Ashlyn Cooper
CPSC 2150
Project 3
Project Report

Requirements Analysis

Functional Requirements

1. As a player, I can input a position so that I can play my token.
2. As a player, I can choose to play again so that I can replay tic tac toe.
3. As a player, I can have my input validated by the program so that I can place my token in a valid position.
4. As a player, I can view output prompting for an input so that I can see when I should make an input.
5. As a player, I can view an outputted message at the end of the game, so that I know how the game ended.
6. As a player, I can win horizontally, as that I can win the game.
7. As a player, I can win vertically, so that I can win the game.
8. As a player, I can win diagonally so that I can win the game.
9. As a player, I can make a move after my opponent (if they have not one), so that I can take my turn.
10. The program can end the game without either player winning, so that the game can end in a tie.
11. As a user, I can input the number of columns and number of rows, so that I can set the size of the gameboard.
12. As a user, I can input the number in a row needed to win, so that the game is played with that rule.
13. As a user, I can input the number of players, so that the game can be played with that number of players.
14. As a player, I can input a unique character, so that I can have a unique token as my marker.
15. As a user, I can choose between a memory efficient or fast implementation of the game, so that the program runs with the desired implementation.

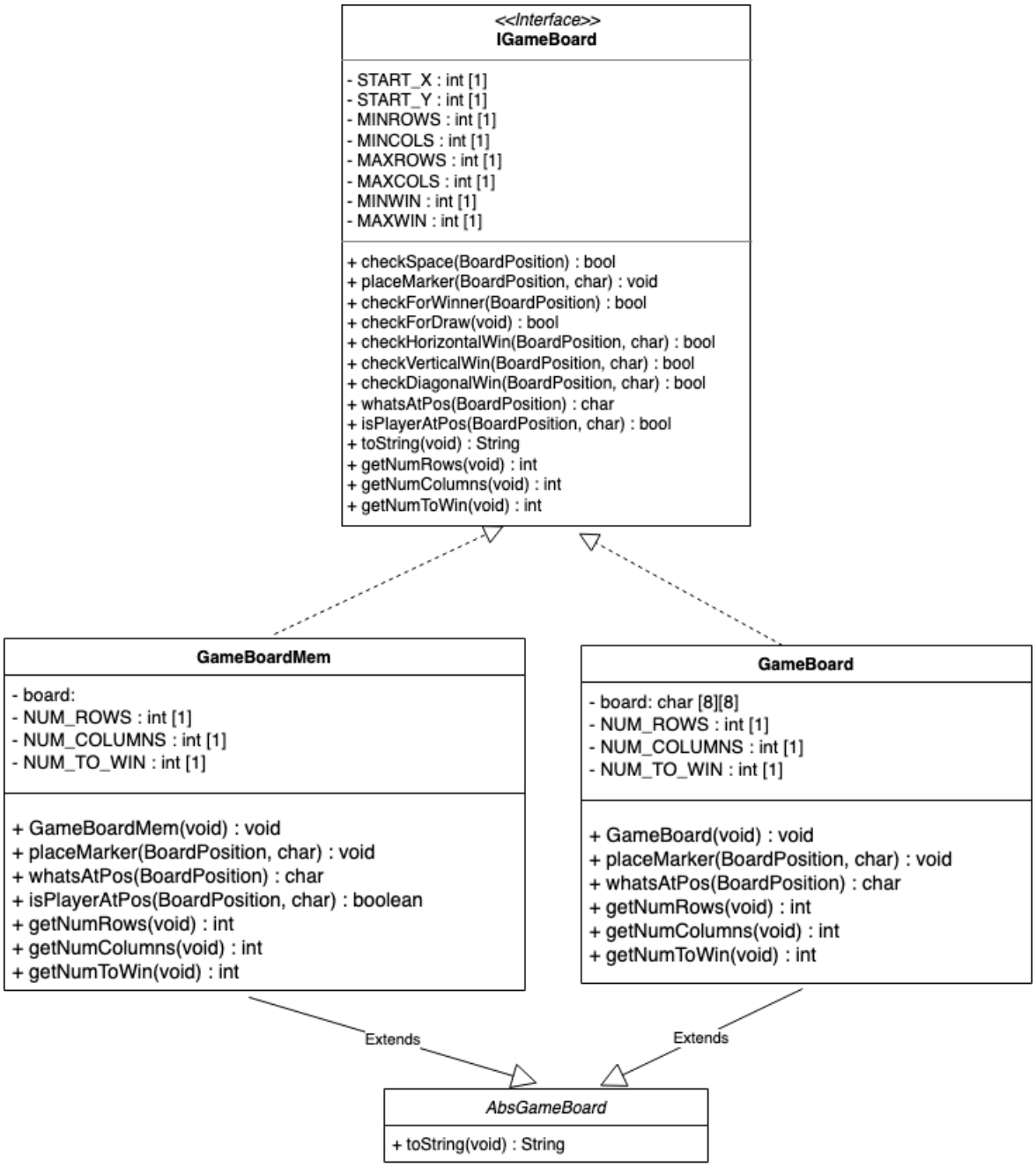
Non-functional Requirements

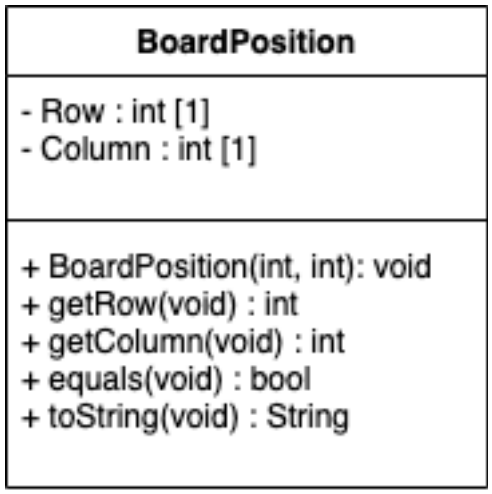
1. The system must be written in Java.
2. The system must run on a Unix machine.
3. The program can print the game board to the screen so that the player can see the board.
4. The program can alternate between players so that the program can be played by at least 2 players and up to 10.
5. Player 1 will always go first.
6. The program can be repeated so that the program has the ability to let the players play again.

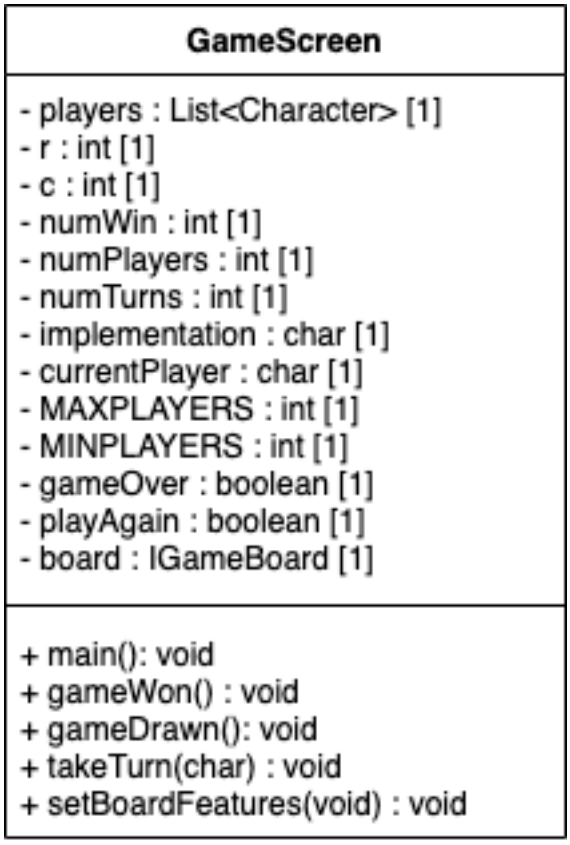
7. The game board is of size NUM_ROWS x NUM_COLUMNS as indicated by user input.
8. Coordinate (0,0) represents the top left corner of the game board.
9. The program has a memory efficient implementation and a fast implementation.

