

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 see which player's turn it is so that I can either take my turn or give control of the computer to my opponent.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. As a player, I need to be able to opt to play the game again so that I can participate in a re-match.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. All program input and output must happen within the GameScreen class.
10. 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.
11. 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.
12. The BoardPosition class must have methods that return the row number and the column number.
13. 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.
14. The BoardPosition class must have an overridden toString() method that creates a string in the format "<row>,<column>".
15. There must be a GameBoard class that represents the actual game board. All of the attributes in the class must be private or static and final.
16. 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.
17. 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.
18. 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.
19. A constructor for the GameBoard class should be provided with no parameters.
20. The GameBoard class must include a method called checkIfFree() that checks to see if a column can accept another token.
21. The GameBoard class must include a method called placeToken() that places a character in the lowest available row in a column.
22. The GameBoard class must include a method called checkForWin() that checks to see if the last token placed in a column resulted in a player winning the game.

23. The GameBoard class must include a method called checkTie() that checks to see if the game has resulted in a tie.
24. The GameBoard class must include a method called checkHorizWin() that checks to see if the last token placed by a player resulted in 5-in-a-row horizontally.
25. The GameBoard class must include a method called checkVertWin() that checks to see if the last token placed by a player resulted in 5-in-a-row vertically.
26. The GameBoard class must include a method called checkDiagWin() that checks to see if the last token placed by a player resulted in 5-in-a-row diagonally.
27. The GameBoard class must include a method called whatsAtPos() that returns either the character representing the player whose token is in the space or a space character if no token has been placed in that space.
28. The GameBoard class must include a method called isPlayerAtPos() that checks whether or not a player's token is at a space.
29. The GameBoard class must have an overridden toString() method that returns a string showing the entire game board in its current state.

Deployment Instructions

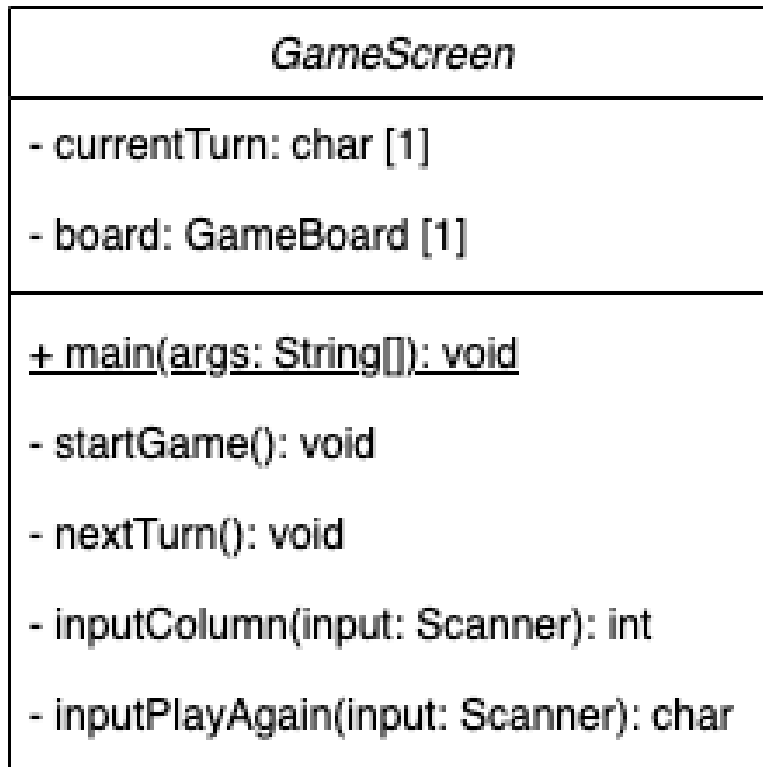
Details in Projects 2-5.

- 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.

