></tr><tr><td>O</td><td>X</td><td>O</td><td></td></tr></table> pos.getRow = 1 pos.getCol = 2 p = 'X'	X				O	X			X	O	X		O	X	O		Output: checkDiagWin = true State of the board is unchanged	Reason: This test is unique and distinct because it tests checkDiagWin when an unbroken sequence of a player's token is found diagonally from left to right moving downwards, which could highlight issues with checkDiagWin if it is not properly counting tokens for some reason. Function name: testCheckDiagWin_unbroken_left_right_down
X																		
O	X																	
X	O	X																
O	X	O																
Input: State: (number to win = 3) <table><tr><td></td><td></td><td></td><td>X</td></tr><tr><td></td><td></td><td>X</td><td>O</td></tr><tr><td></td><td>X</td><td>O</td><td>X</td></tr><tr><td></td><td>O</td><td>X</td><td>O</td></tr></table>				X			X	O		X	O	X		O	X	O	Output: checkDiagWin = true State of the board is unchanged	Reason: This test is unique and distinct because it tests checkDiagWin when an unbroken sequence of a player's token is found diagonally from right to left moving downwards, which could highlight issues with checkDiagWin if it is not properly counting tokens for some reason. Function name:
			X															
		X	O															
	X	O	X															
	O	X	O															

pos.getRow = 1 pos.getCol = 1 p = 'X'		testCheckDiagWin_unbroken_right_left_down																
Input: State: (number to win = 3) <table><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td>O</td><td>X</td><td></td></tr><tr><td></td><td>X</td><td>O</td><td>X</td></tr></table> pos.getRow = 0 pos.getCol = 3 p = 'X'						X				O	X			X	O	X	Output: checkDiagWin = true State of the board is unchanged	Reason: This test is unique and distinct because it tests checkDiagWin when an unbroken sequence of a player’s token is found diagonally from right to left moving upwards, which could highlight issues with checkDiagWin if it is not properly counting tokens for some reason. Function name: testCheckDiagWin_unbroken_right_left_up
	X																	
	O	X																
	X	O	X															
Input: State: (number to win = 3) <table><tr><td>X</td><td>O</td><td>X</td></tr><tr><td>O</td><td>X</td><td>O</td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table> pos.getRow = 2 pos.getCol = 2 p = 'X'	X	O	X	O	X	O	X	O	X	Output: checkDiagWin = true State of the board is unchanged	Reason: This test is unique and distinct because it tests checkDiagWin when a player’s token is the last token placed in the GameBoard, which could highlight issues with checkDiagWin if it is mistakenly failing to bother checking for a win if the board is full. Function name: testCheckDiagWin_unbroken_placed_last							
X	O	X																
O	X	O																
X	O	X																
Input: State: (number to win = 3) <table><tr><td>X</td><td></td><td></td></tr><tr><td>O</td><td>O</td><td></td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table> pos.getRow = 0 pos.getCol = 2 p = 'X'	X			O	O		X	O	X	Output: checkDiagWin = false State of the board is unchanged	Reason: This test is unique and distinct because it tests checkDiagWin when a player’s token is placed in a broken sequence (where another player’s token exists), which could highlight issues with checkDiagWin if it is incorrectly counting another player’s token as part of the current player’s win sequence for some reason. Function name: testCheckDiagWin_broken_sequence							
X																		
O	O																	
X	O	X																

boolean checkTie()

Input: State: <table><tr><td>X</td><td>X</td><td>X</td></tr><tr><td>X</td><td>X</td><td>X</td></tr><tr><td>X</td><td>X</td><td>X</td></tr></table>	X	X	X	X	X	X	X	X	X	Output: checkTie = true State of the board is unchanged	Reason: This test case is unique and distinct because it tests checkTie when all of the tokens on the board are the same, which could highlight issues with checkTie if it is looking for differences in tokens for some reason. Function name: testCheckTie_full_all_same_tokens
X	X	X									
X	X	X									
X	X	X									
Input: State: <table><tr><td>X</td><td>O</td><td>X</td></tr><tr><td>O</td><td>X</td><td>O</td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table>	X	O	X	O	X	O	X	O	X	Output: checkTie = true State of the board is unchanged	Reason: This test case is unique and distinct because it tests checkTie when all of the tokens on the board are not the same, which could highlight issues with checkTie if it is looking for all of the tokens to be the same for some reason. Function name: testCheckTie_full_different_tokens
X	O	X									
O	X	O									
X	O	X									
Input: State: <table><tr><td>X</td><td></td><td>X</td></tr><tr><td>X</td><td>X</td><td>X</td></tr><tr><td>X</td><td>X</td><td>X</td></tr></table>	X		X	X	X	X	X	X	X	Output: checkTie = false State of the board is unchanged	Reason: This test case is unique and distinct because it tests checkTie when all but one of the tokens on the board are filled, which could highlight issues with checkTie if it is mis-counting the number of full spaces for some reason. Function name: testCheckTie_one_empty
X		X									
X	X	X									
X	X	X									
Input: State: <table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										Output: checkTie = false State of the board is unchanged	Reason: This test case is unique and distinct because it tests checkTie when all of the tokens on the board are empty, which could highlight issues with checkTie if it is counting spaces as a valid token for some reason. Function name: testCheckTie_all_empty

```
char whatsAtPos(BoardPosition pos)
```