Design

Class Diagrams



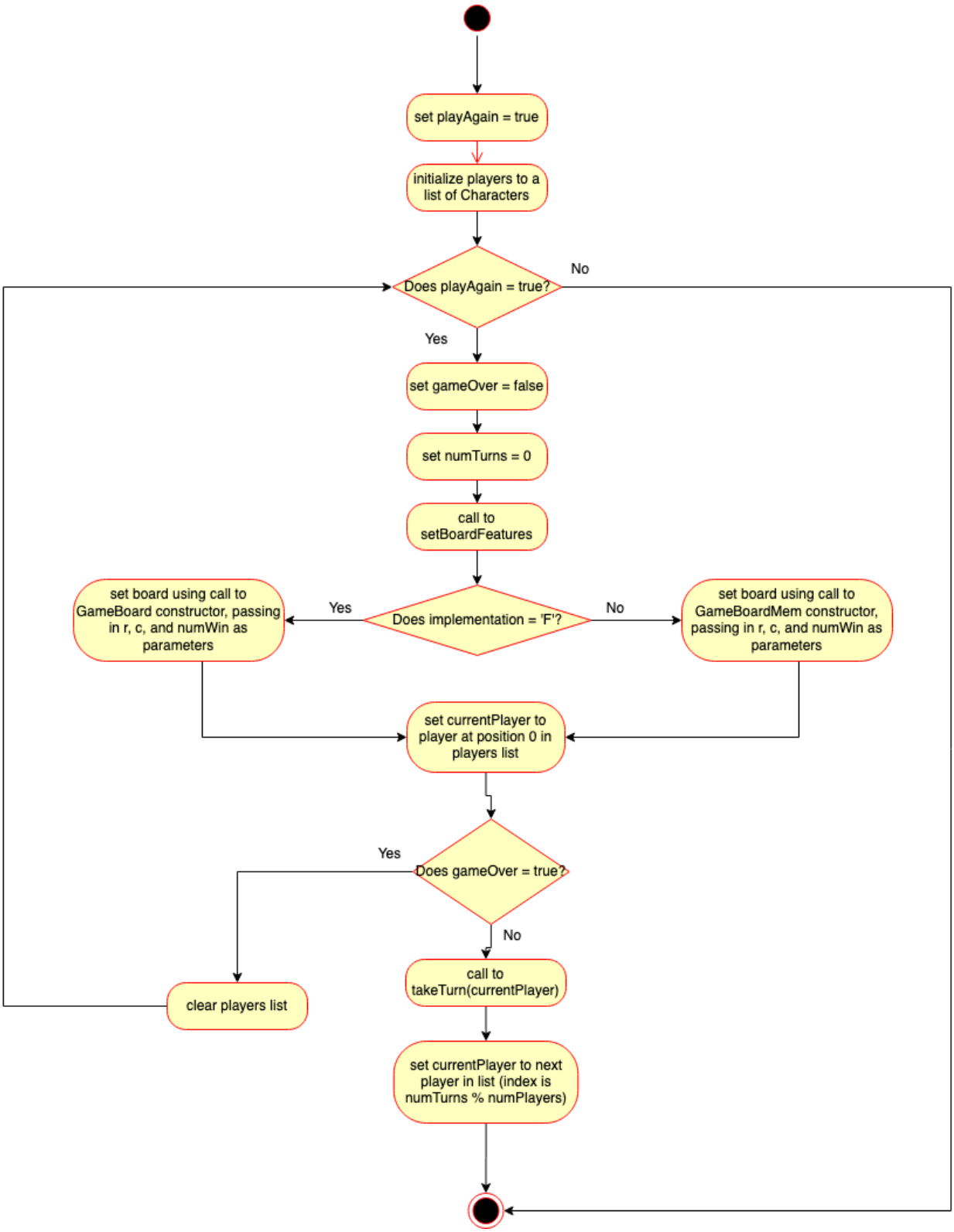

BoardPosition
- Row : int [1]
- Column : int [1]
+ BoardPosition(int, int): void
+ getRow(void) : int
+ getColumn(void) : int
+ equals(void) : bool
+ toString(void) : String


GameScreen
- players : List<Character> [1]
- r : int [1]
- c : int [1]
- numWin : int [1]
- numPlayers : int [1]
- numTurns : int [1]
- implementation : char [1]
- currentPlayer : char [1]
- MAXPLAYERS : int [1]
- MINPLAYERS : int [1]
- gameOver : boolean [1]
- playAgain : boolean [1]
- board : IGameBoard [1]
+ main(): void
+ gameWon() : void
+ gameDrawn(): void
+ takeTurn(char) : void
+ setBoardFeatures(void) : void

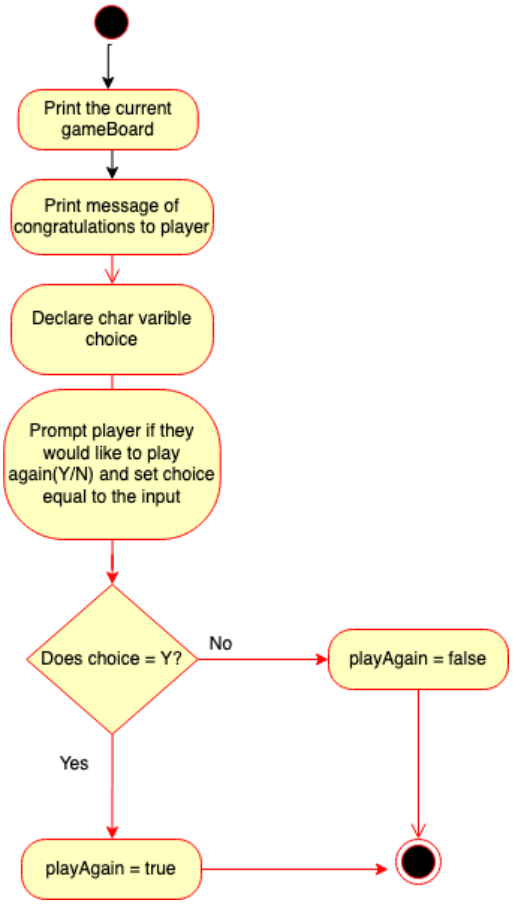
Activity Diagrams

GameScreen

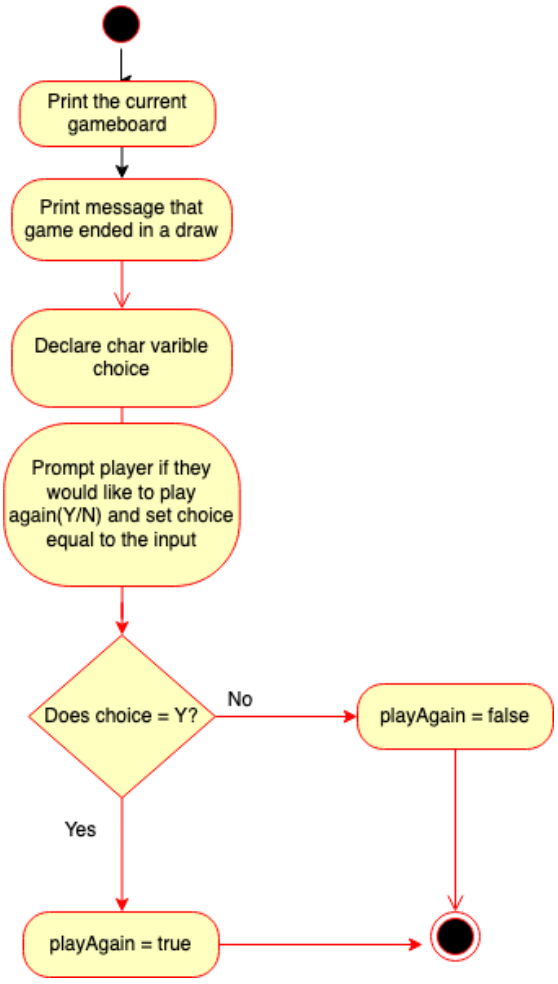
Main()



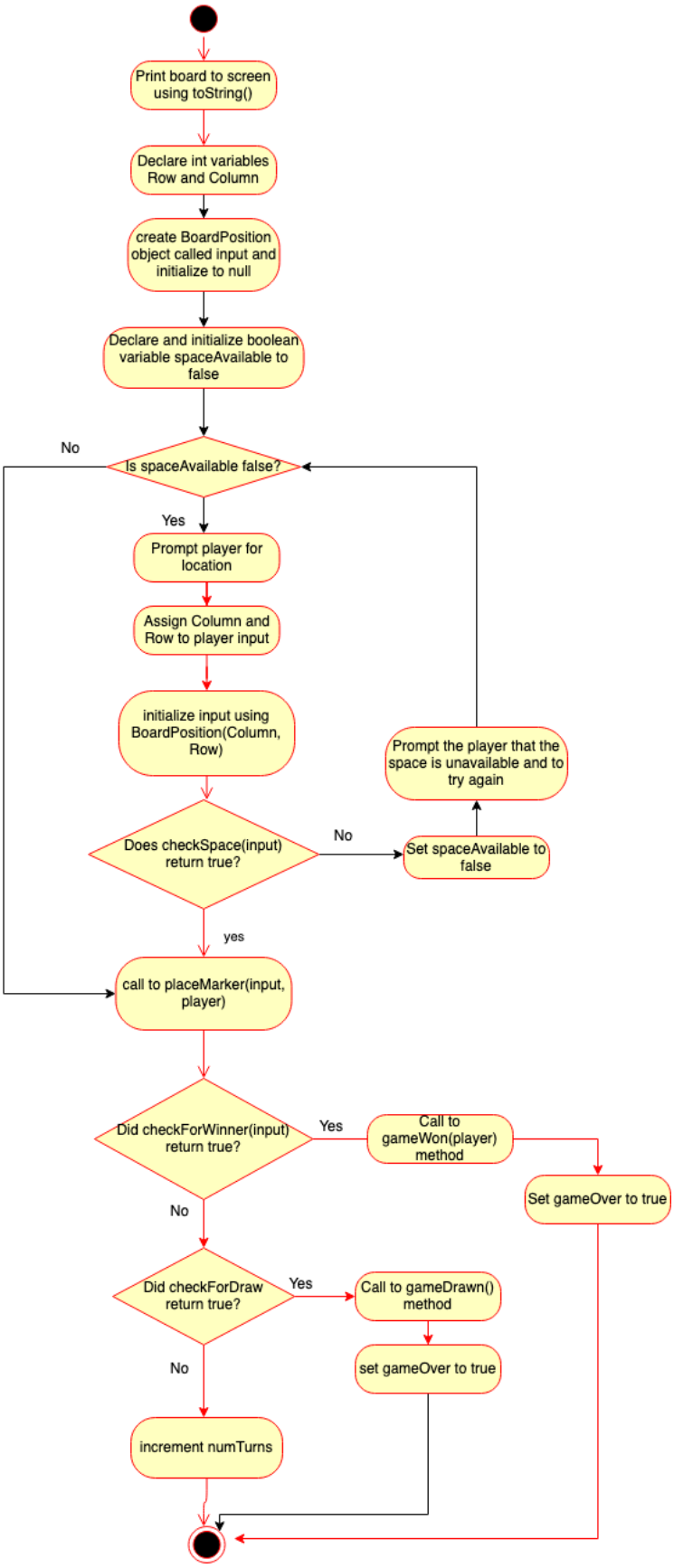
gameWon()