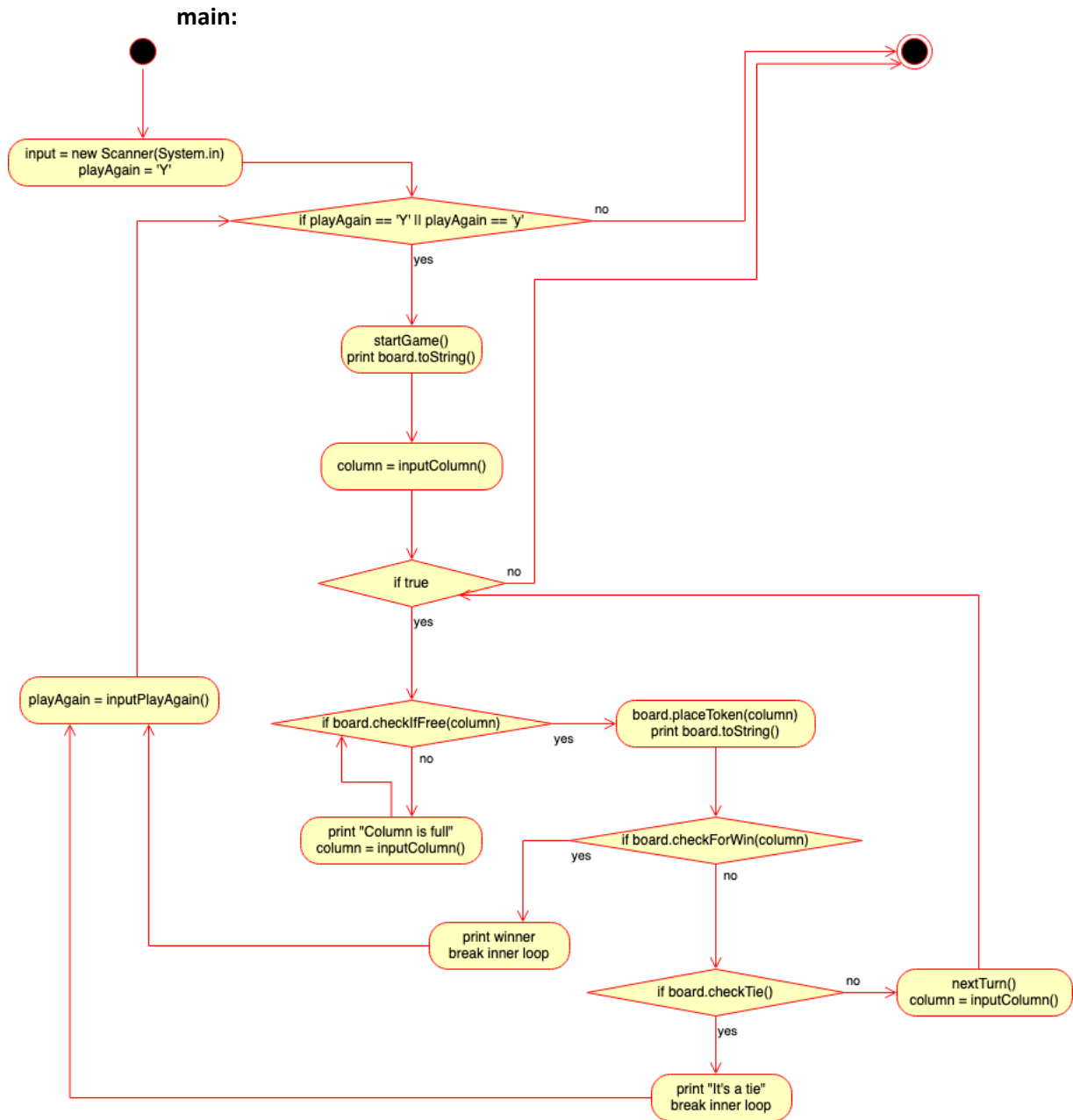
System Design

Class 1: GameScreen

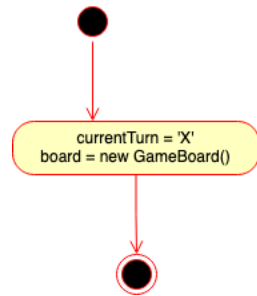
Class diagram



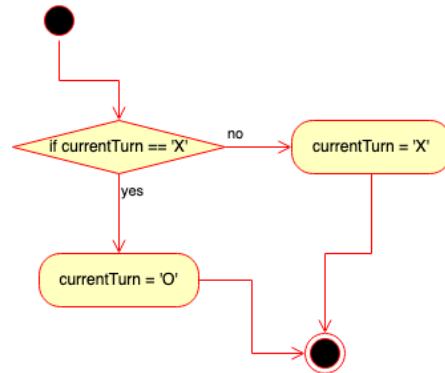
Activity diagrams



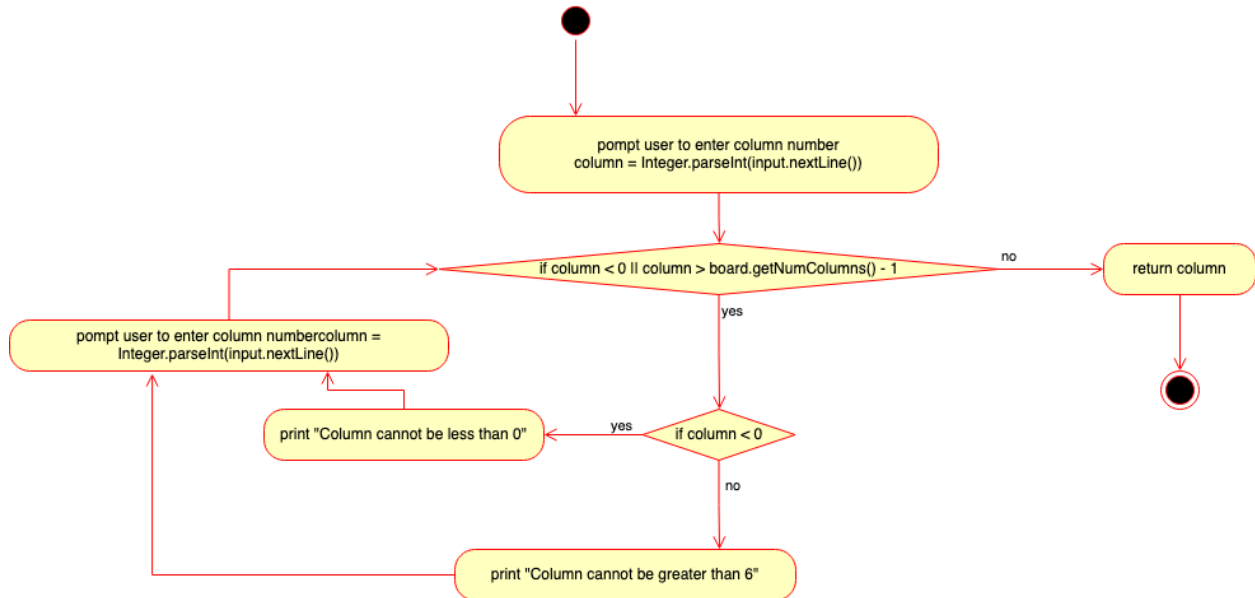
startGame:



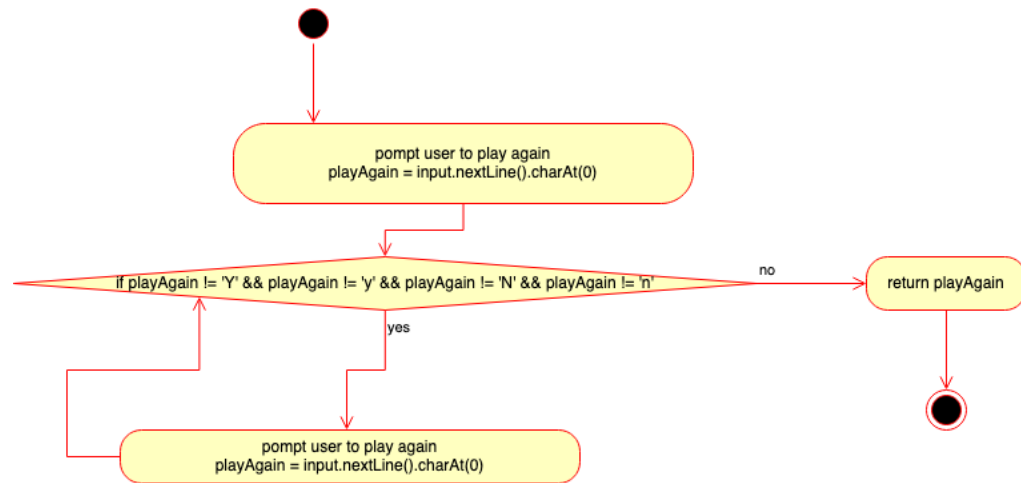
nextTurn:



inputColumn:

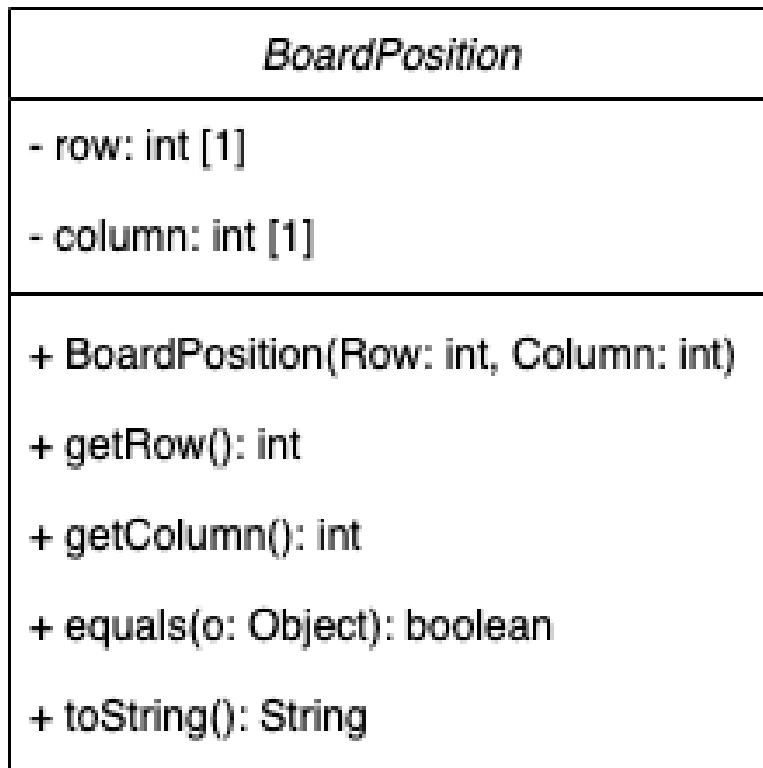


inputPlayAgain:



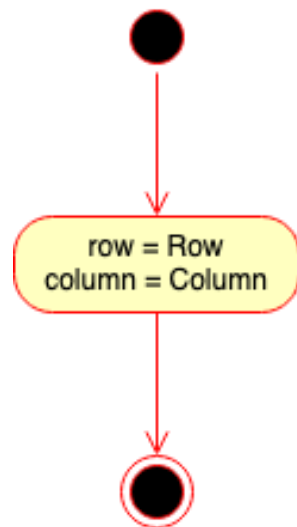
Class 2: BoardPosition

Class diagram

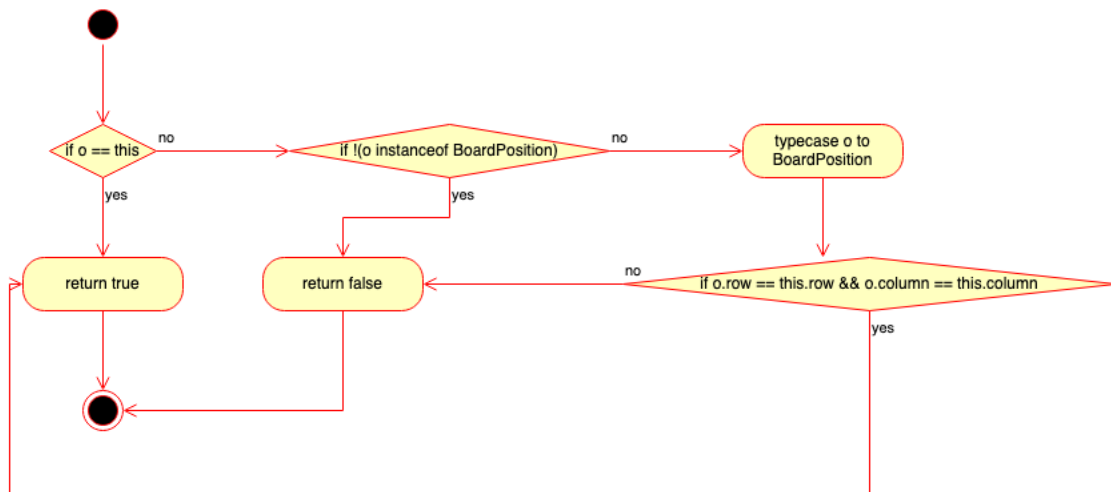


Activity diagrams

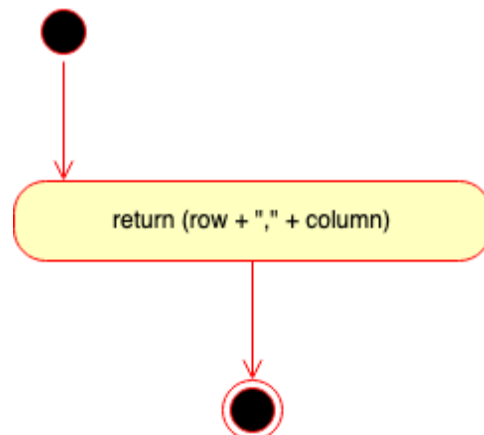
Constructor (BoardPosition):



