David Gleaton

CPSC 2150

Homework 2

Section 003

1. Requirements
    a. Functional Requirements
        i. Must be able to change the size of the game board
            1. User Story: As a player, I can set the game board to my desired size in order to play the game as I want
        ii. Must be able to change the number of tokens in a row in order to win
            1. User Story: As a player, I can set the winning limit in order to play the game as I see fit
        iii. Must be able to print the game board in a readable fashion
            1. User Story: As a player, I can read the game board easily in order to make my next moves in the game
        iv. Must be able to input the amount of rows
            1. User Story: As a player, I can set the rows to whatever I in order to play the game size I want
        v. Must be able to input the amount of columns
            1. User Story: As a player, I can set the columns to whatever I in order to play the game size I want
        vi. Must be able to input the win number
            1. User Story: As a player, I can set the win number to whatever I in order to set the rules to my benefit
        vii. Must be able to print out gameboard and input prompt
            1. User Story: As a player, I can view the gameboard so I can see the state of play and where to place my token.
        viii. Must be able to input token into specified column
            1. User Story: The player enters their token into specified column to attempt to align 4 of the same tokens.
        ix. Must be able to place token in the gameboard and check for victory
            1. User Story: As a player, I can see if I won so that I can lord it over the other player
        x. Must be able to ask for another round of gameplay
            1. User Story: As a player, I can choose to play again for as long as I want

        //Hold over from Homework 1, I'm still not sure if these should be outlined or not

        (These are back calculations with no display to the user, so I am unsure how to write user stories for these. I will list them anyway for clarity)

        xi. User(program) checks if column is empty.

1. User Story: User(program) checks to see if the column is empty in order to accurately place the token

xii. If true, User(program) will place token in lowest row in column

1. User Story: User(program) will place the toke into the lowest slot in order to represent the current state of play

xiii. User(program) checks for win

1. User(program) checks for horizontal win in order to see if either player won
2. User(program) checks for vertical win in order to see if either player won
3. User(program) checks for diagonal win in order to see if either player won

xiv. User(program) checks for tie

1. User(program) checks for tie in order to determine if the game has ended without a victory

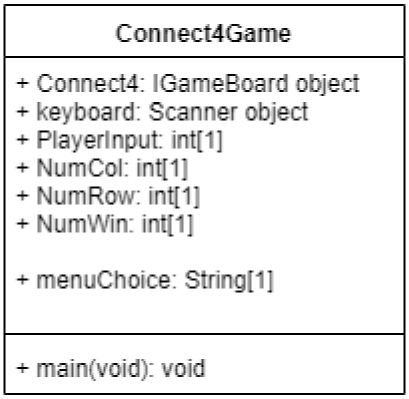
b. Non-Functional Requirements

i. No magic numbers
ii. Good comments must be provided
iii. Written contracts
iv. Must be able to run on Unix
v. Must be written in Java
vi. No dead code
vii. Game starts with player X
viii. Must compile
ix. Must have a working makefile

2. Testing = N/A
3. Deployment
   a. Unzip file
   b. Enter the command “make”
   c. Enter the command “make run”
4. Design
   a. UML Classes