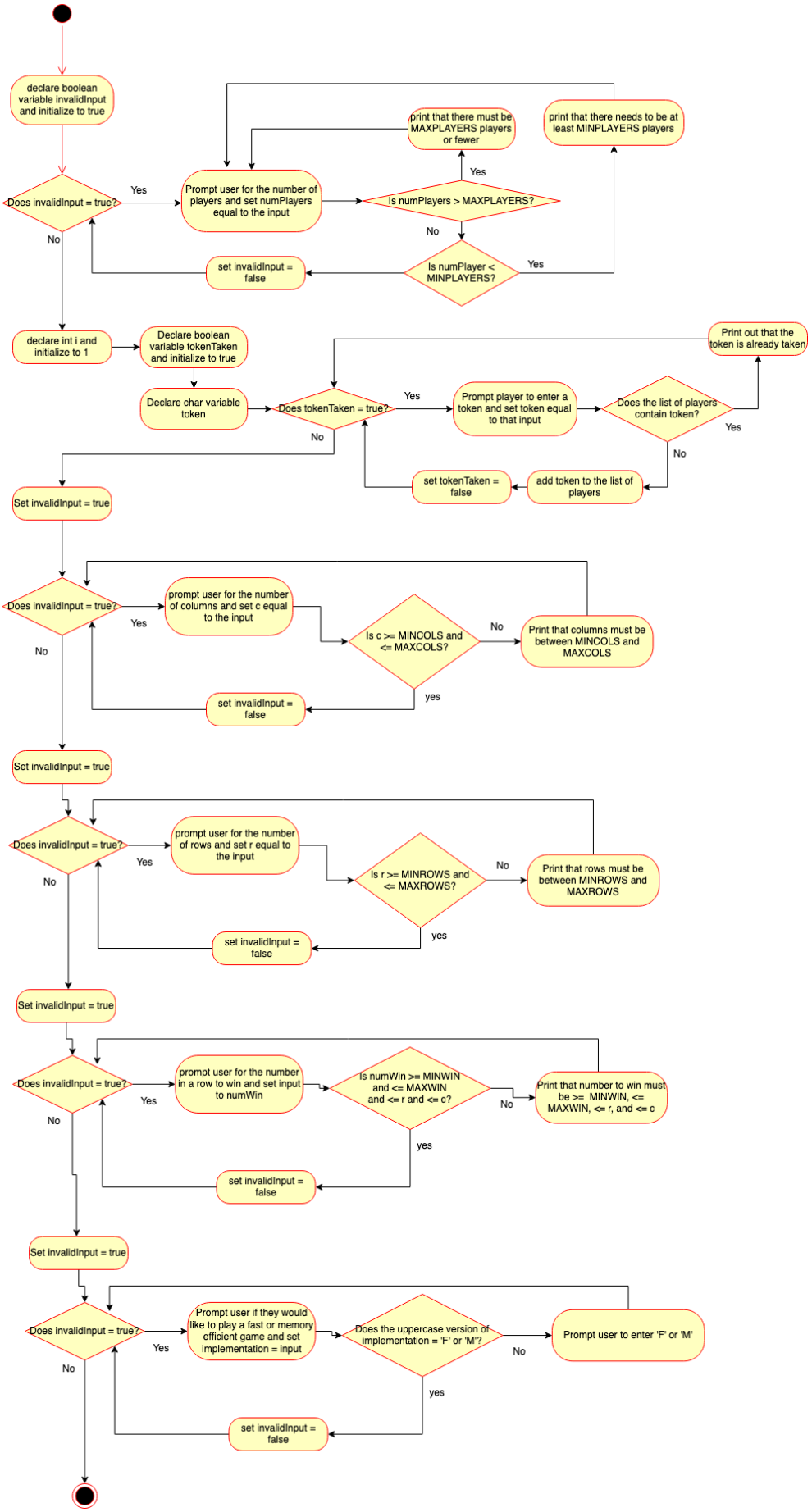
gameDrawn()



takeTurn(char player)

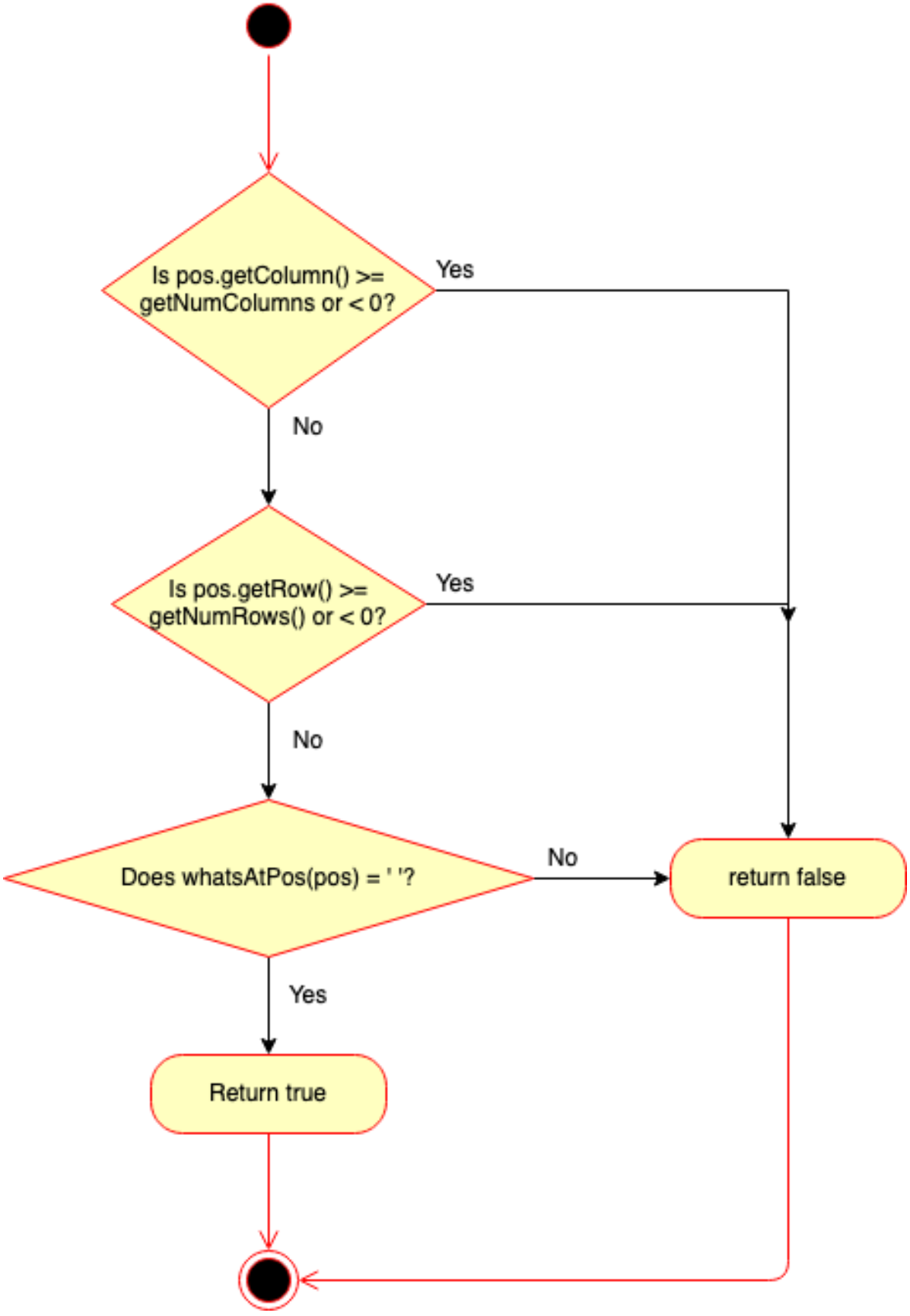


setBoardFeatures()

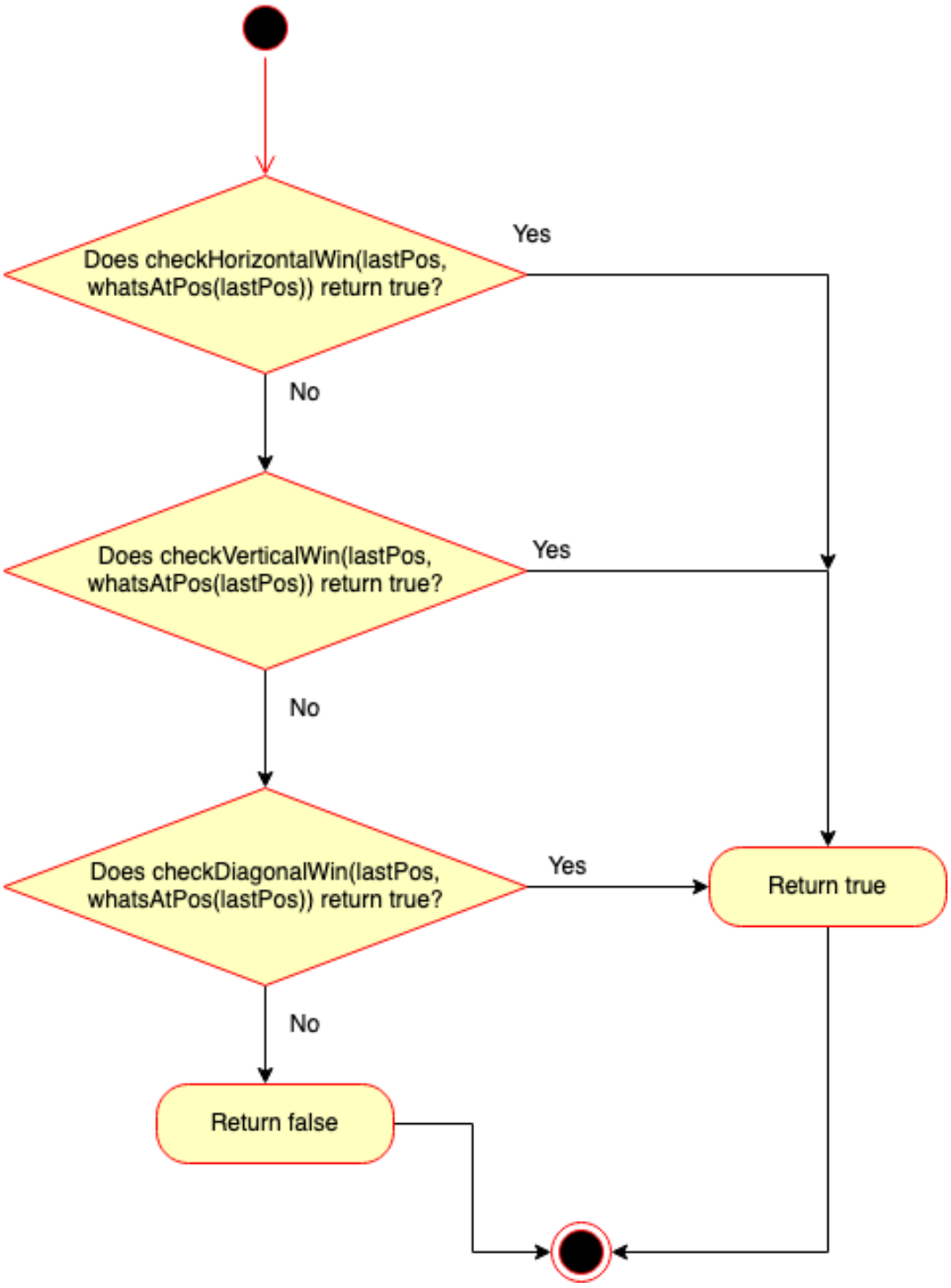


IGameBoard (default methods)