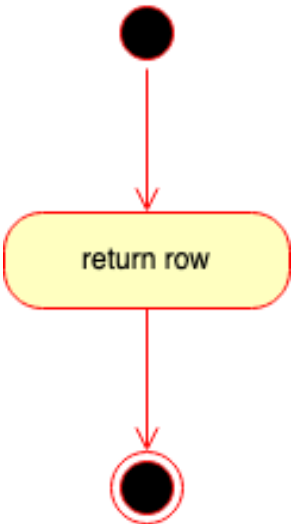
equals:



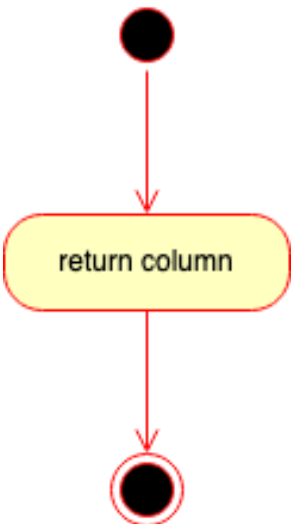
toString:



getRow:



getColumn:



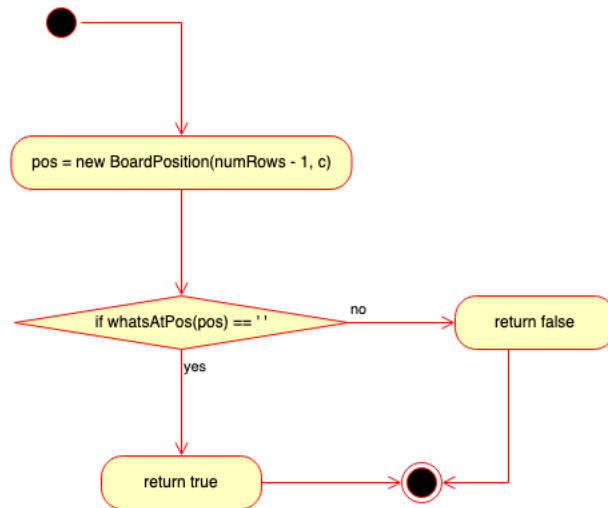
Interface 1: IGameBoard

Interface diagram

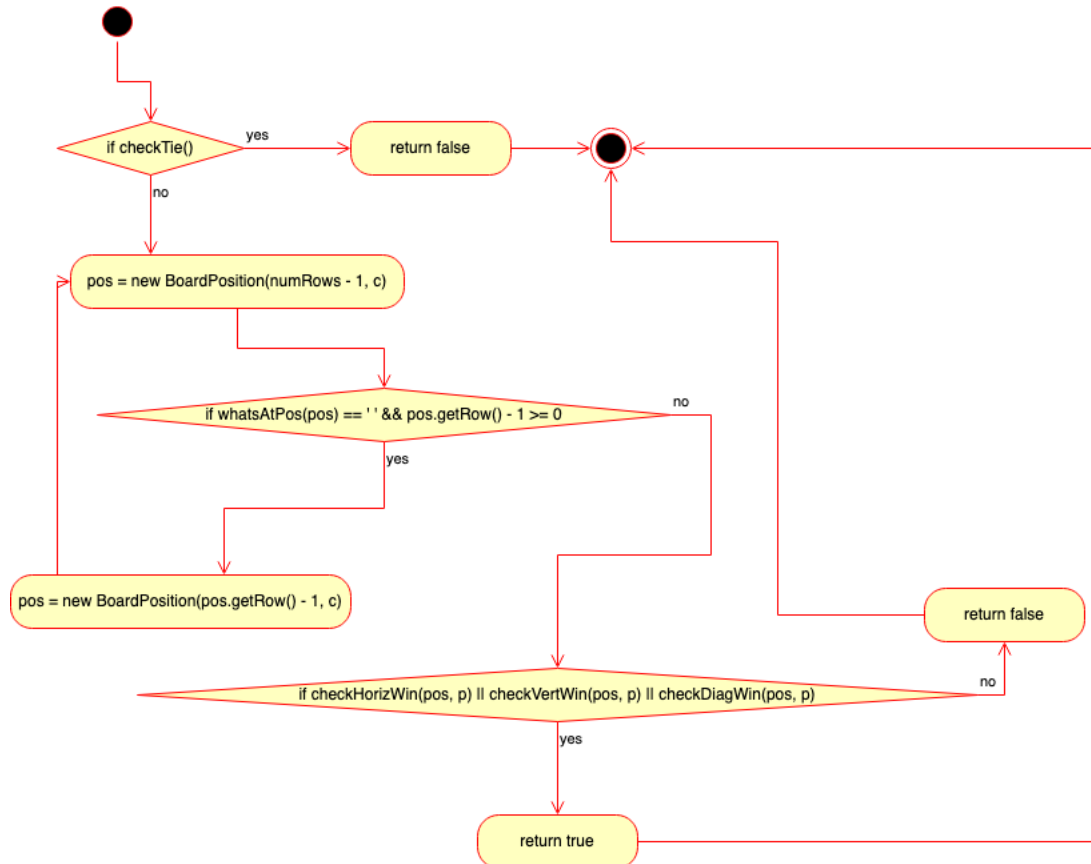
<i>IGameBoard</i>
- MAX_ROWS: int [1] - MAX_COLUMNS: int [1] - NUM_TO_WIN: int [1]
+ checkIfFree(c: int): boolean + placeToken(p: char, c: int): void + checkForWin(c: int): boolean + checkTie(): boolean + checkHorizWin(pos: BoardPosition, p: char): boolean + checkVertWin(pos: BoardPosition, p: char): boolean + checkDiagWin(pos: BoardPosition, p: char): boolean + whatsAtPos(pos: BoardPosition): char + isPlayerAtPos(pos: BoardPosition, player: char): boolean + getNumRows() : int + getNumColumns() : int + getNumToWin() : int