Next Page

i. ConnectX class



Connect4Game

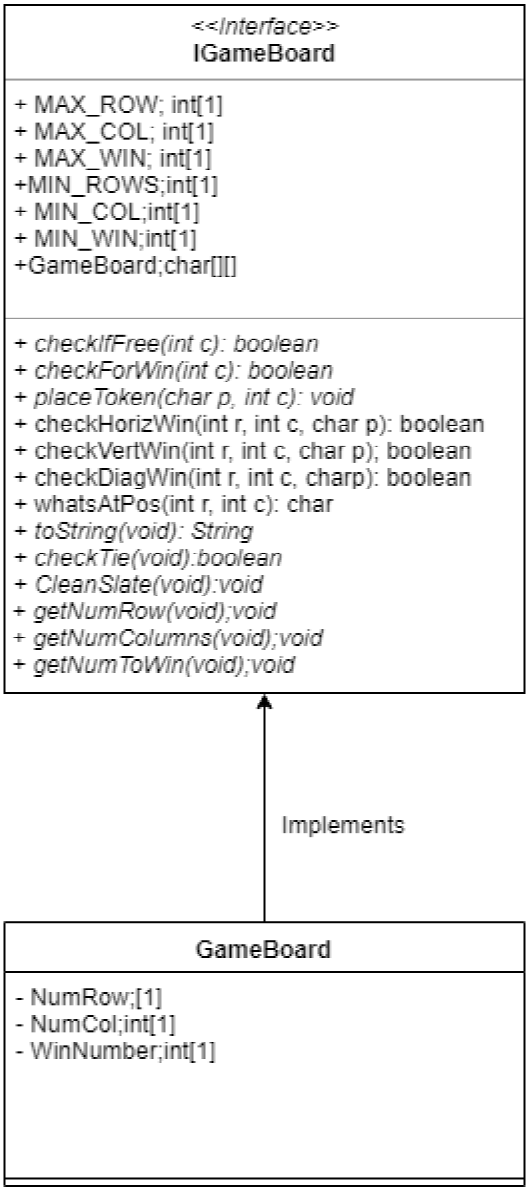
+ Connect4: IGameBoard object
+ keyboard: Scanner object
+ PlayerInput: int[1]
+ NumCol: int[1]
+ NumRow: int[1]
+ NumWin: int[1]

+ menuChoice: String[1]

+ main(void): void

ii.

iii. IGameBoard interface and GameBoard classes



<<Interface>>
IGameBoard

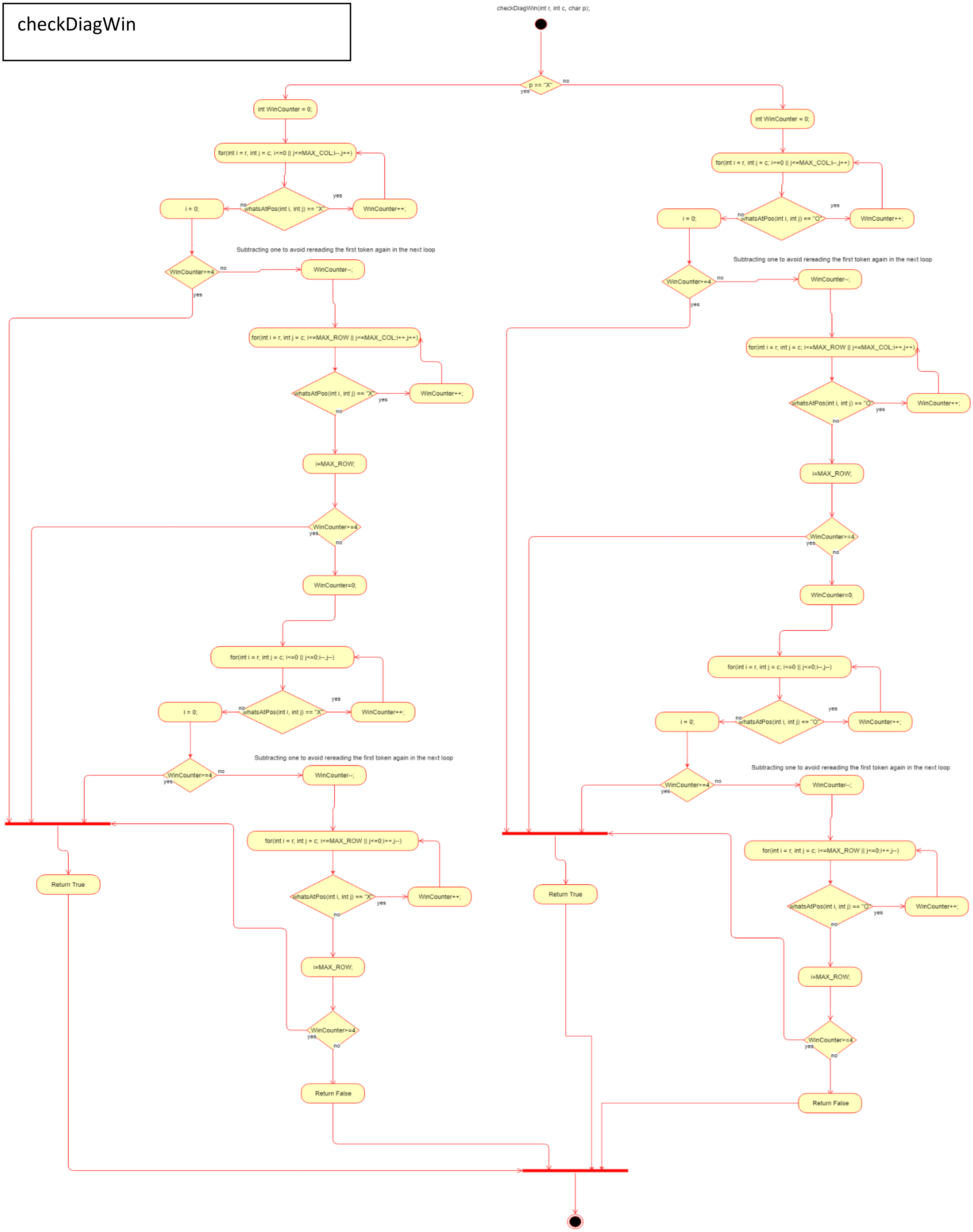
+ MAX_ROW; int[1]
+ MAX_COL; int[1]
+ MAX_WIN; int[1]
+MIN_ROWS;int[1]
+ MIN_COL;int[1]
+ MIN_WIN;int[1]
+GameBoard;char[][]

