

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

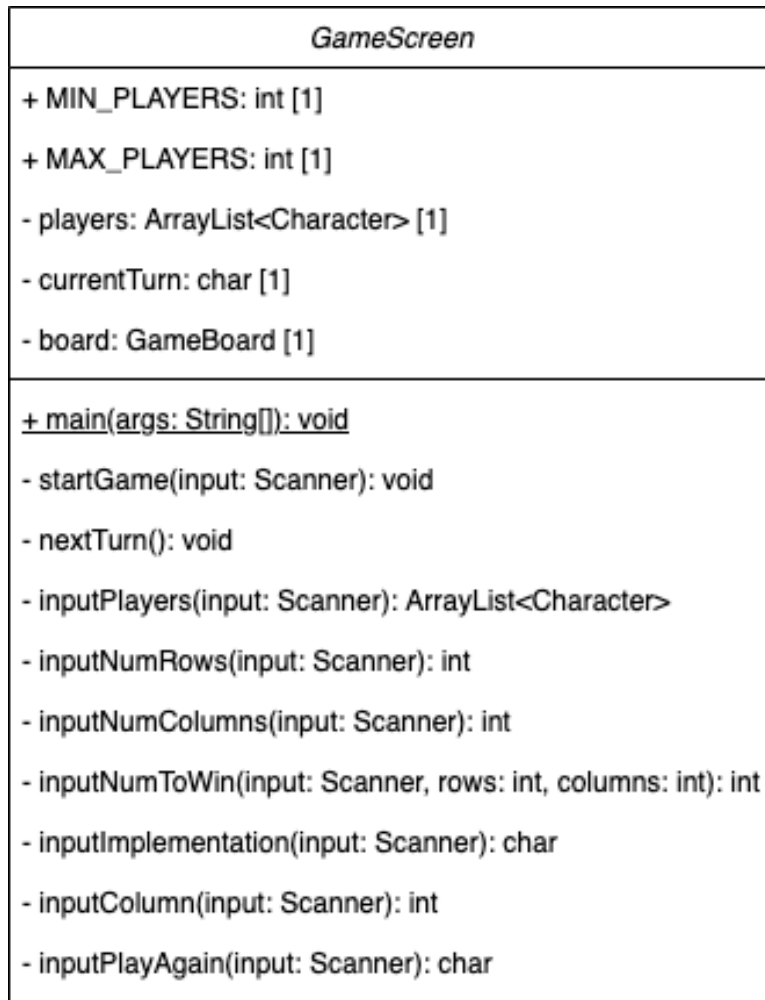
Details in Projects 2-5.

- 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.