Activity diagrams

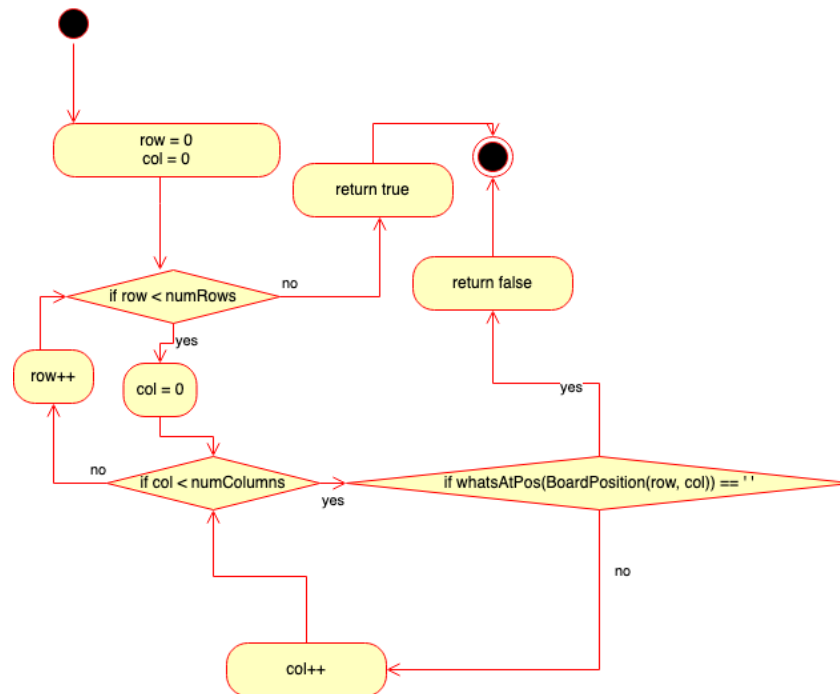
checkIfFree:



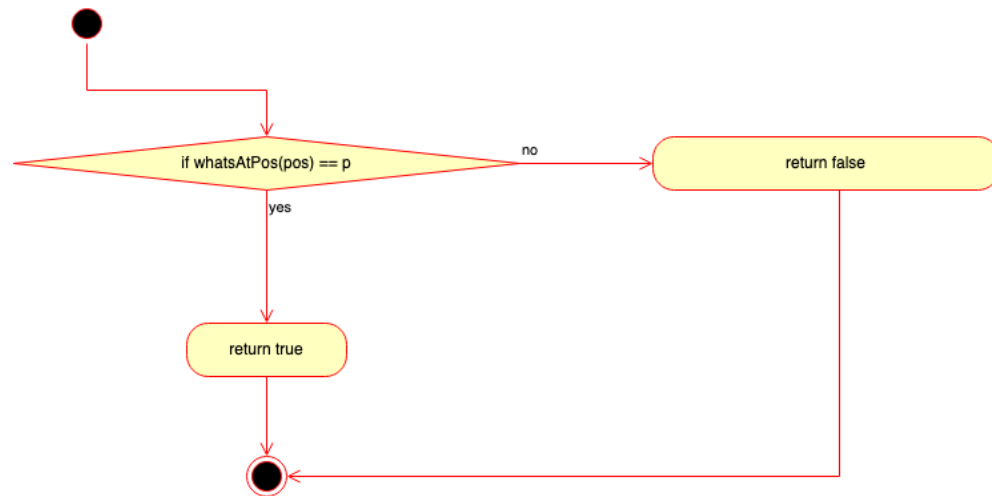
checkForWin:



checkTie:

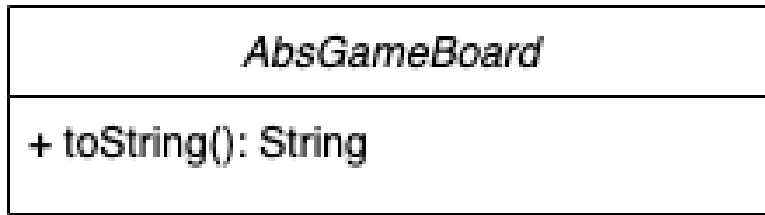


isPlayerAtPos:



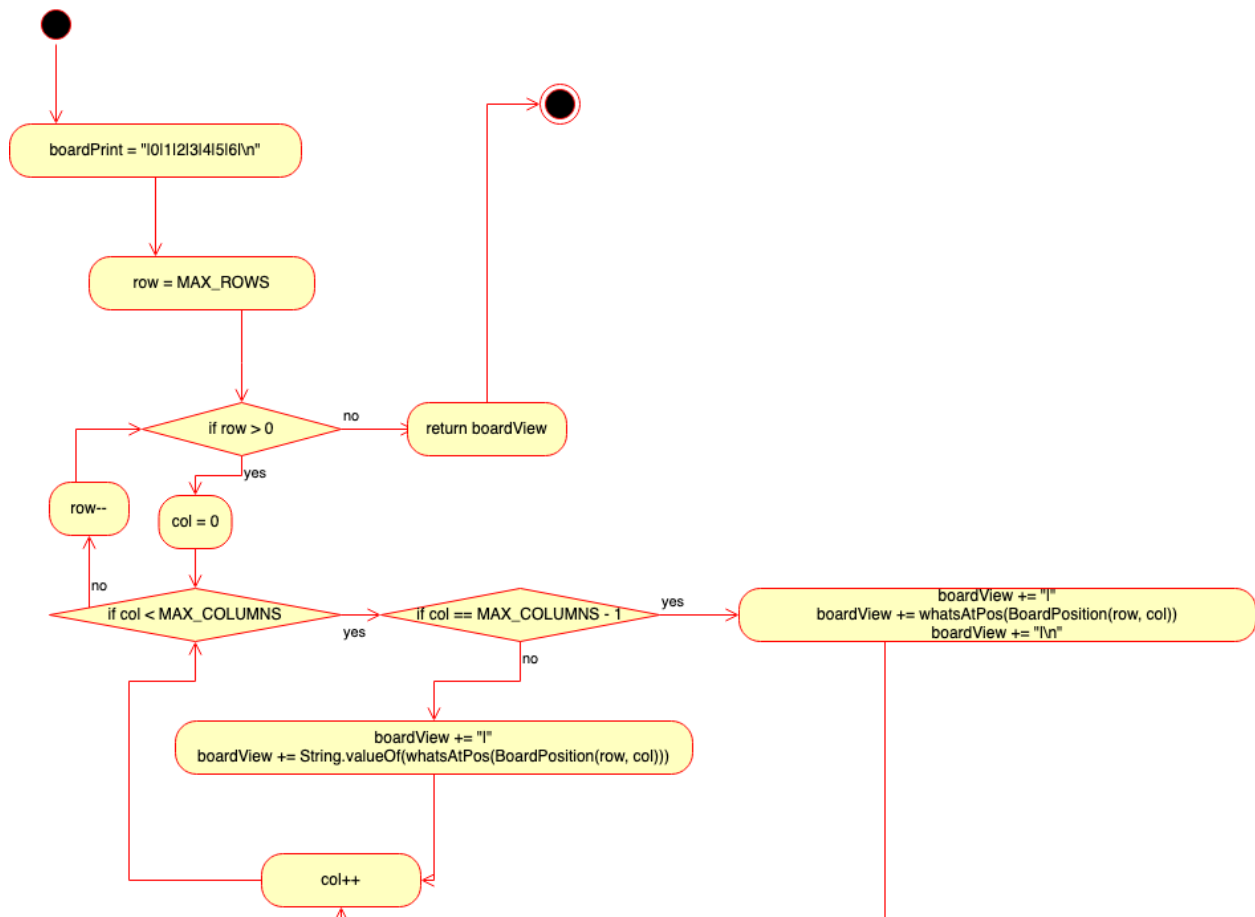
Class 3: AbsGameBoard

Class diagram



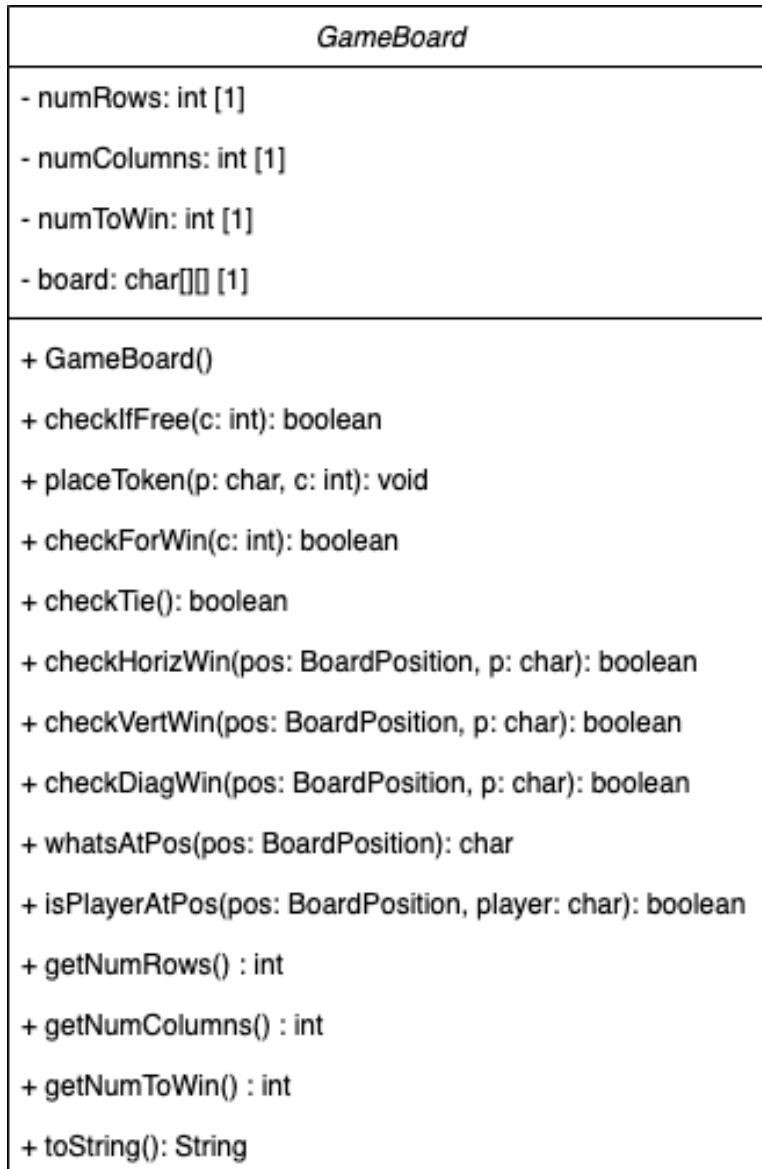
Activity diagrams

toString:



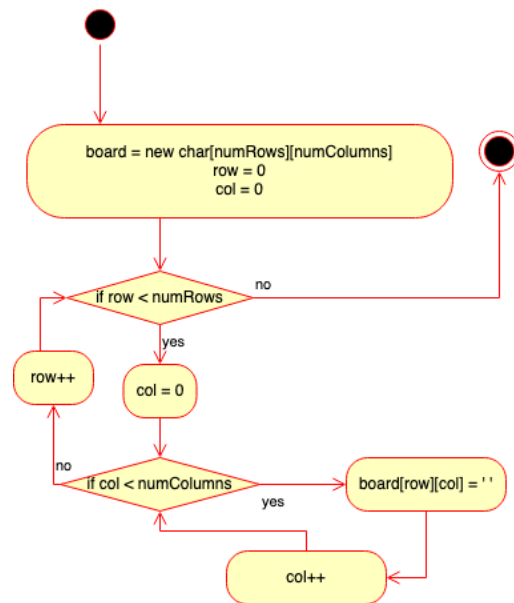
Class 4: GameBoard

Class diagram

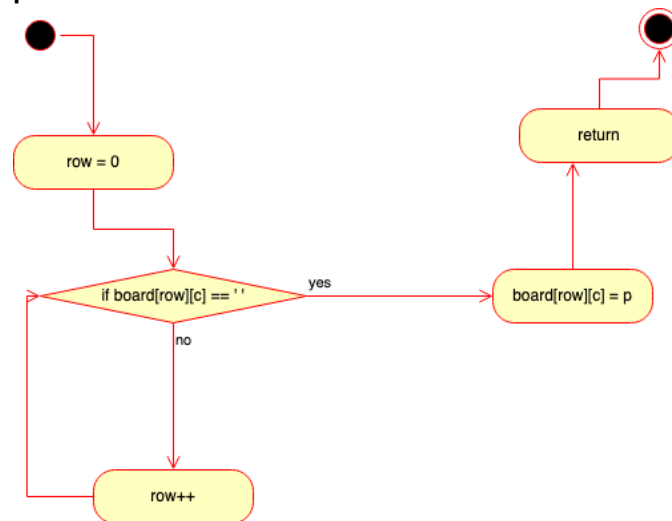


Activity diagrams

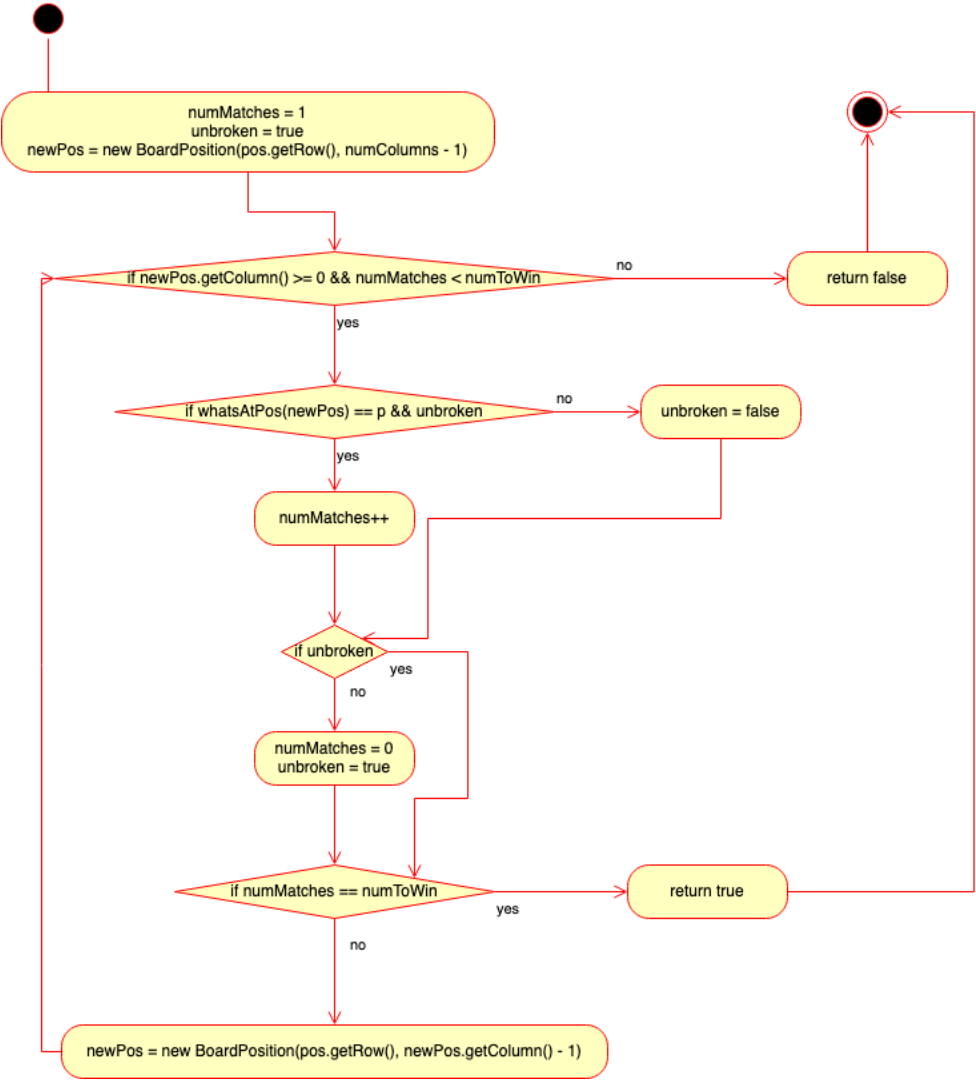
Constructor (GameBoard):