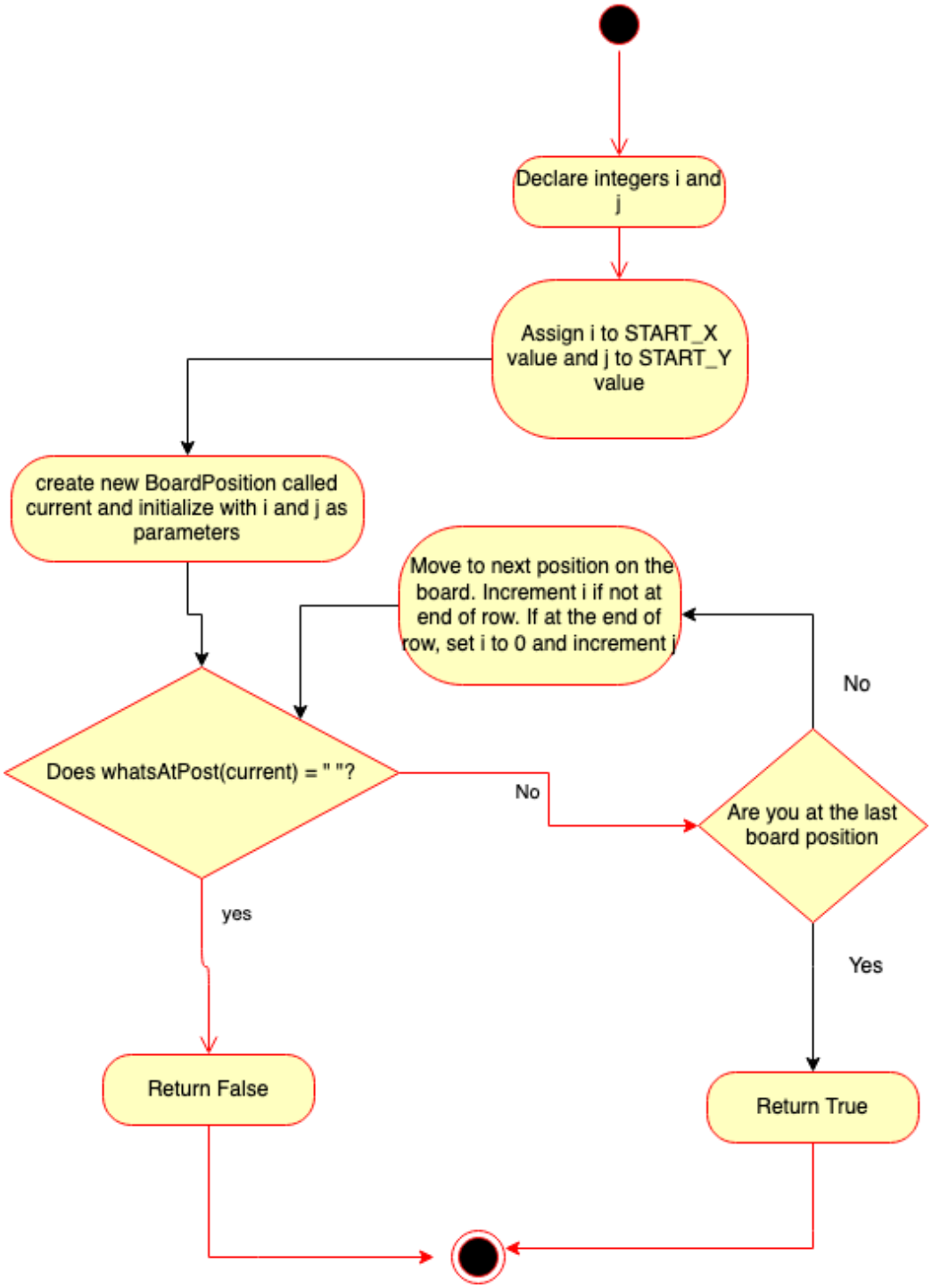
checkSpace(BoardPosition pos)



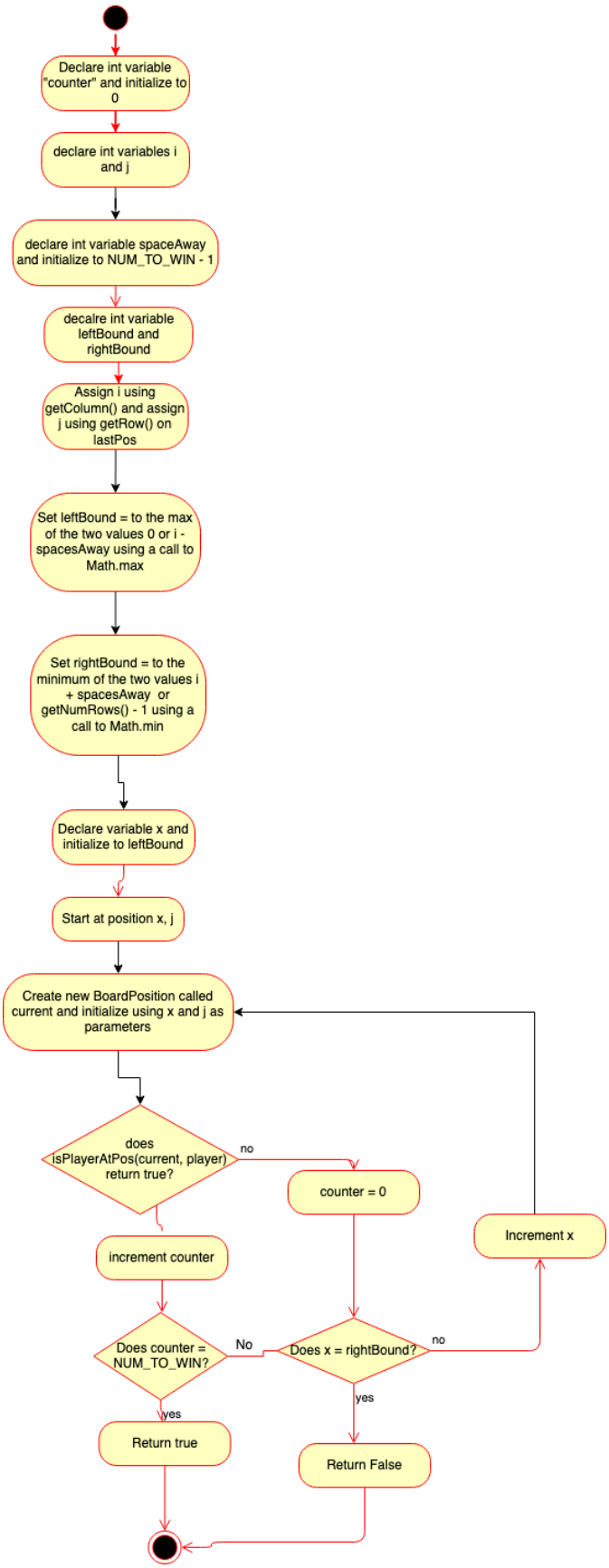
checkForWinner(BoardPosition lastPos)



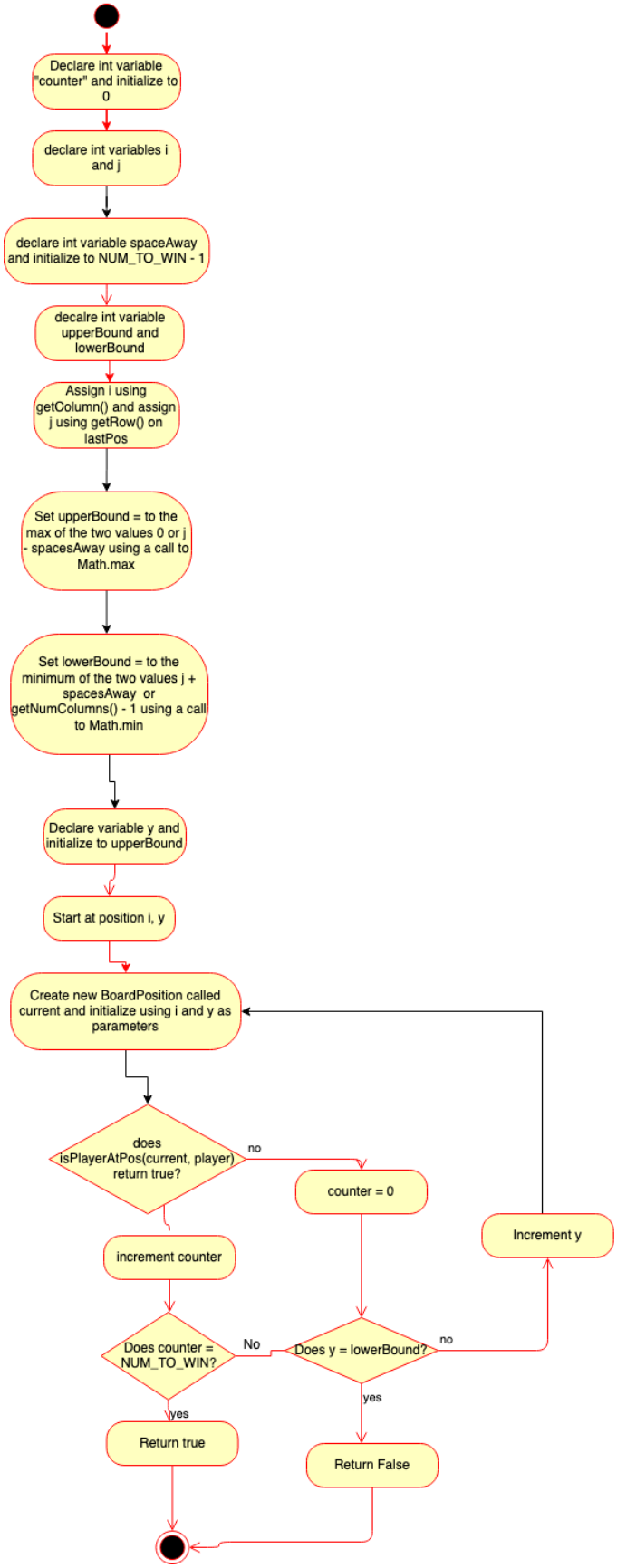
checkForDraw()