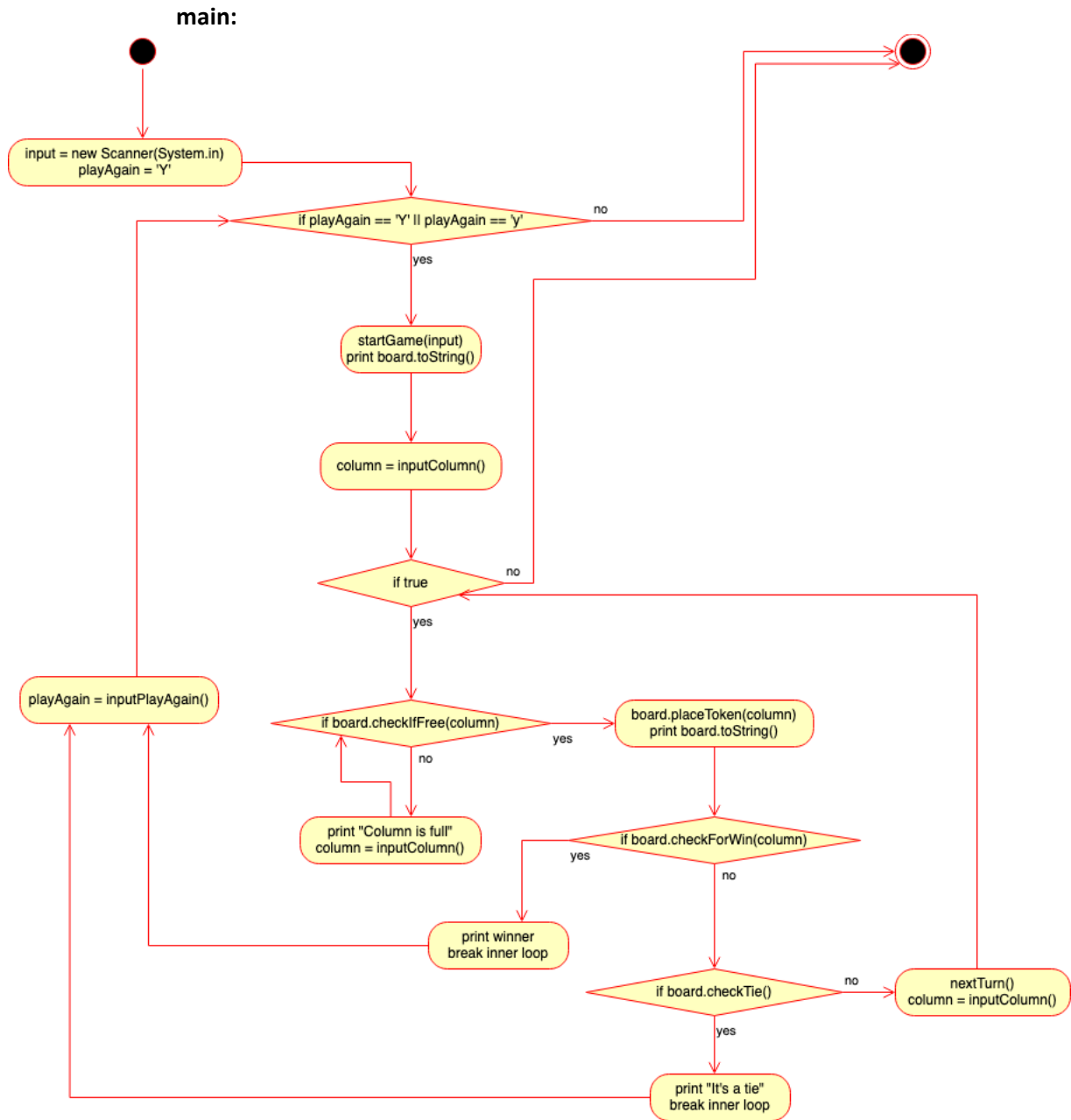
System Design

Class 1: GameScreen

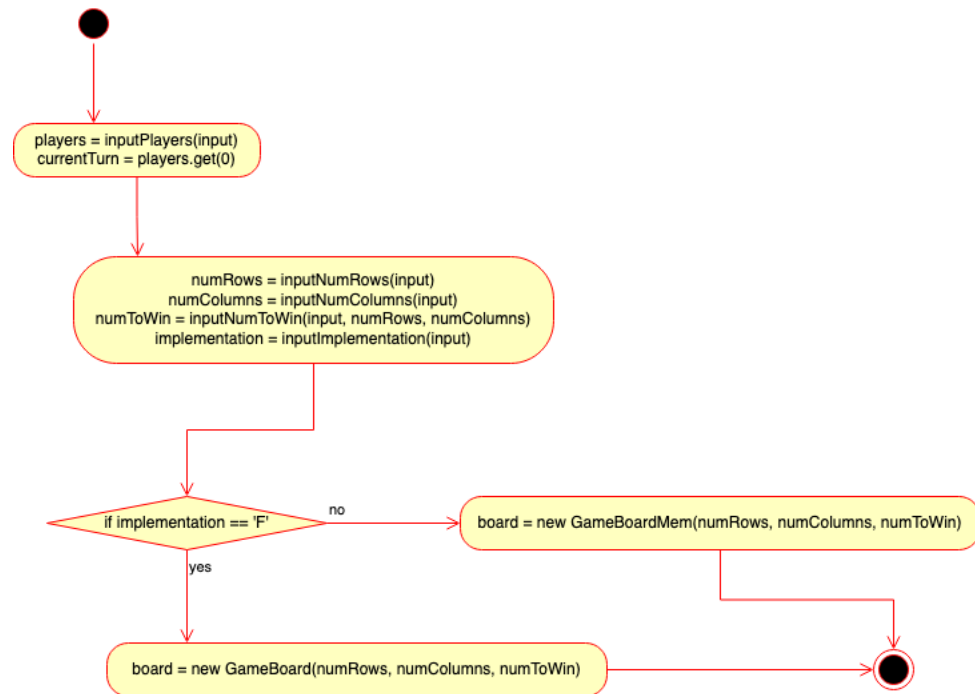
Class diagram



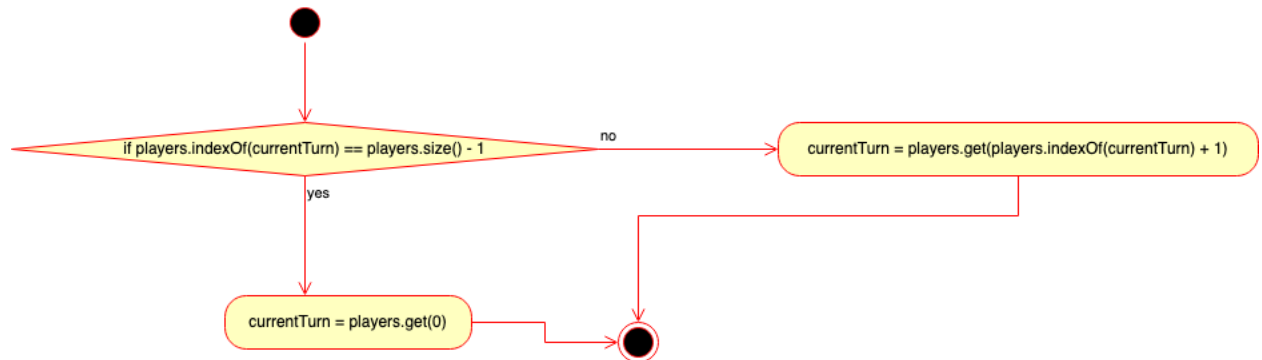
Activity diagrams



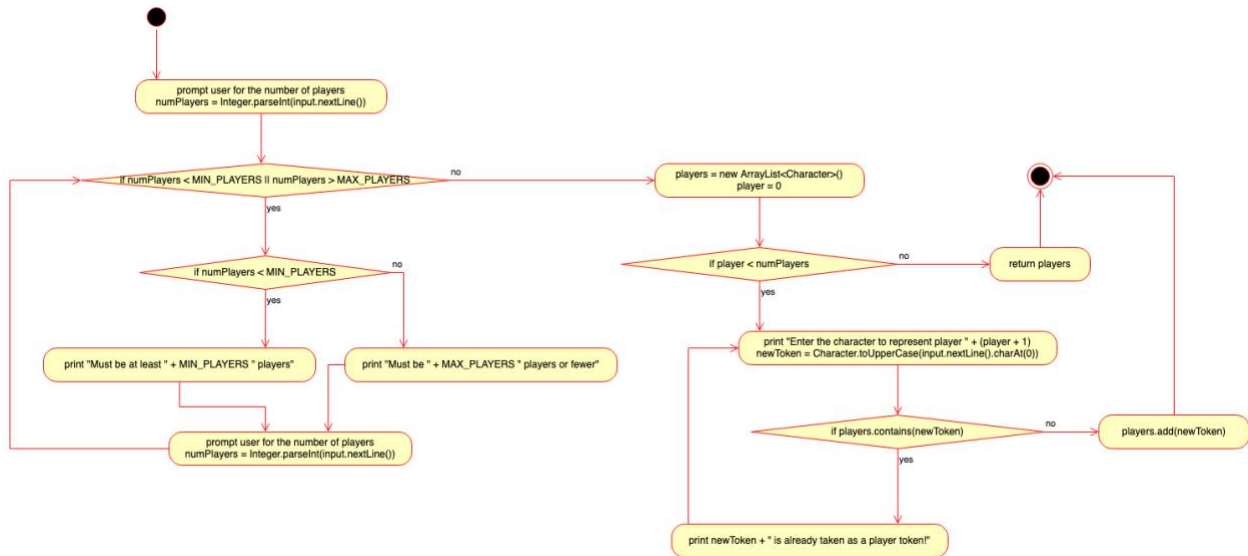
startGame:



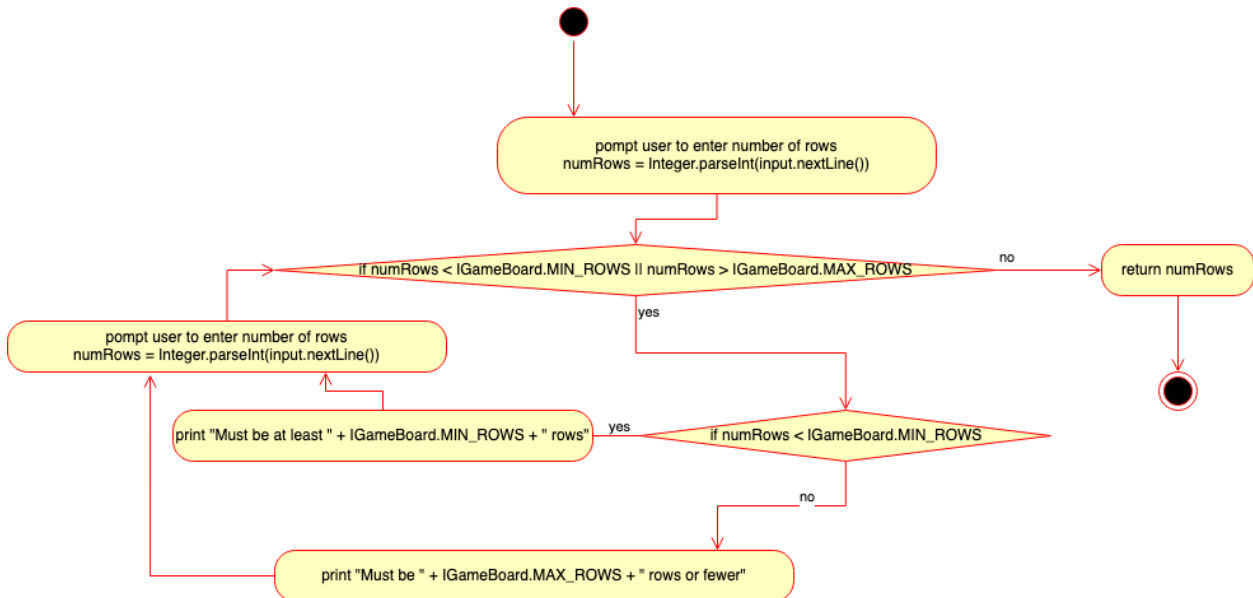
nextTurn:



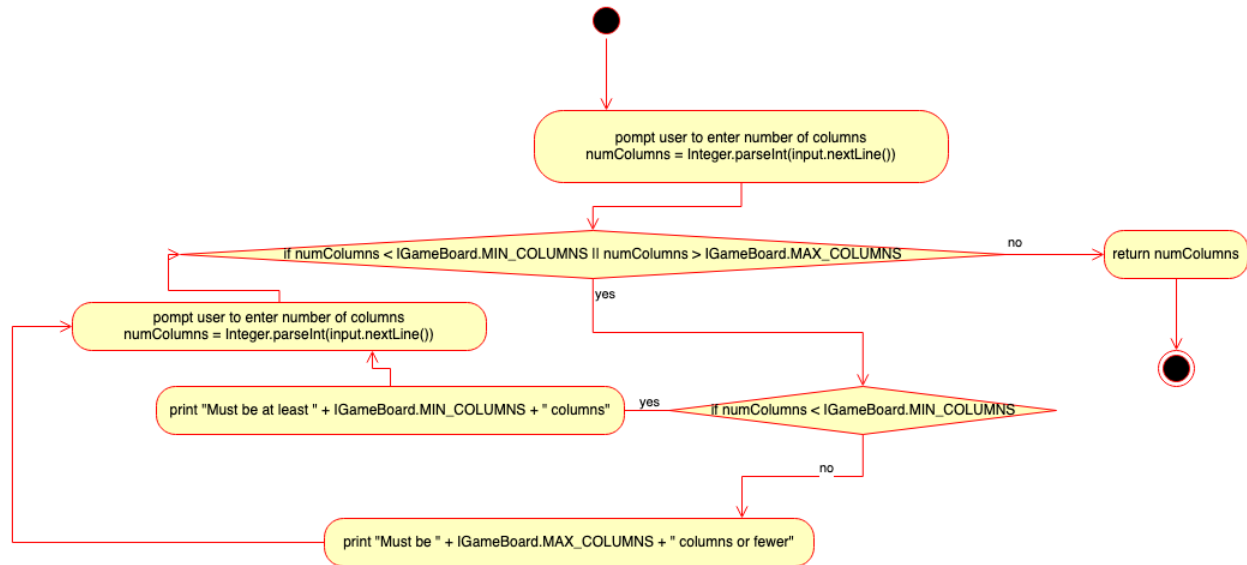
inputPlayers:



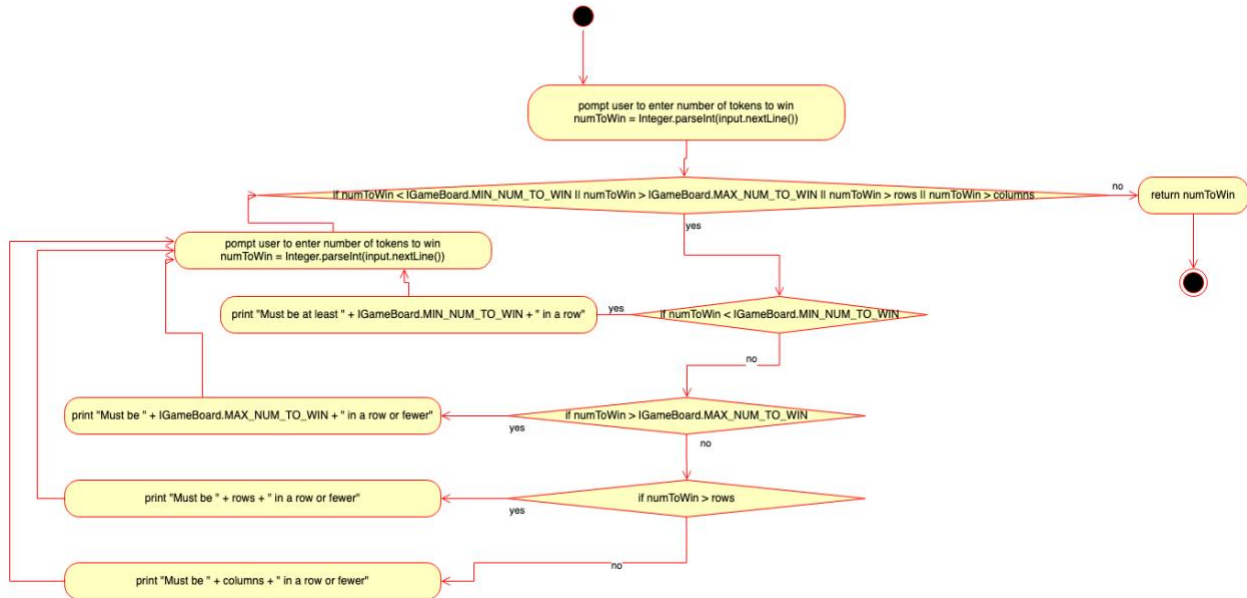
inputNumRows:



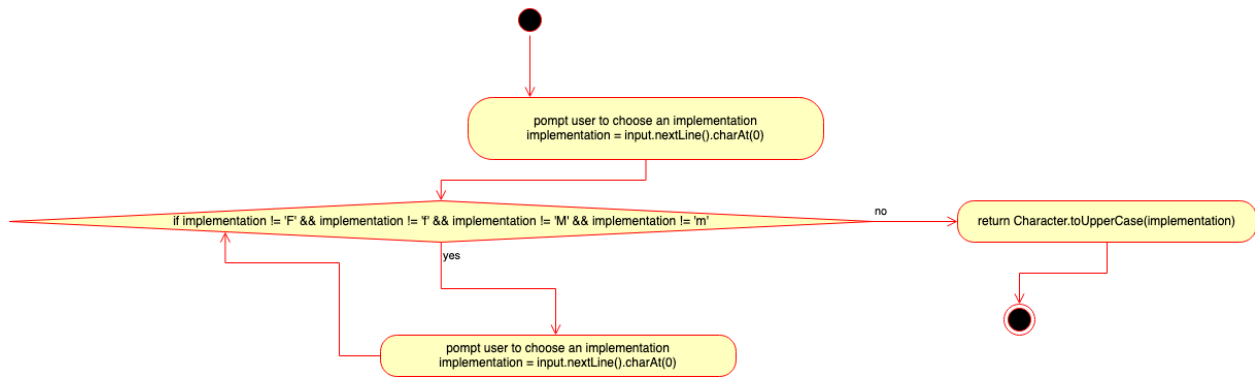
inputNumColumns:



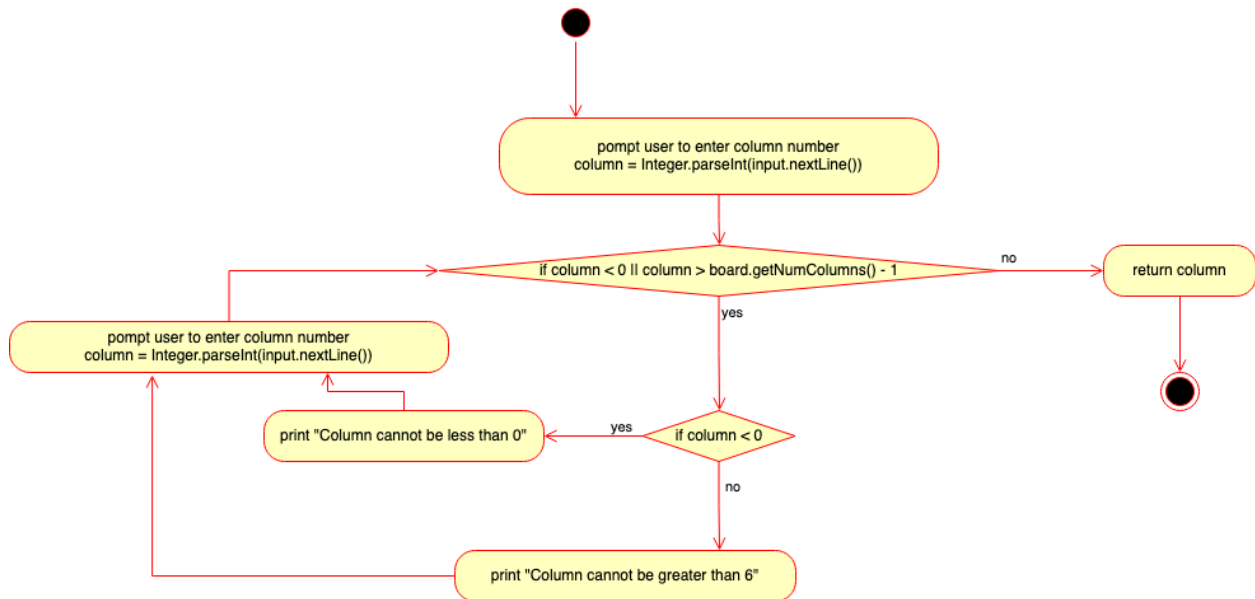
inputNumToWin:



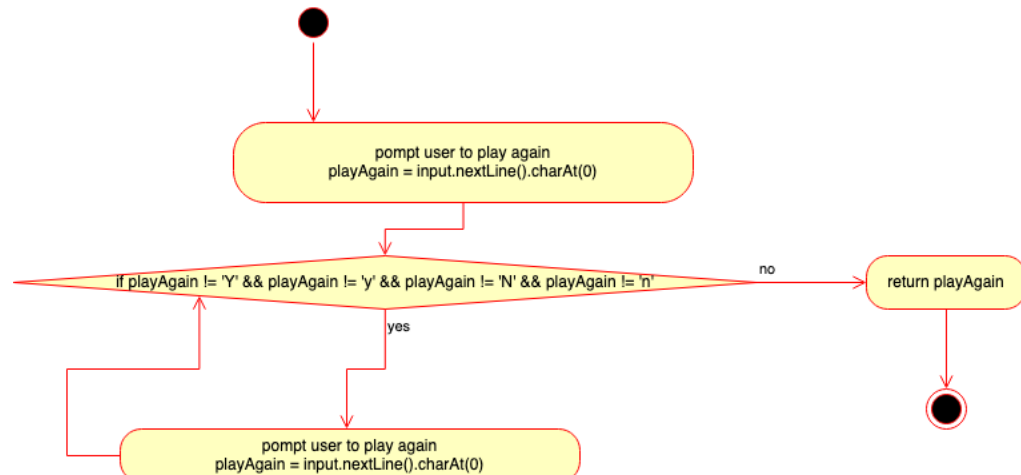
inputImplementation:



inputColumn:

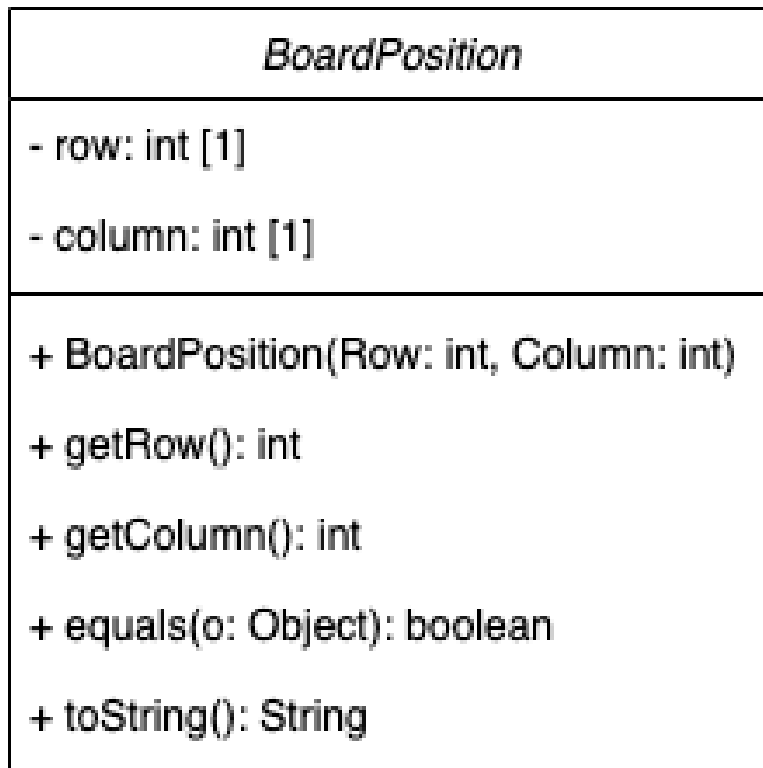


inputPlayAgain:



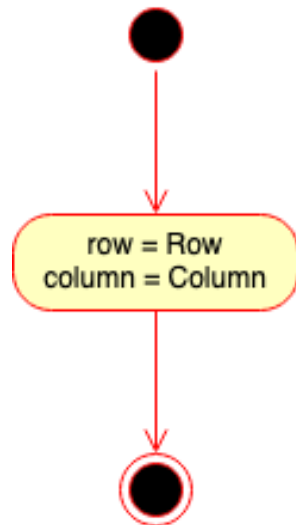
Class 2: BoardPosition

Class diagram

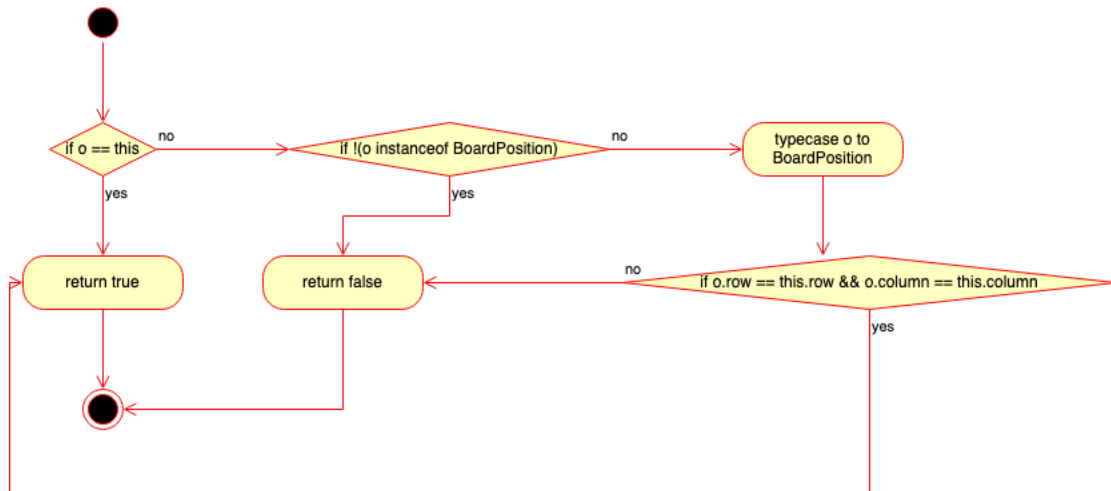


Activity diagrams

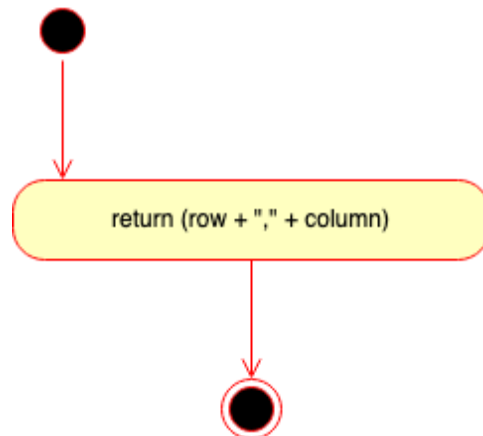
Constructor (BoardPosition):



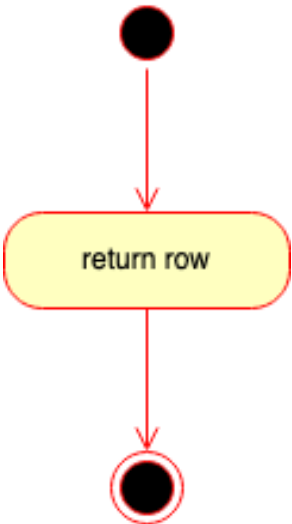
equals:



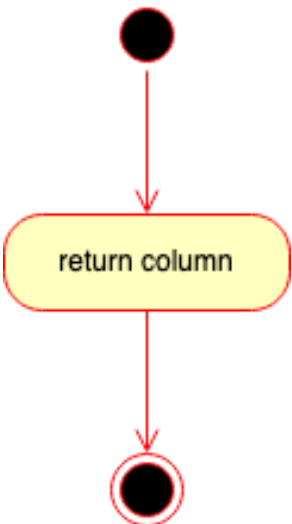
toString:



getRow:

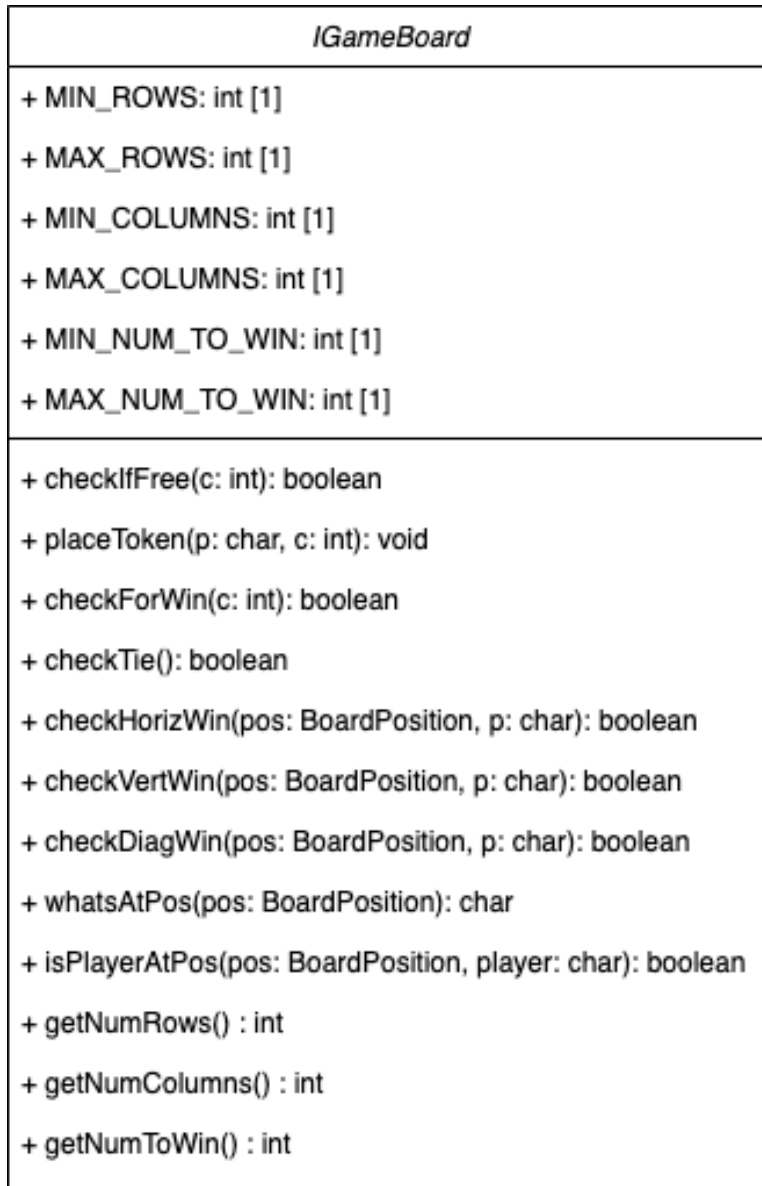


getColumn:



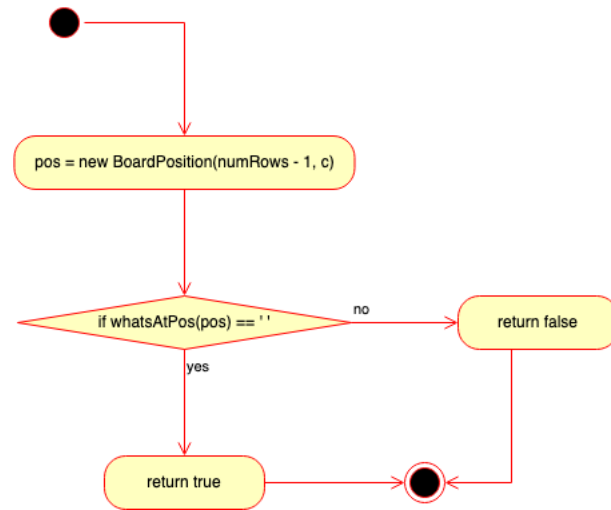
Interface 1: IGameBoard

Interface diagram

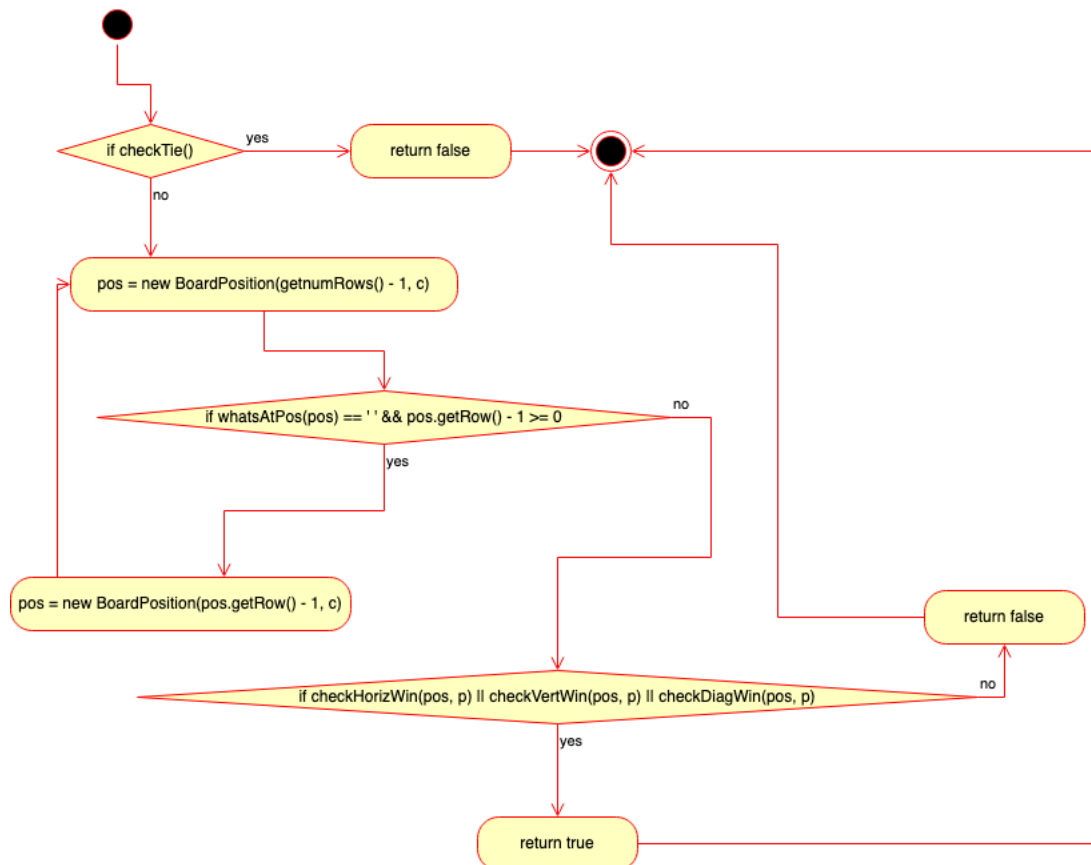


Activity diagrams

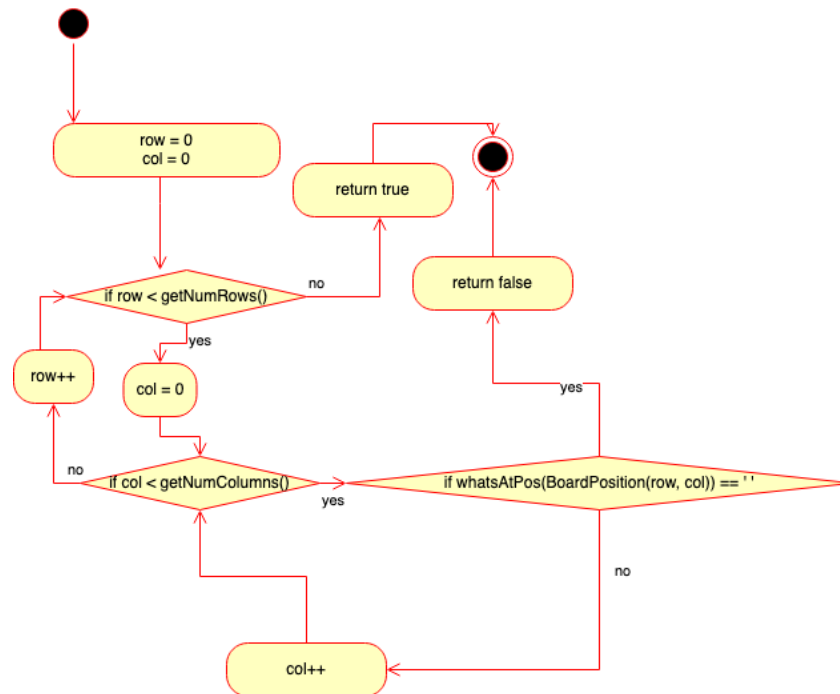
checkIfFree:



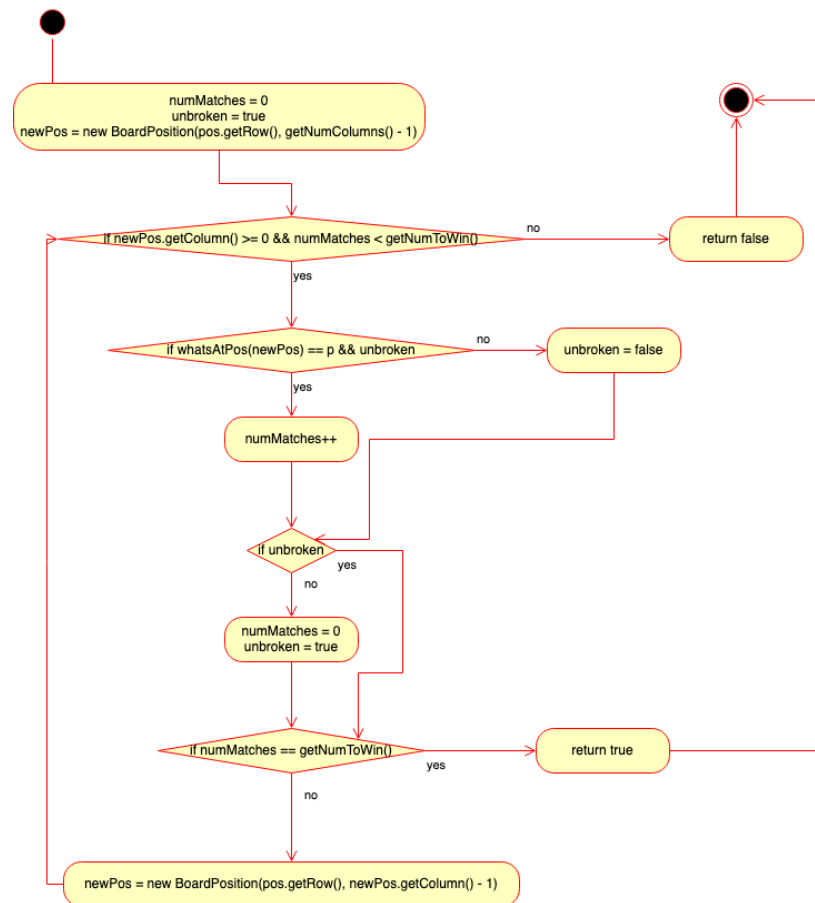
checkForWin:



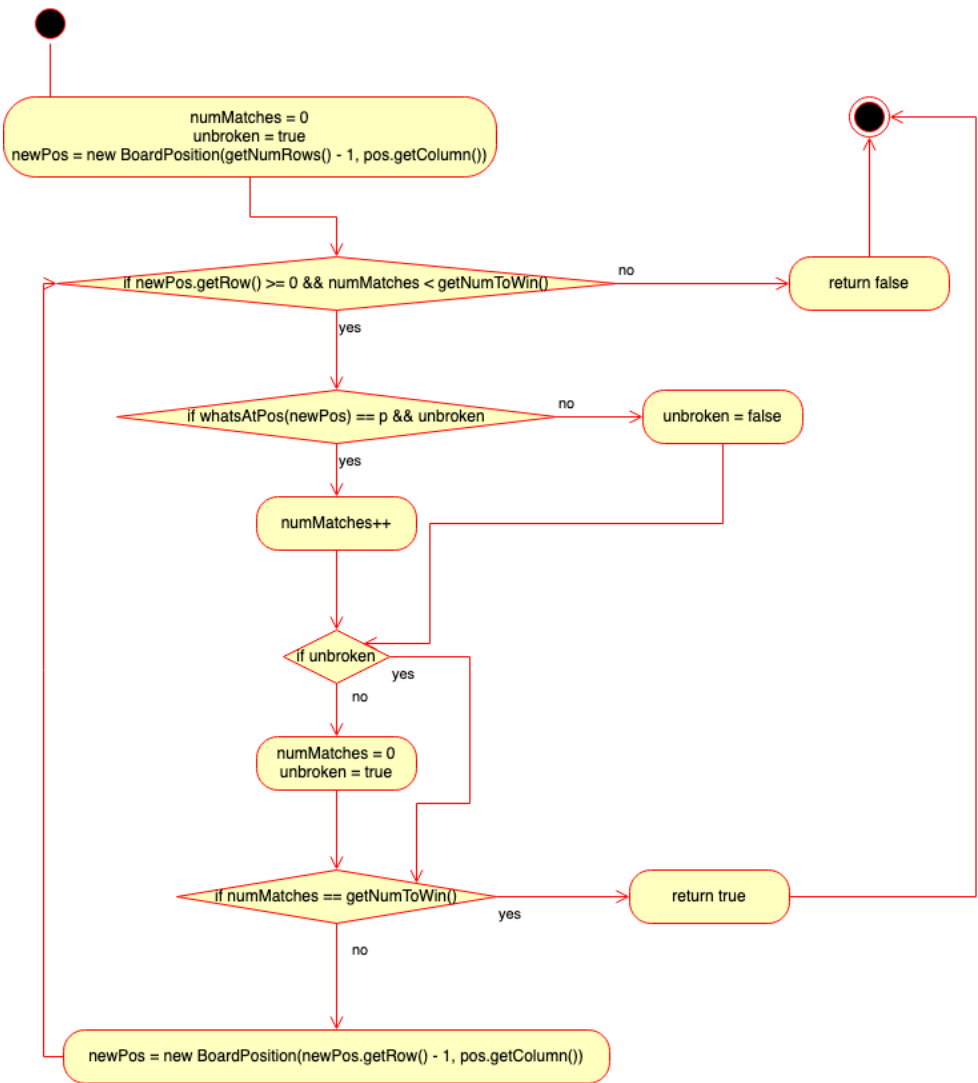
checkTie:



checkHorizWin:



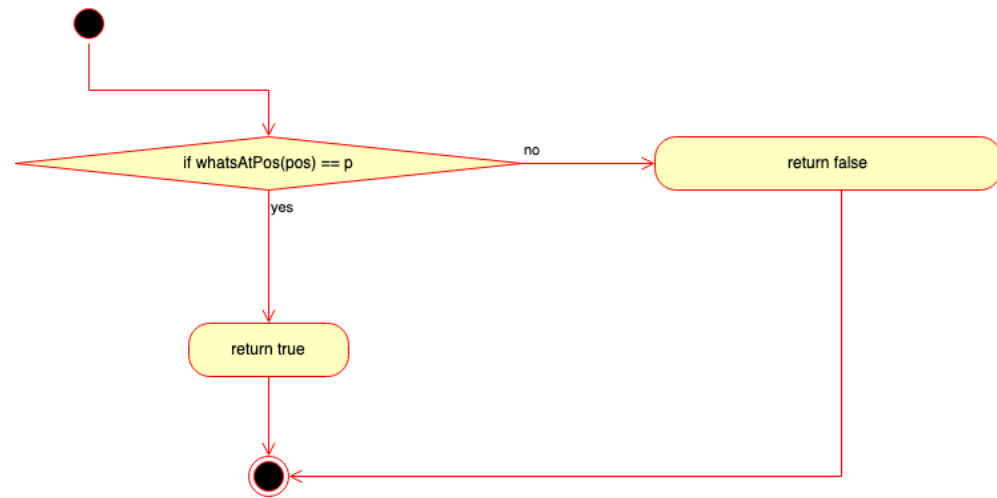
checkVertWin:



checkDiagWin:

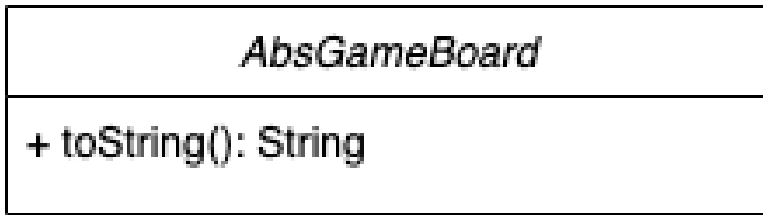


isPlayerAtPos:



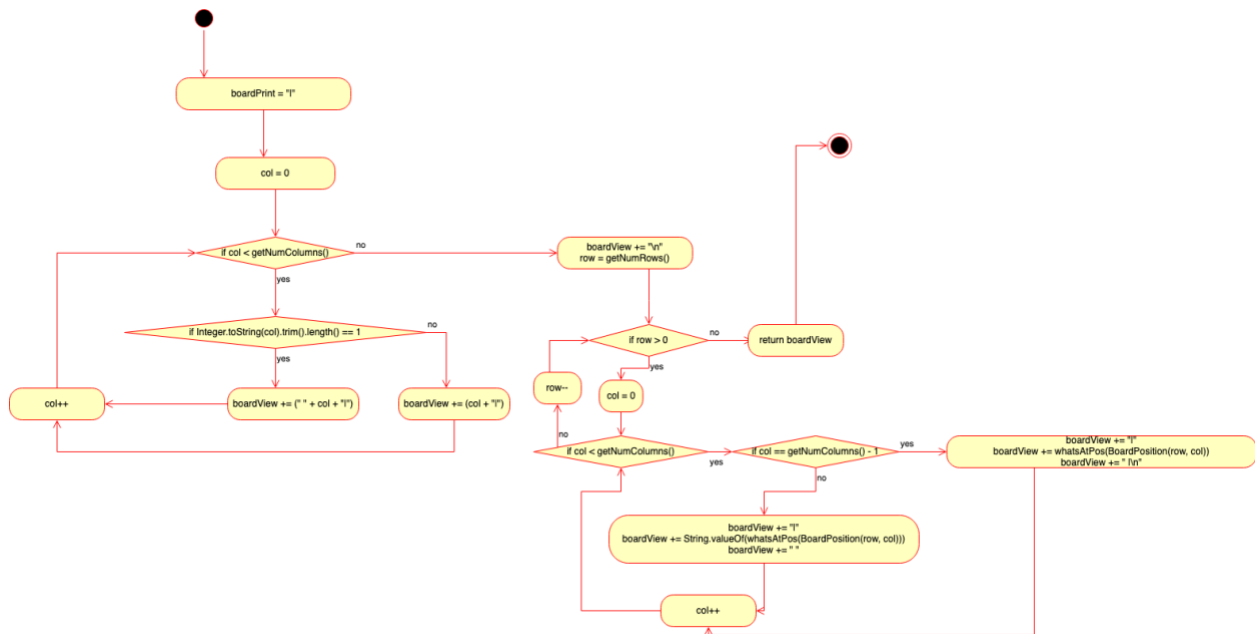
Class 3: AbsGameBoard

Class diagram



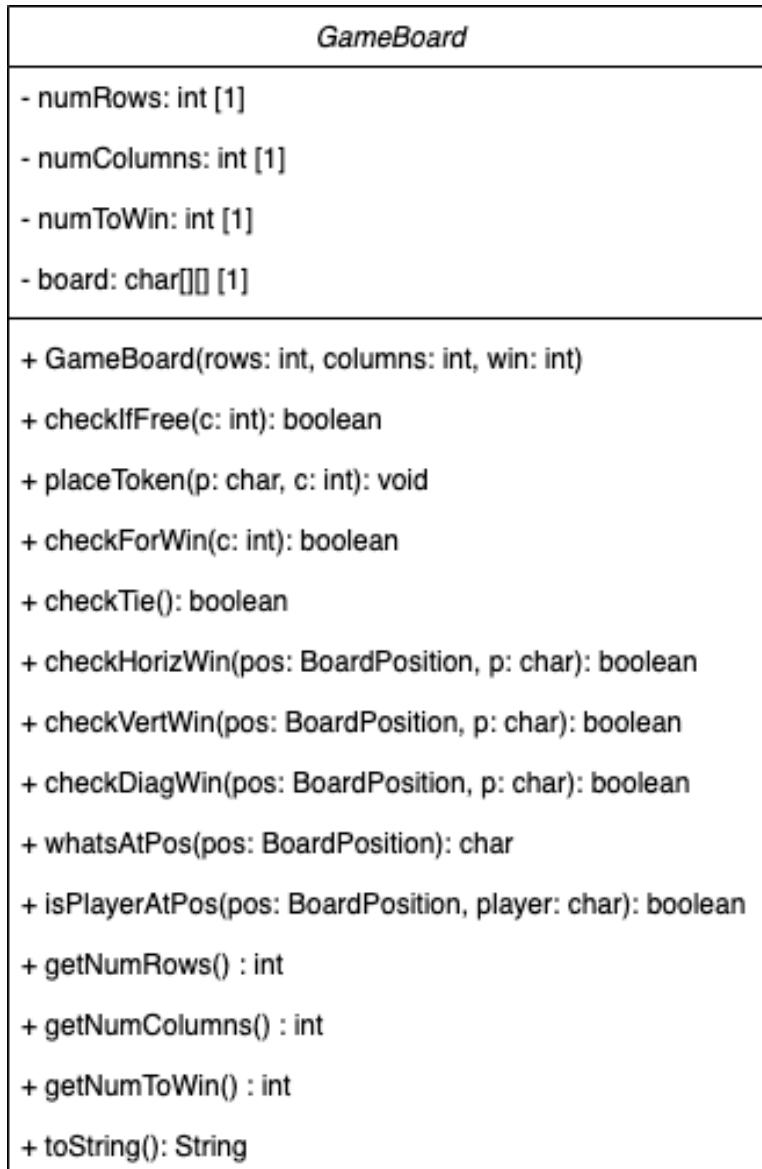
Activity diagrams

toString:



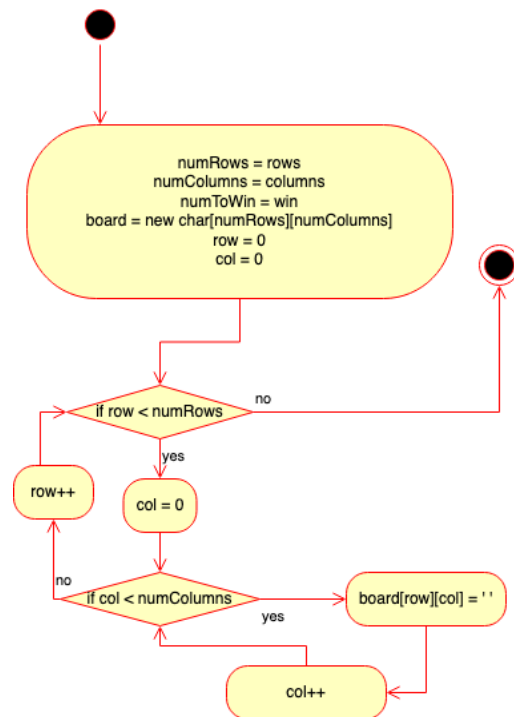
Class 4: GameBoard

Class diagram

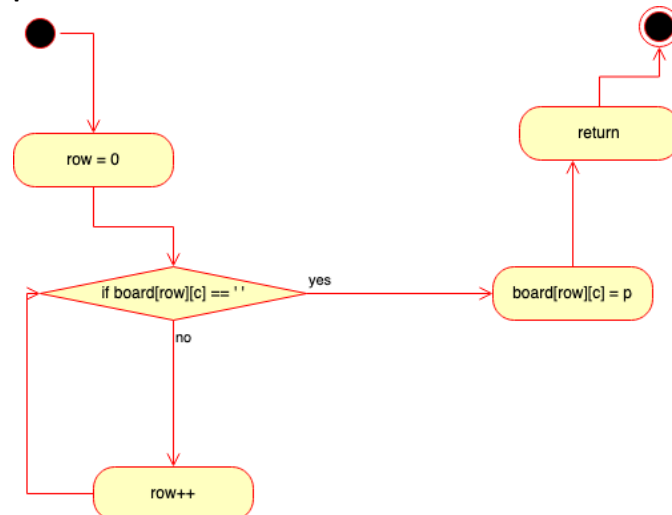


Activity diagrams

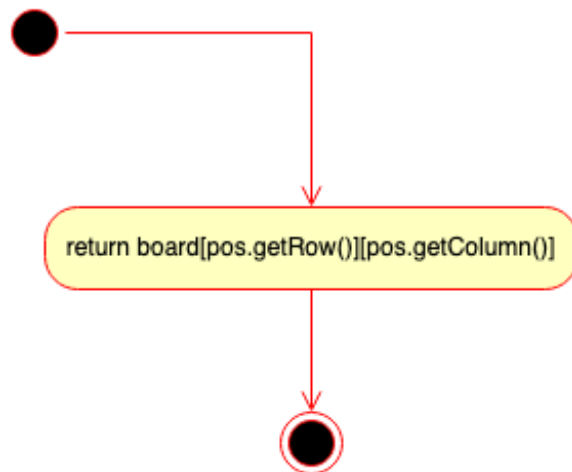
Constructor (GameBoard):



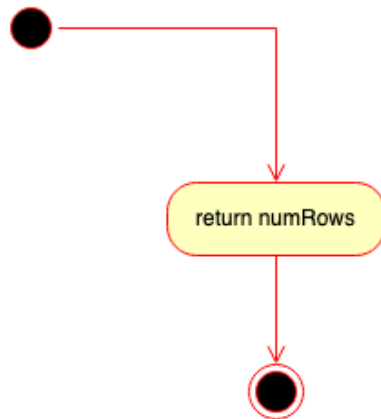
placeToken:



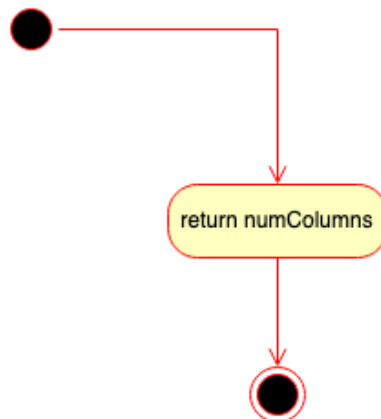
whatsAtPos:



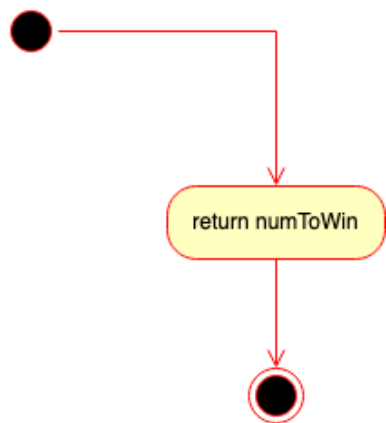
getNumRows:



getNumColumns:

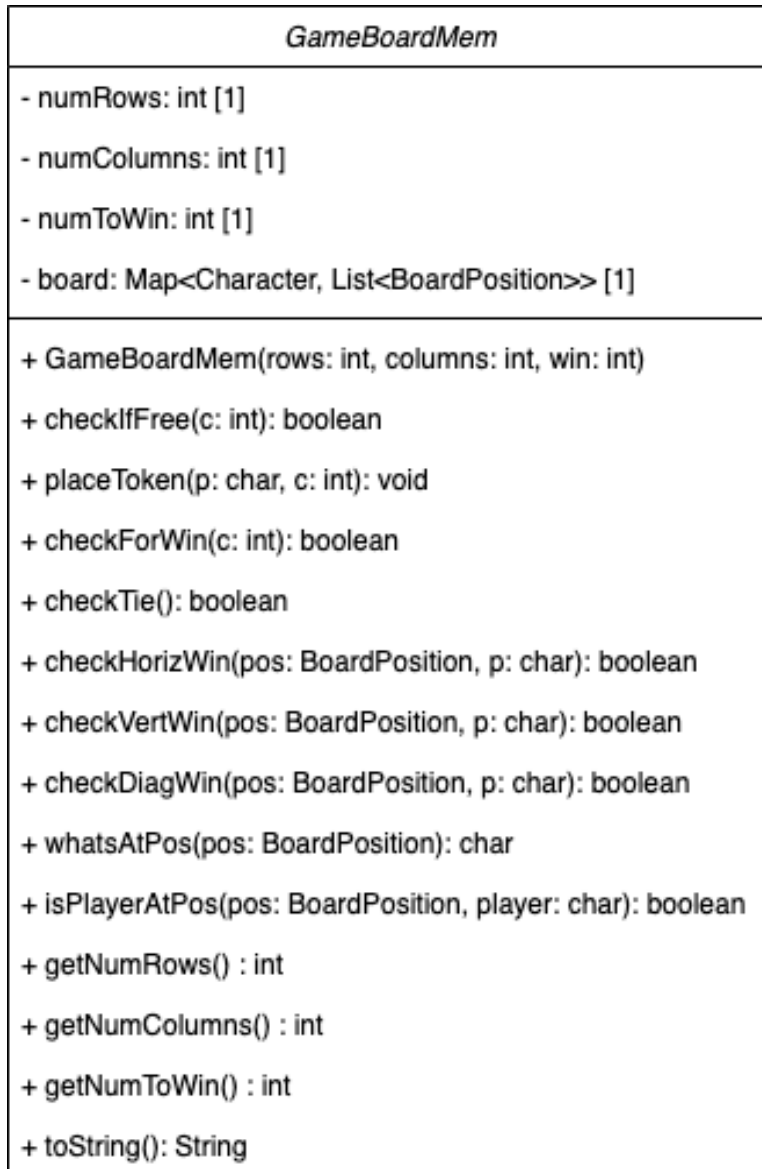


getNumToWin:



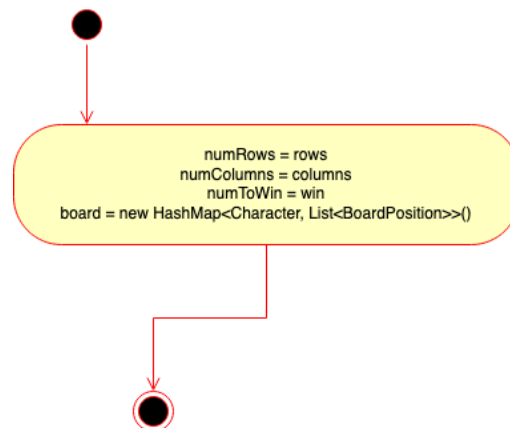
Class 4: GameBoardMem

Class diagram

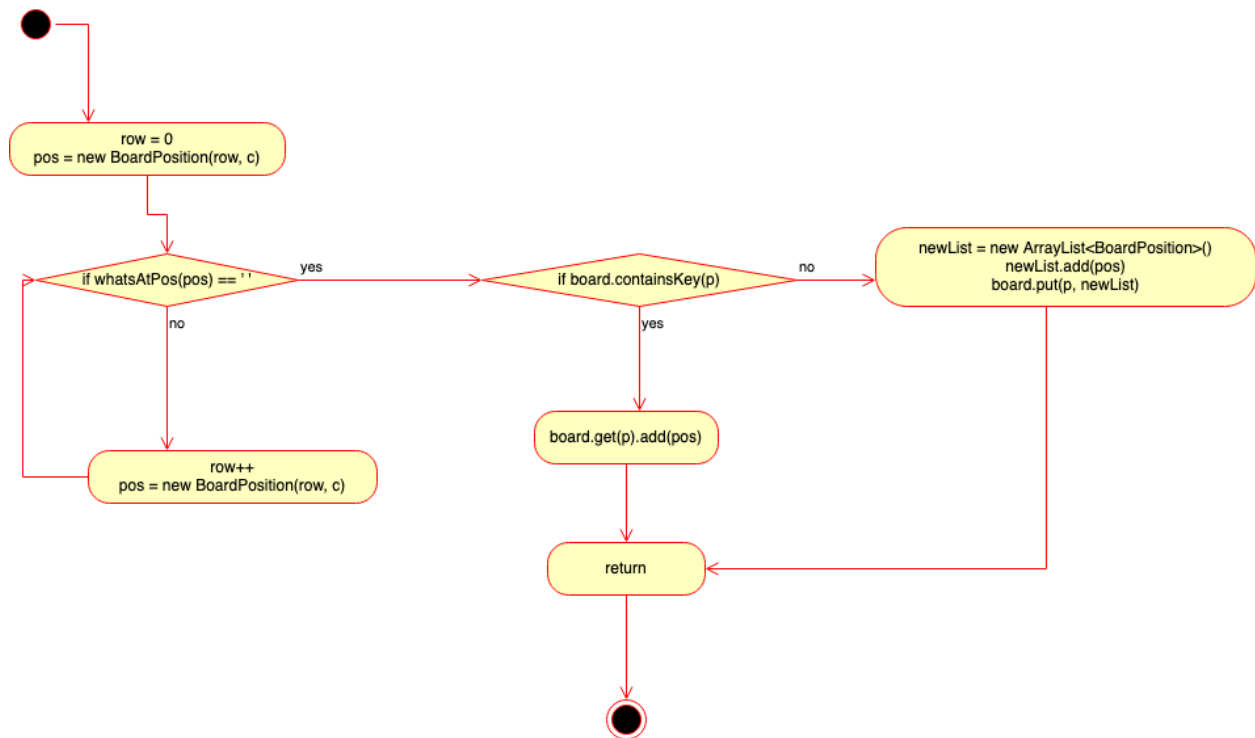


Activity diagrams

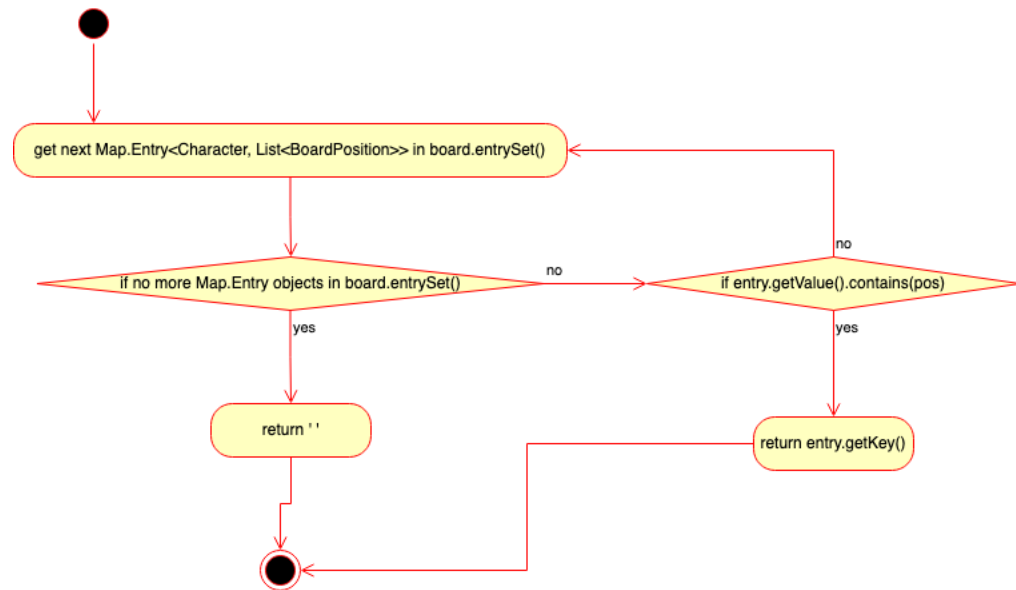
Constructor (GameBoardMem):



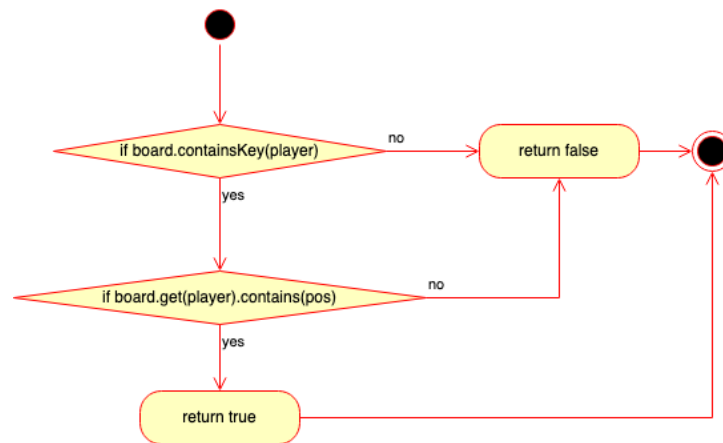
placeToken:



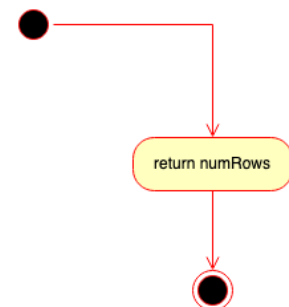
whatsAtPos:



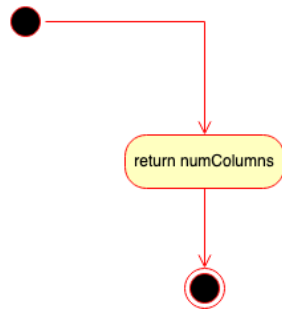
isPlayerAtPos:



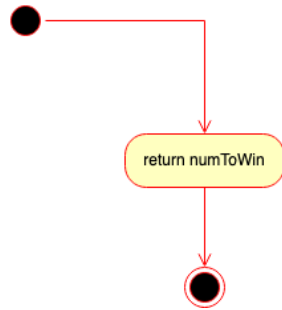
getNumRows:



getNumColumns:



getNumToWin:



Test Cases

Details in Project 4.