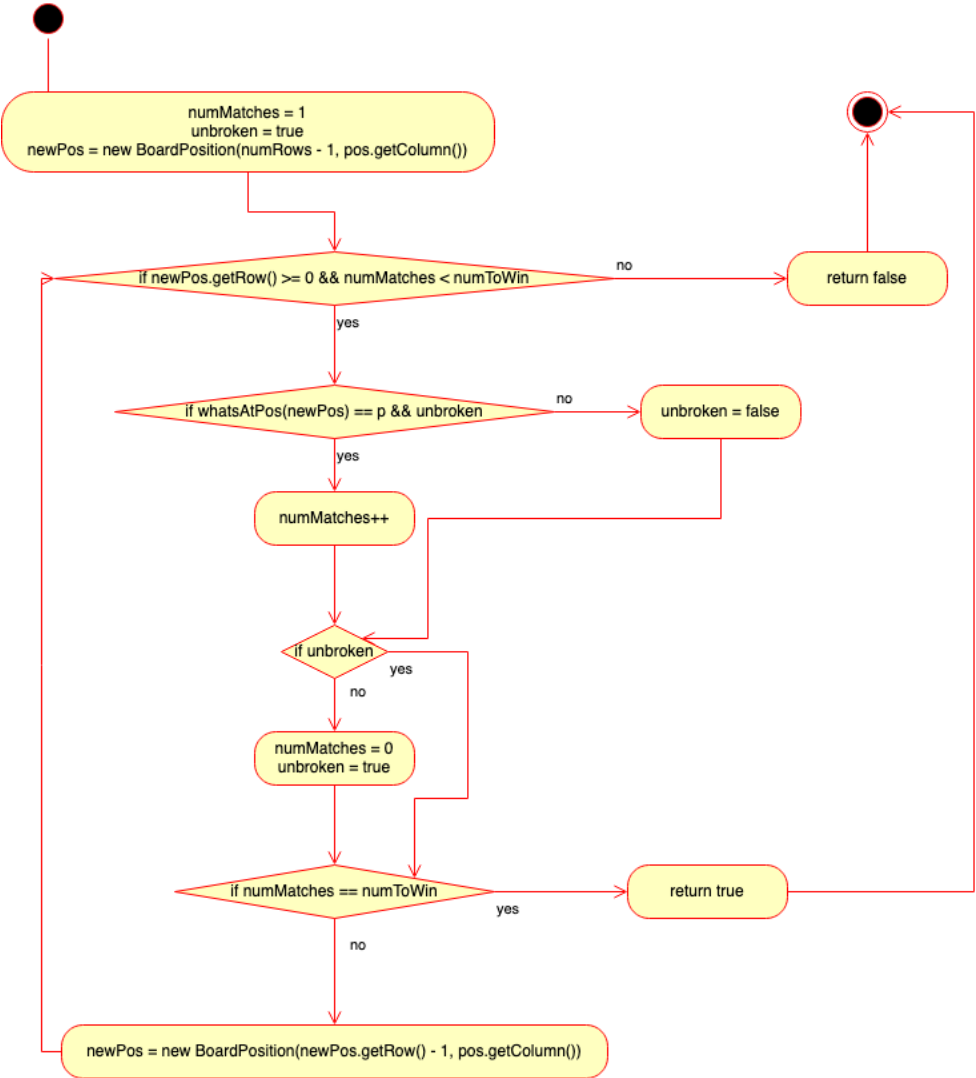
placeToken:



checkHorizWin:



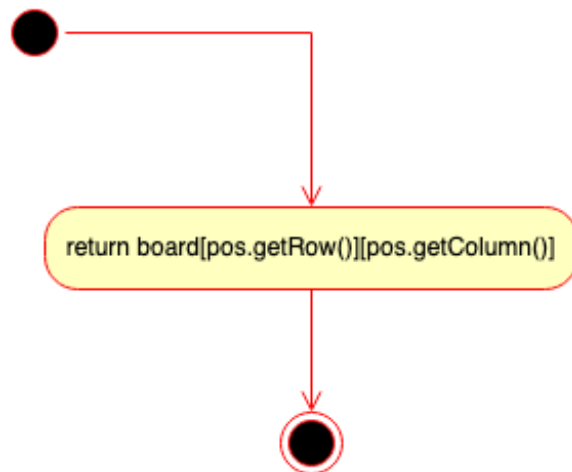
checkVertWin:



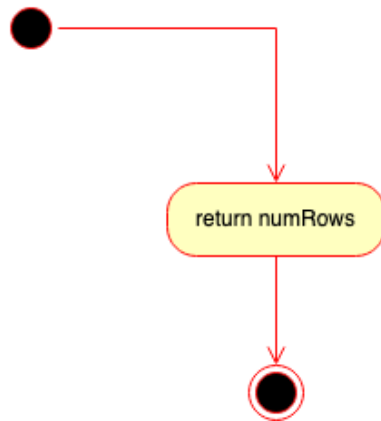
checkDiagWin:



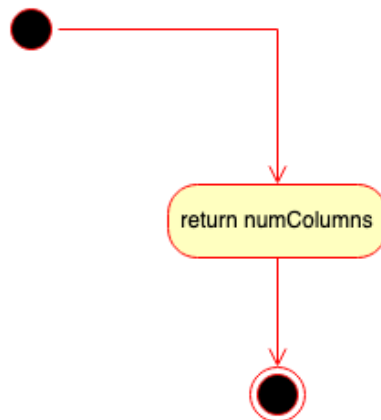
whatsAtPos:



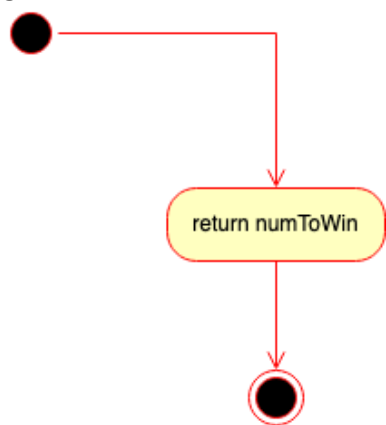
getNumRows:



getNumColumns:



getNumToWin:



Test Cases

Details in Project 4.