Input: State: <table border="1" data-bbox="204 340 440 453"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </table> pos.getRow = 1 pos.getCol = 1										Output: whatsAtPos = ' ' State of the board is unchanged	Reason: This test case is unique and distinct because it tests whatsAtPos when the space at pos is empty, which could highlight issues with whatsAtPos if it is not properly returning space characters for empty positions for some reason. Function name: testWhatsAtPos_empty_space
Input: State: <table border="1" data-bbox="204 701 440 814"> <tr><td>X</td><td>O</td><td>X</td></tr> <tr><td>O</td><td>X</td><td>O</td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table> pos.getRow = 2 pos.getCol = 0	X	O	X	O	X	O	X	O	X	Output: whatsAtPos = 'X' State of the board is unchanged	Reason: This test case is unique and distinct because it tests whatsAtPos with a position at the very top-left corner of the GameBoard, which could highlight issues with whatsAtPos if it is incorrectly offsetting the row number for some reason. Function name: testWhatsAtPos_top_left_corner
X	O	X									
O	X	O									
X	O	X									
Input: State: <table border="1" data-bbox="204 1068 440 1182"> <tr><td>X</td><td>O</td><td>X</td></tr> <tr><td>O</td><td>X</td><td>O</td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table> pos.getRow = 0 pos.getCol = 0	X	O	X	O	X	O	X	O	X	Output: whatsAtPos = 'X' State of the board is unchanged	Reason: This test case is unique and distinct because it tests whatsAtPos with a position at the very bottom-left corner of the GameBoard, which could highlight issues with whatsAtPos if it is incorrectly offsetting the column number for some reason. Function name: testWhatsAtPos_bottom_left_corner
X	O	X									
O	X	O									
X	O	X									
Input: State: <table border="1" data-bbox="204 1457 440 1570"> <tr><td>X</td><td>O</td><td>X</td></tr> <tr><td>O</td><td>X</td><td>O</td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table> pos.getRow = 2 pos.getCol = 2	X	O	X	O	X	O	X	O	X	Output: whatsAtPos = 'X' State of the board is unchanged	Reason: This test case is unique and distinct because it tests whatsAtPos with a position at the very top-right corner of the GameBoard, which could highlight issues with whatsAtPos if it is incorrectly offsetting the row number for some reason. Function name: testWhatsAtPos_top_right_corner
X	O	X									
O	X	O									
X	O	X									
Input: State: <table border="1" data-bbox="204 1824 440 1898"> <tr><td>X</td><td>O</td><td>X</td></tr> <tr><td>O</td><td>X</td><td>O</td></tr> </table>	X	O	X	O	X	O	Output: whatsAtPos = 'X' State of the board is unchanged	Reason: This test case is unique and distinct because it tests whatsAtPos with a position at the very bottom-right corner of the GameBoard, which could highlight issues with whatsAtPos if it is			
X	O	X									
O	X	O									

<table><tr><td>X</td><td>O</td><td>X</td></tr></table> pos.getRow = 0 pos.getCol = 2	X	O	X		incorrectly offsetting the column number for some reason. Function name: testWhatsAtPos_bottom_right_corner
X	O	X			

```
boolean isPlayerAtPos(BoardPosition pos, char player)
```