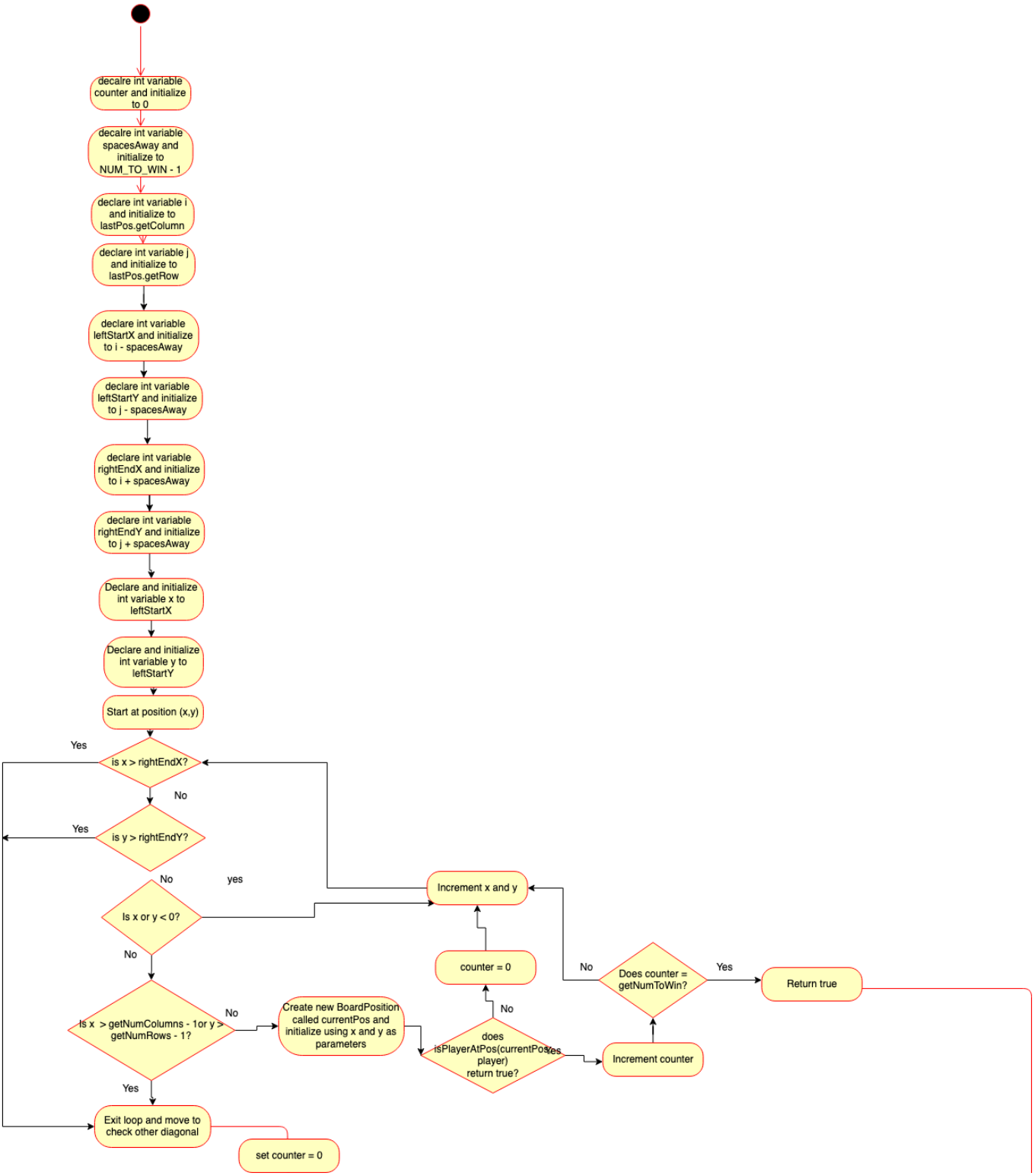
checkHorizontalWin(BoardPosition lastPos, char player)

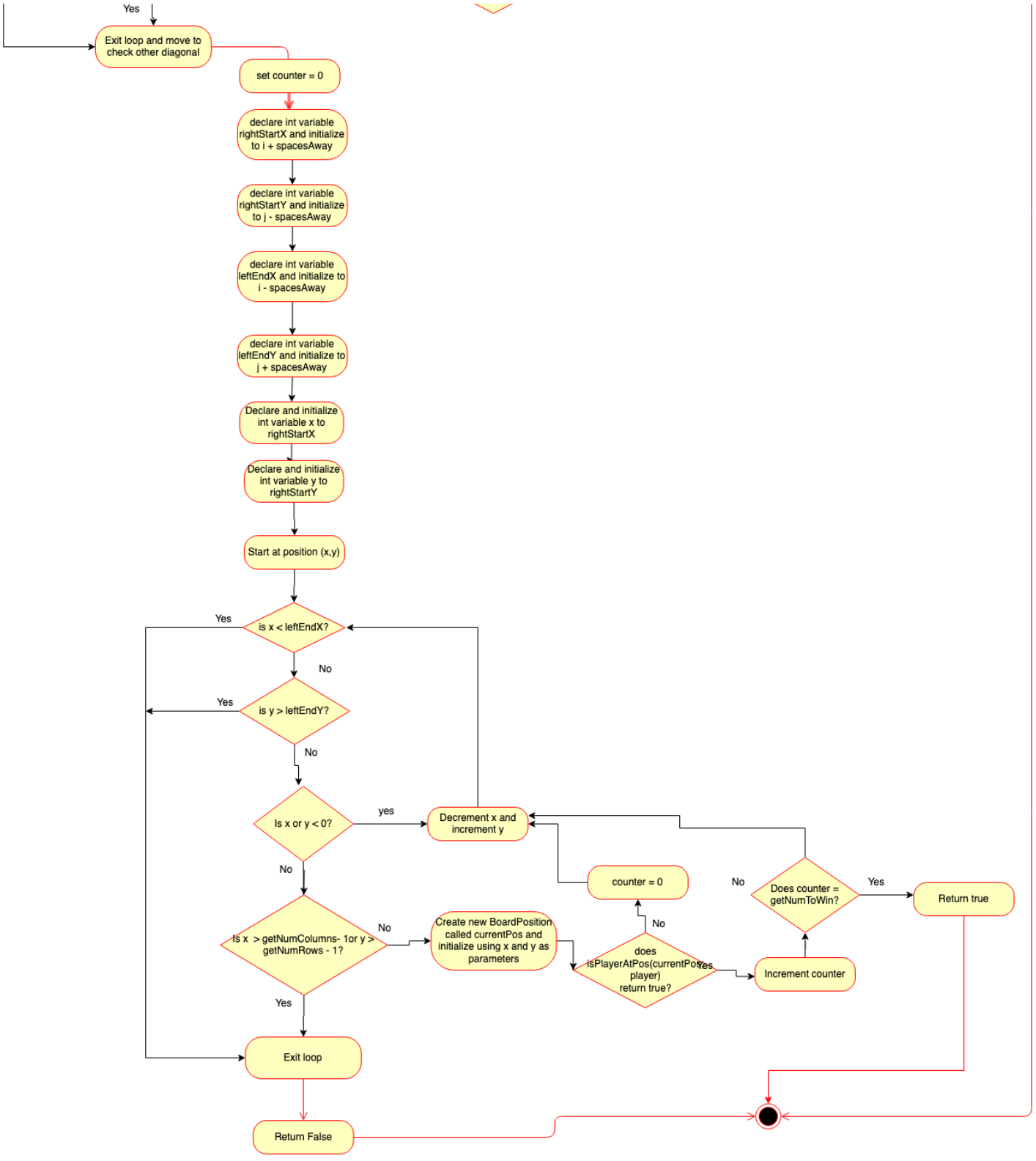


checkVerticallWin(BoardPosition lastPos, char player)