+ *checkIfFree(int c): boolean*
+ *checkForWin(int c): boolean*
+ *placeToken(char p, int c): void*
+ checkHorizWin(int r, int c, char p): boolean
+ checkVertWin(int r, int c, char p); boolean
+ checkDiagWin(int r, int c, charp): boolean
+ whatsAtPos(int r, int c): char
+ *toString(void): String*
+ *checkTie(void):boolean*
+ *CleanSlate(void):void*
+ *getNumRow(void);void*
+ *getNumColumns(void);void*
+ *getNumToWin(void);void*

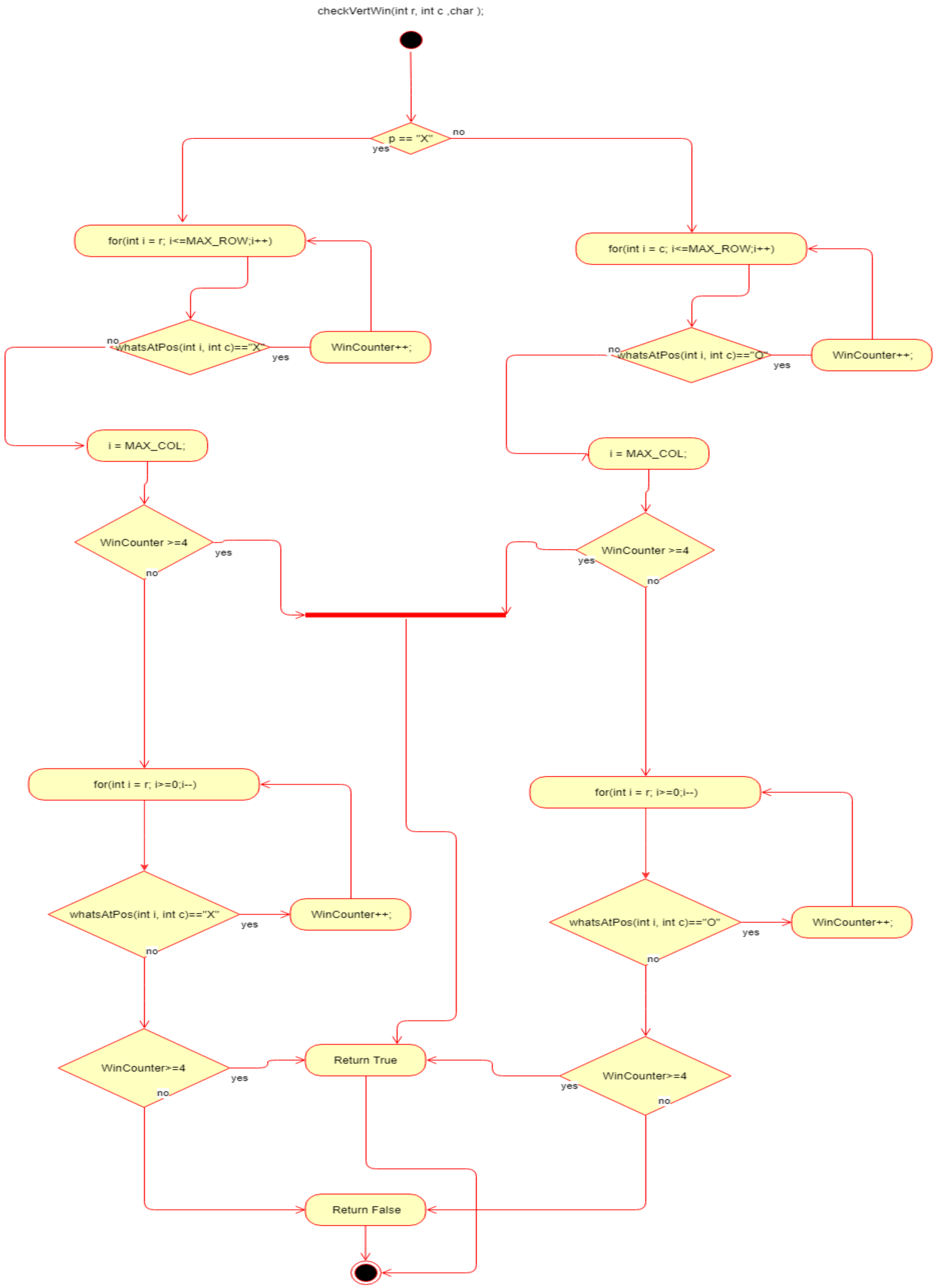
GameBoard

- NumRow;[1]
- NumCol;int[1]
- WinNumber;int[1]

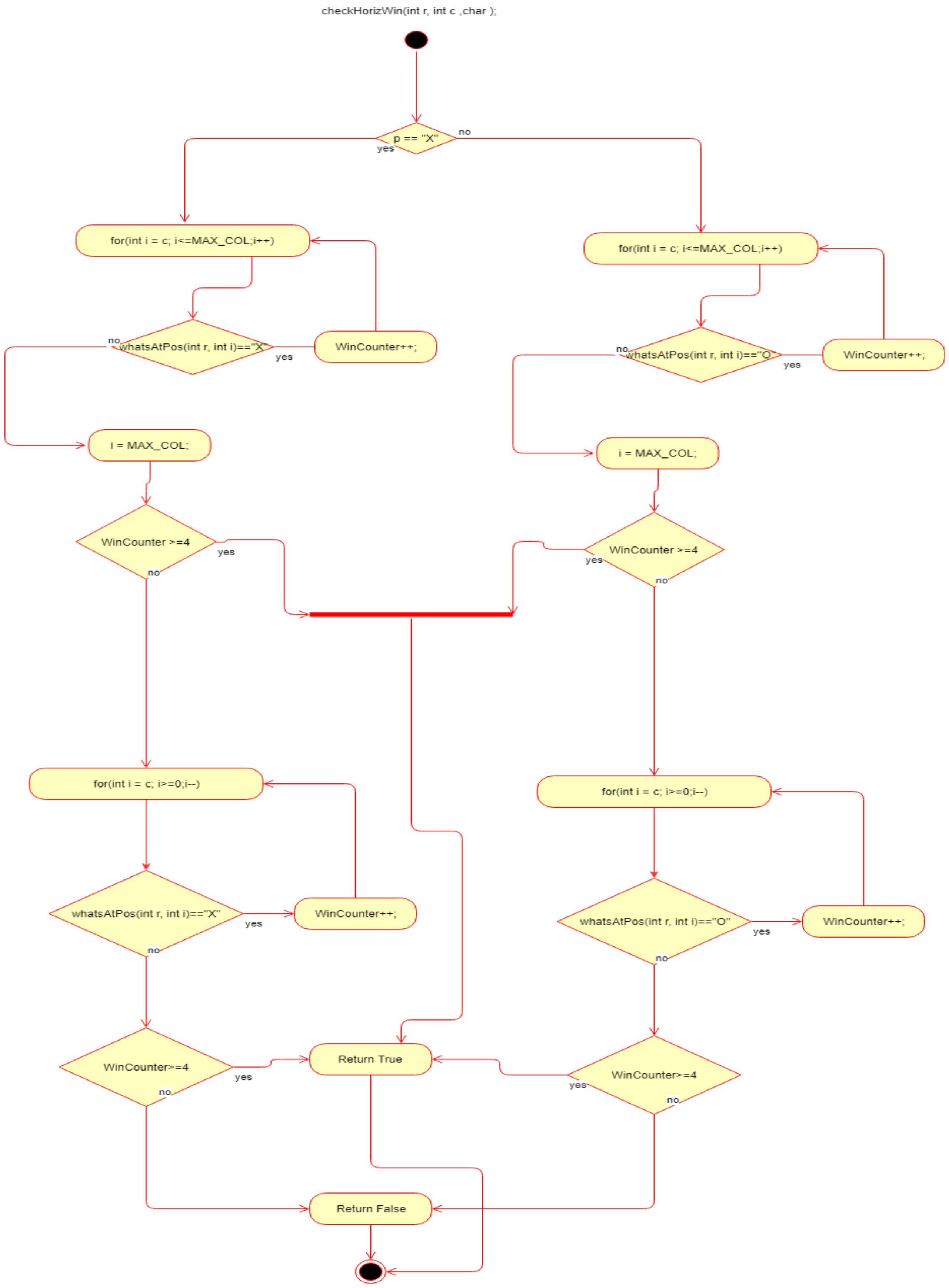
iv. Below are the Default classes within the IGameBoard Interface checkDiagWin