Input: State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </table> pos.getRow = 0 pos.getCol = 0 player = 'X'										Output: isPlayerAtPos = false State of the board is unchanged	Reason: This test is unique and distinct because it tests isPlayerAtPos when the GameBoard is completely empty, which could highlight issues with isPlayerAtPos if it is incorrectly trying to account for an empty board by making up a token for some reason. Function name: testIsPlayerAtPos_empty_board
Input: State: <table border="1"> <tr><td>X</td><td>O</td><td>X</td></tr> <tr><td>O</td><td>X</td><td>O</td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table> pos.getRow = 0 pos.getCol = 0 player = 'X'	X	O	X	O	X	O	X	O	X	Output: isPlayerAtPos = true State of the board is unchanged	Reason: This test is unique and distinct because it tests isPlayerAtPos when the GameBoard is completely full, which could highlight issues with isPlayerAtPos if it is incorrectly trying to account for a full board by clearing the board for some reason. Function name: testIsPlayerAtPos_full_board
X	O	X									
O	X	O									
X	O	X									
Input: State: <table border="1"> <tr><td>X</td><td></td><td>X</td></tr> <tr><td>O</td><td>X</td><td>O</td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table> pos.getRow = 2 pos.getCol = 1 player = 'X'	X		X	O	X	O	X	O	X	Output: isPlayerAtPos = false State of the board is unchanged	Reason: This test is unique and distinct because it tests isPlayerAtPos when the token immediately below the specified position is the same as the token being checked for, which could highlight issues with isPlayerAtPos if it is defaulting to the next-lowest instance of the player's token for some reason. Function name: testIsPlayerAtPos_player_immediately_below
X		X									
O	X	O									
X	O	X									
Input: State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td>X</td><td></td><td></td></tr> </table> pos.getRow = 0 pos.getCol = 0 player = 'X'							X			Output: isPlayerAtPos = true State of the board is unchanged	Reason: This test is unique and distinct because it tests isPlayerAtPos when the token at pos is the player's token and is the only token on the GameBoard, which could highlight issues with isPlayerAtPos if it is failing to find the correct token for some reason. Function name: testIsPlayerAtPos_player_is_only_token
X											
Input:	Output:	Reason:									

State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td>X</td><td></td><td></td></tr> </table> pos.getRow = 0 pos.getCol = 0 player = 'O'							X			isPlayerAtPos = false State of the board is unchanged	This test is unique and distinct because it tests isPlayerAtPos when the token at pos is not the player's token, which could highlight issues with isPlayerAtPos if it is incorrectly returning true when the wrong token is found for some reason. Function name: testIsPlayerAtPos_not_correct_token
X											

```
void placeToken(char p, int c)
```

Input: State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td>X</td><td>O</td><td></td></tr> </table> p = 'X' c = 2							X	O		Output: State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table>							X	O	X	Reason: This test case is unique and distinct because it tests placeToken with a column that is empty, which could highlight issues with placeToken if it is not correctly finding the bottom space in an empty column for some reason. Function name: testPlaceToken_bottom_row
X	O																			
X	O	X																		
Input: State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table> p = 'X' c = 0							X	O	X	Output: State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td>X</td><td></td><td></td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table>				X			X	O	X	Reason: This test case is unique and distinct because it tests placeToken with a column that has only one token in it already, which could highlight issues with placeToken if it is not correctly finding the next available space in the column for some reason. Function name: testPlaceToken_only_one_in_row
X	O	X																		
X																				
X	O	X																		
Input: State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td>X</td><td></td><td></td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table> p = 'O' c = 0				X			X	O	X	Output: State: <table border="1"> <tr><td>O</td><td></td><td></td></tr> <tr><td>X</td><td></td><td></td></tr> <tr><td>X</td><td>O</td><td>X</td></tr> </table>	O			X			X	O	X	Reason: This test case is unique and distinct because it tests placeToken with a column that only has one free token space left, which could highlight issues with placeToken if it is incorrectly treating a column as full for some reason. Function name: testPlaceToken_row_almost_full
X																				
X	O	X																		
O																				
X																				
X	O	X																		
Input: State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> </table> p = 'X' c = 1										Output: State: <table border="1"> <tr><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td></tr> <tr><td></td><td>X</td><td></td></tr> </table>								X		Reason: This test case is unique and distinct because it tests placeToken with a GameBoard that is completely empty, which could highlight issues with placeToken if it is incorrectly checking to make sure the board is not empty for some reason. Function name: testPlaceToken_empty_board
	X																			
Input: State:	Output: State:	Reason: This test case is unique and distinct because it tests placeToken with a GameBoard that is																		

X		X	X	O	X	almost completely full, which could highlight issues with placeToken if it is incorrectly checking to make sure placing a token does not result in a full board for some reason. Function name: testPlaceToken_board_almost_full
O	X	O	O	X	O	
X	O	X	X	O	X	
p = 'O' c = 1						