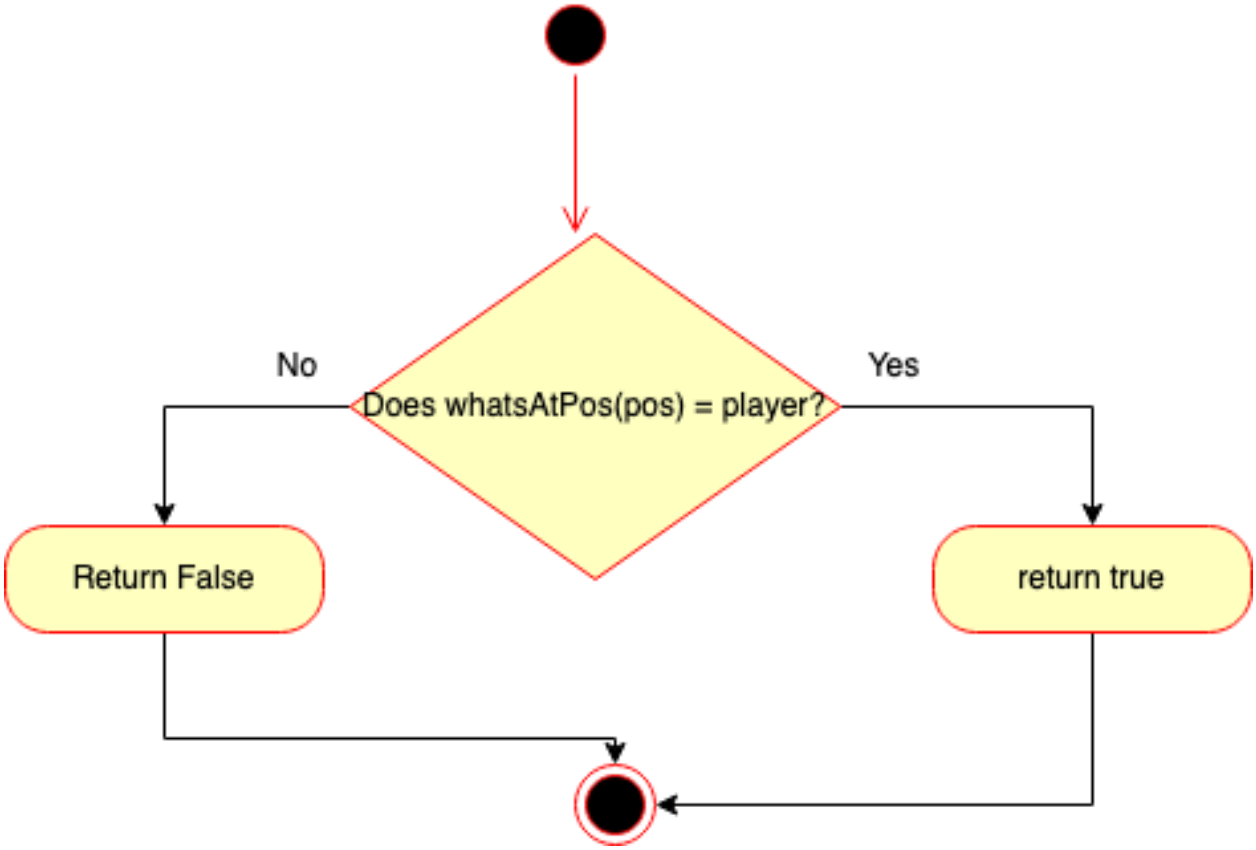


checkDiagonalWin(BoardPosition lastPos, char player)



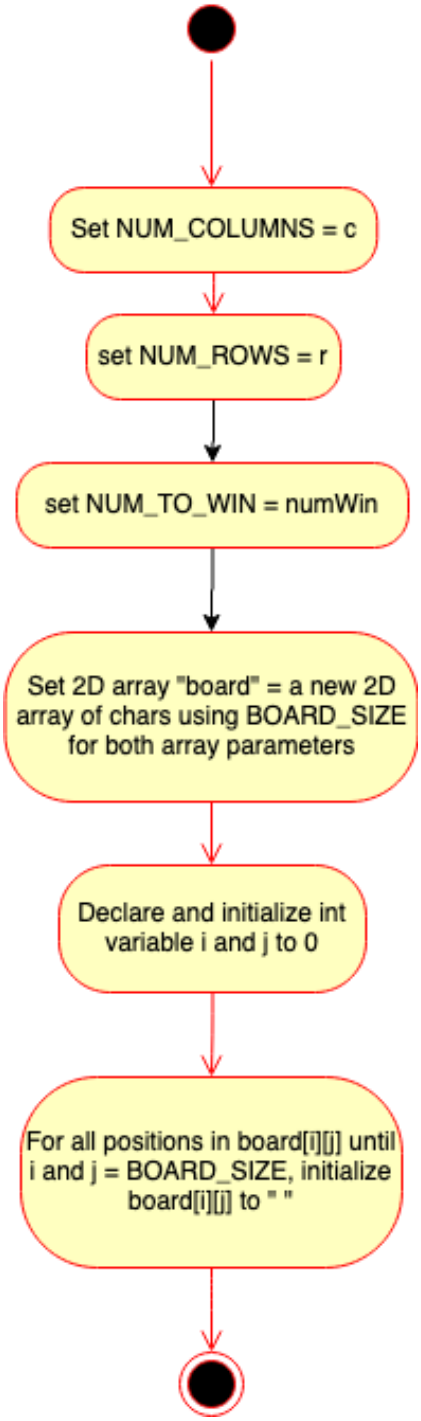

Yes
Exit loop and move to check other diagonal
set counter = 0
declare int variable rightStartX and initialize to i + spacesAway
declare int variable rightStartY and initialize to j - spacesAway
declare int variable leftEndX and initialize to i - spacesAway
declare int variable leftEndY and initialize to j + spacesAway
Declare and initialize int variable x to rightStartX
Declare and initialize int variable y to rightStartY
Start at position (x,y)
Yes
is x < leftEndX?
No
Yes
is y > leftEndY?
No
Is x or y < 0?
yes
Decrement x and increment y
No
Is x > getNumColumns- 1or y > getNumRows - 1?
No
Create new BoardPosition called currentPos and initialize using x and y as parameters
does isPlayerAtPos(currentPos, player) return true?
No
Yes
counter = 0
Increment counter
No
Does counter = getNumToWin?
Yes
Return true
Yes
Exit loop
Return False

isPlayerAtPos(BoardPosition pos, char player)

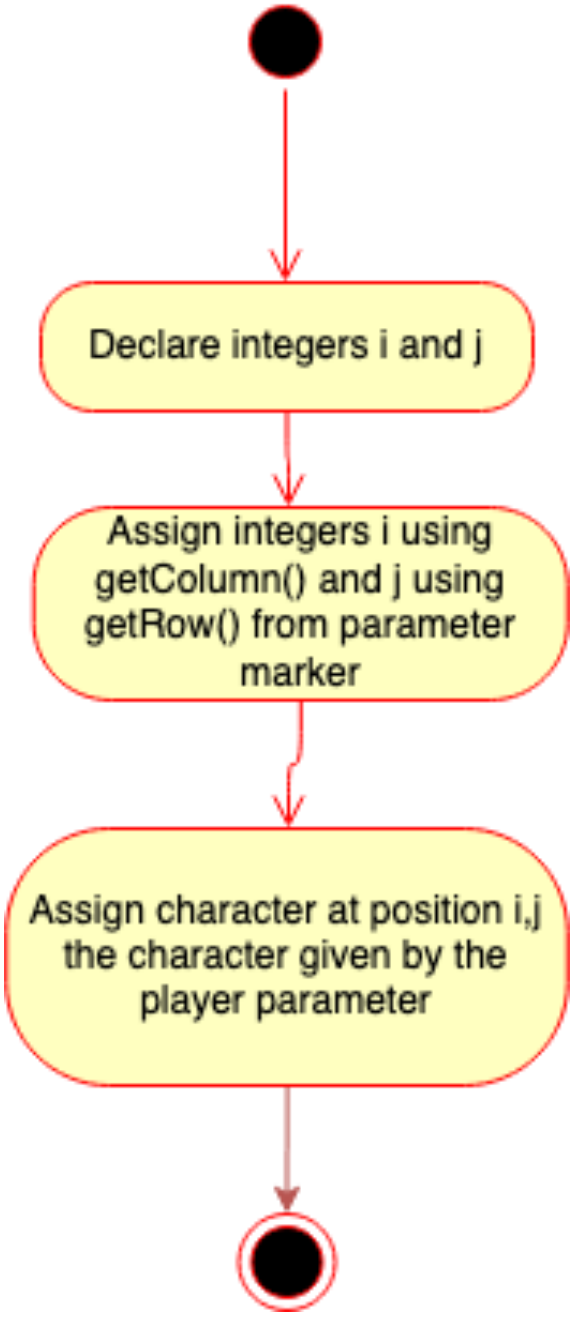


GameBoard – Primary Implementations

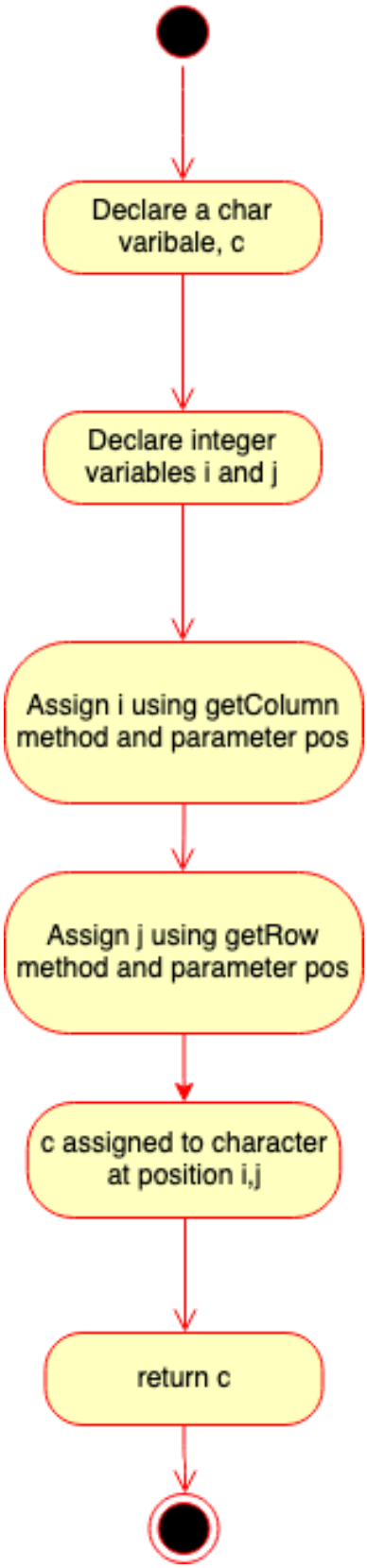
GameBoard(int r, int c, int numWin) (constructor)



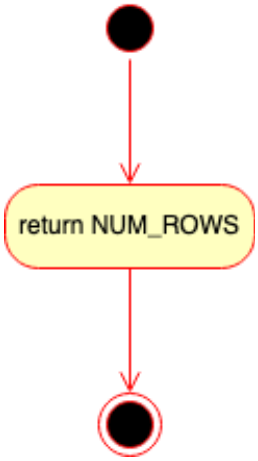
placeMarker(BoardPosition marker, char player)



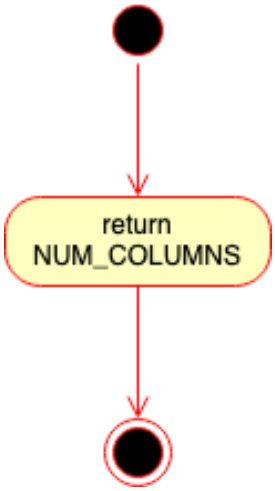
whatsAtPos(BoardPosition pos)



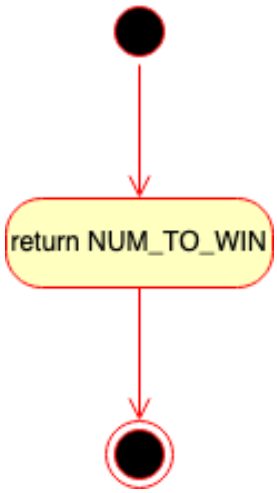
getNumRows()



getNumColumns()

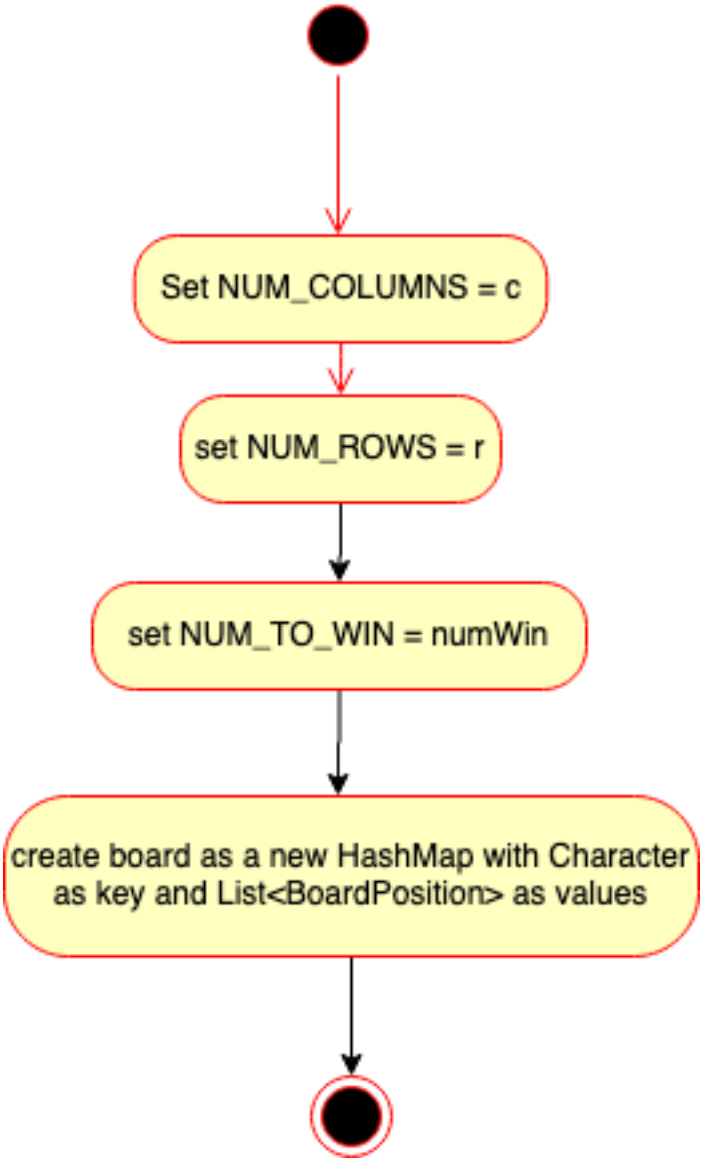


getNumToWin()

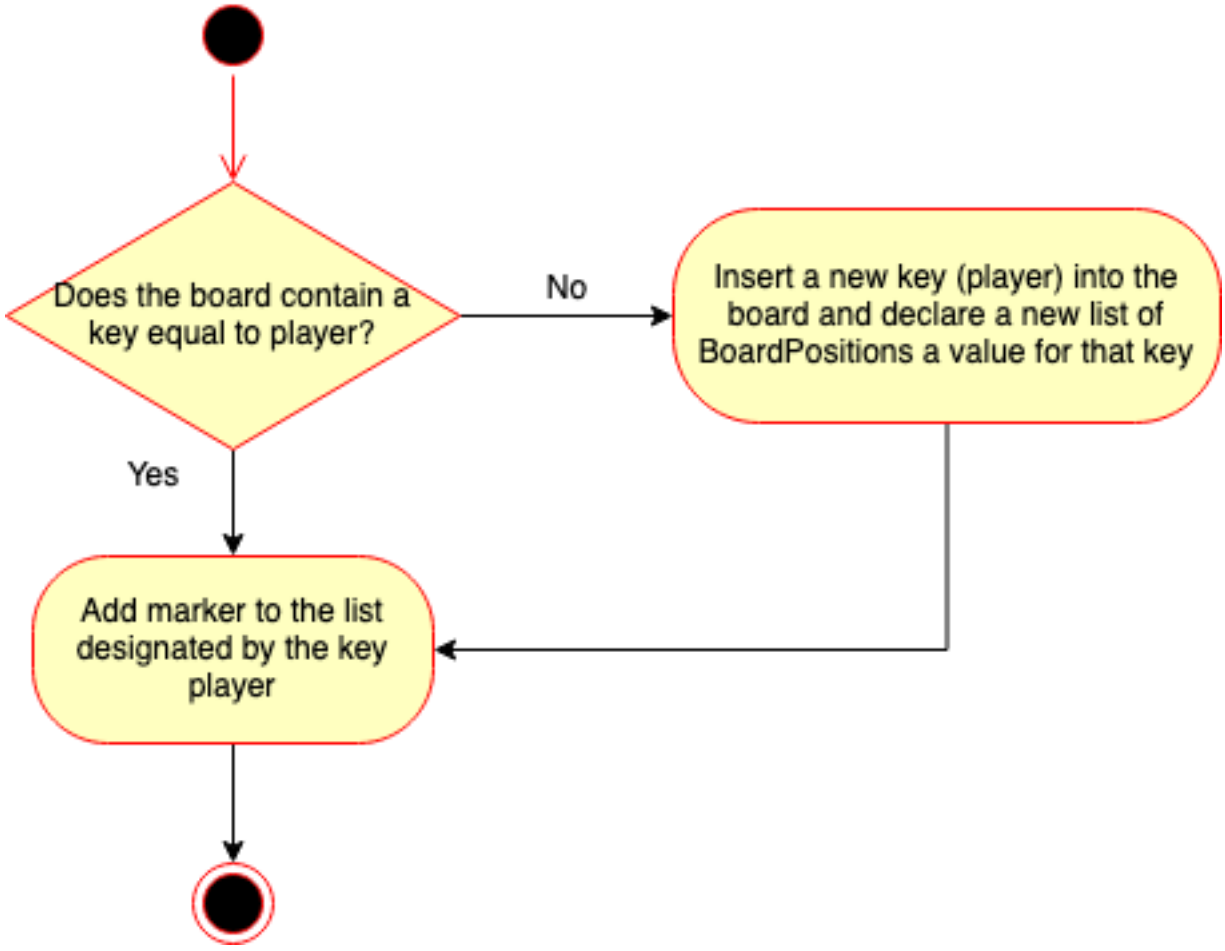


GameBoardMem – Primary Implementations

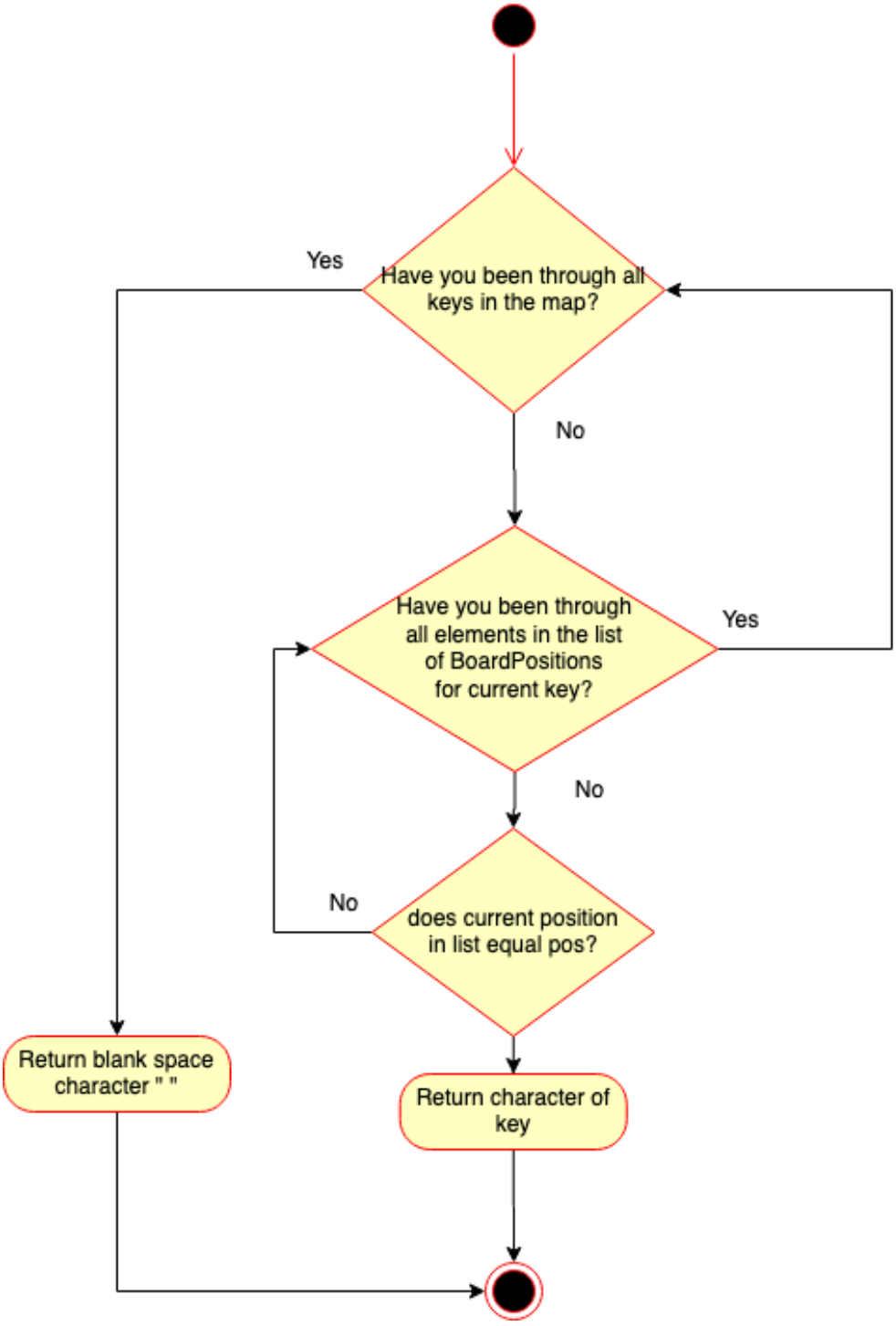
GameBoardMem(int r, int c, int numWin) (constructor)



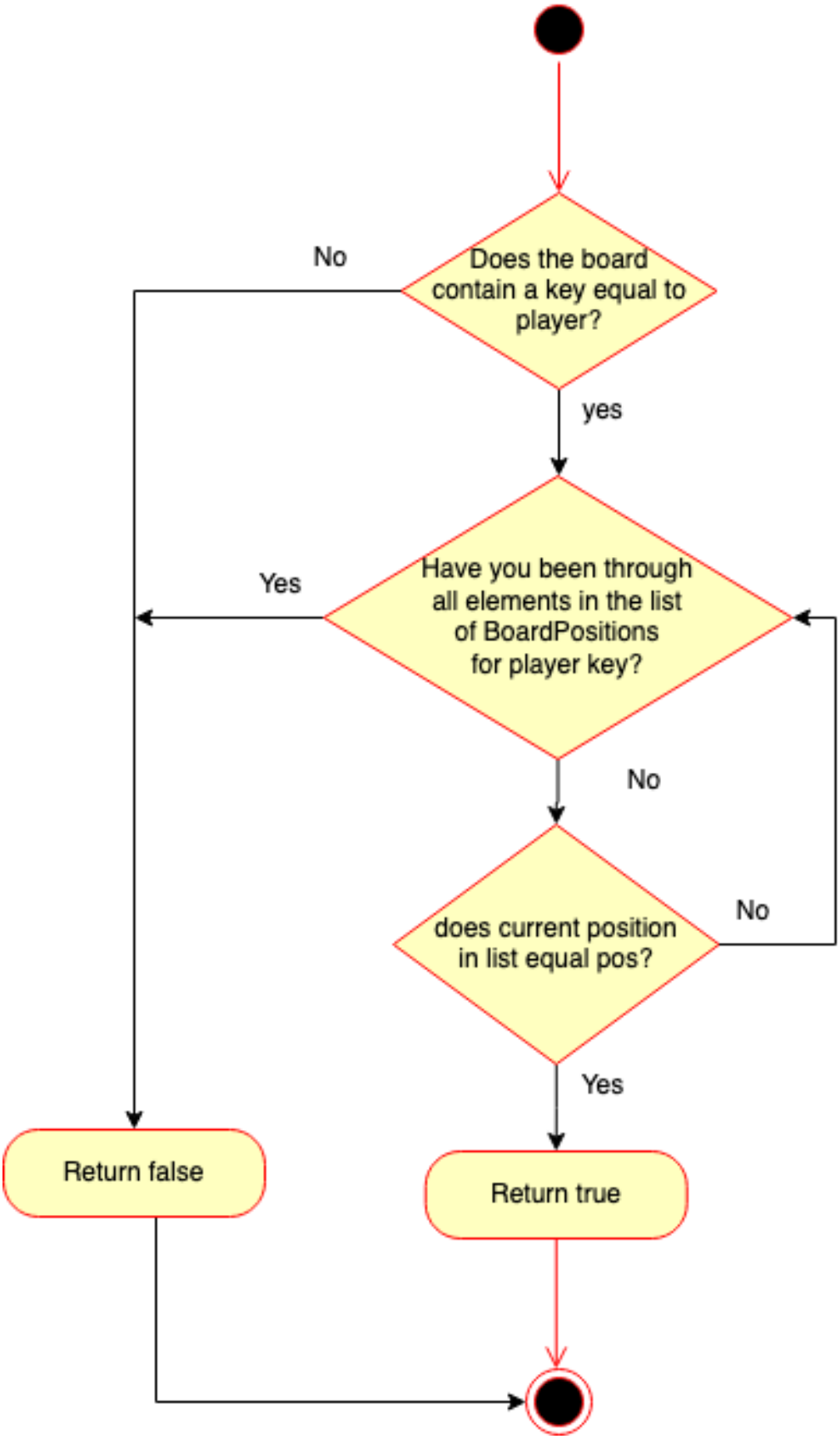
placeMarker(BoardPosition marker, char player)



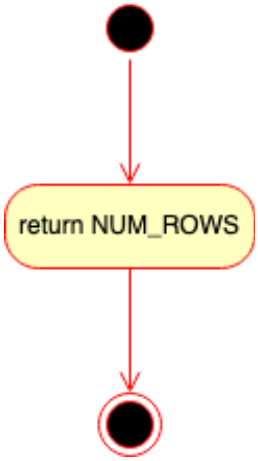
whatsAtPos(BoardPosition pos)



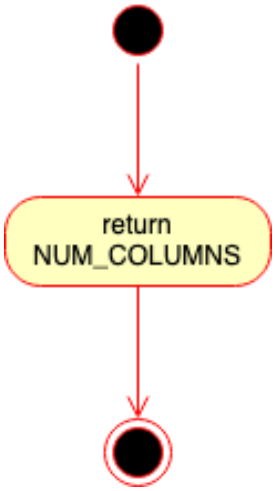
Override of isPlayerAtPos(BoardPosition pos, char player)



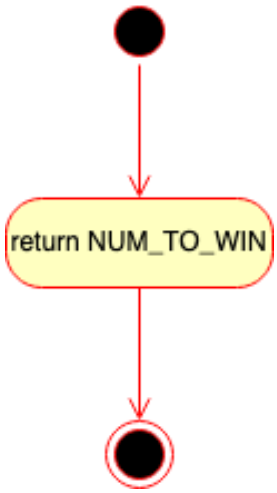
getNumRows()



getNumColumns()

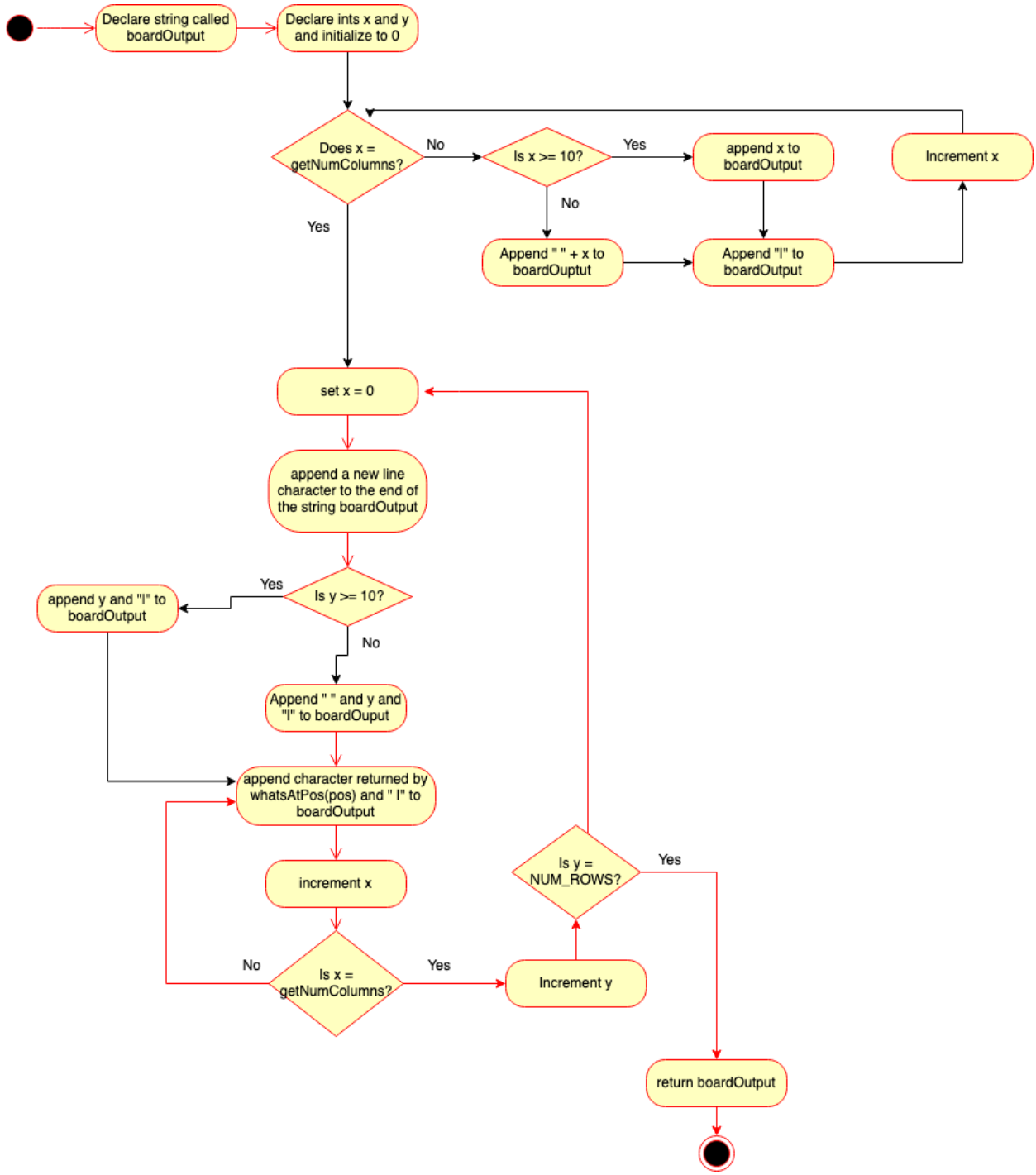


getNumToWin()



AbsGameBoard

toString()



Deployment

Running the extended Tic-Tac-Toe is based off the makefile. To use the makefile navigate to the project directory in the terminal window and type the following commands:

- To compile the code type `make`
- To run the program and play the game type `make run`
- To remove the compiled (.class) files type `make clean`