checkDiagWin(int r, int c, char p);
p == "X"
yes
no
int WinCounter = 0;
for(int i = r, int j = c; i<=0 || j<=MAX_COL;i--,j++)
whatsAtPos(int i, int j) == "X"
yes
no
WinCounter++;
i = 0;
Subtracting one to avoid rereading the first token again in the next loop
WinCounter>=4
no
yes
WinCounter--;
for(int i = r, int j = c; i<=MAX_ROW || j<=MAX_COL;i++,j++)
whatsAtPos(int i, int j) == "X"
yes
no
WinCounter++;
i=MAX_ROW;
WinCounter>=4
yes
no
WinCounter=0;
for(int i = r, int j = c; i<=0 || j<=0;i--,j--)
whatsAtPos(int i, int j) == "X"
yes
no
WinCounter++;
i = 0;
Subtracting one to avoid rereading the first token again in the next loop
WinCounter>=4
no
yes
WinCounter--;
for(int i = r, int j = c; i<=MAX_ROW || j<=0;i++,j--)
whatsAtPos(int i, int j) == "X"
yes
no
WinCounter++;
i=MAX_ROW;
WinCounter>=4
yes
no
Return True
Return False
int WinCounter = 0;
for(int i = r, int j = c; i<=0 || j<=MAX_COL;i--,j++)
whatsAtPos(int i, int j) == "O"
yes
no
WinCounter++;
i = 0;
Subtracting one to avoid rereading the first token again in the next loop
WinCounter>=4
no
yes
WinCounter--;
for(int i = r, int j = c; i<=MAX_ROW || j<=MAX_COL;i++,j++)
whatsAtPos(int i, int j) == "O"
yes
no
WinCounter++;
i=MAX_ROW;
WinCounter>=4
yes
no
WinCounter=0;
for(int i = r, int j = c; i<=0 || j<=0;i--,j--)
whatsAtPos(int i, int j) == "O"
yes
no
WinCounter++;
i = 0;
Subtracting one to avoid rereading the first token again in the next loop
WinCounter>=4
no
yes
WinCounter--;
for(int i = r, int j = c; i<=MAX_ROW || j<=0;i++,j--)
whatsAtPos(int i, int j) == "O"
yes
no
WinCounter++;
i=MAX_ROW;
WinCounter>=4
yes
no
Return True
Return False

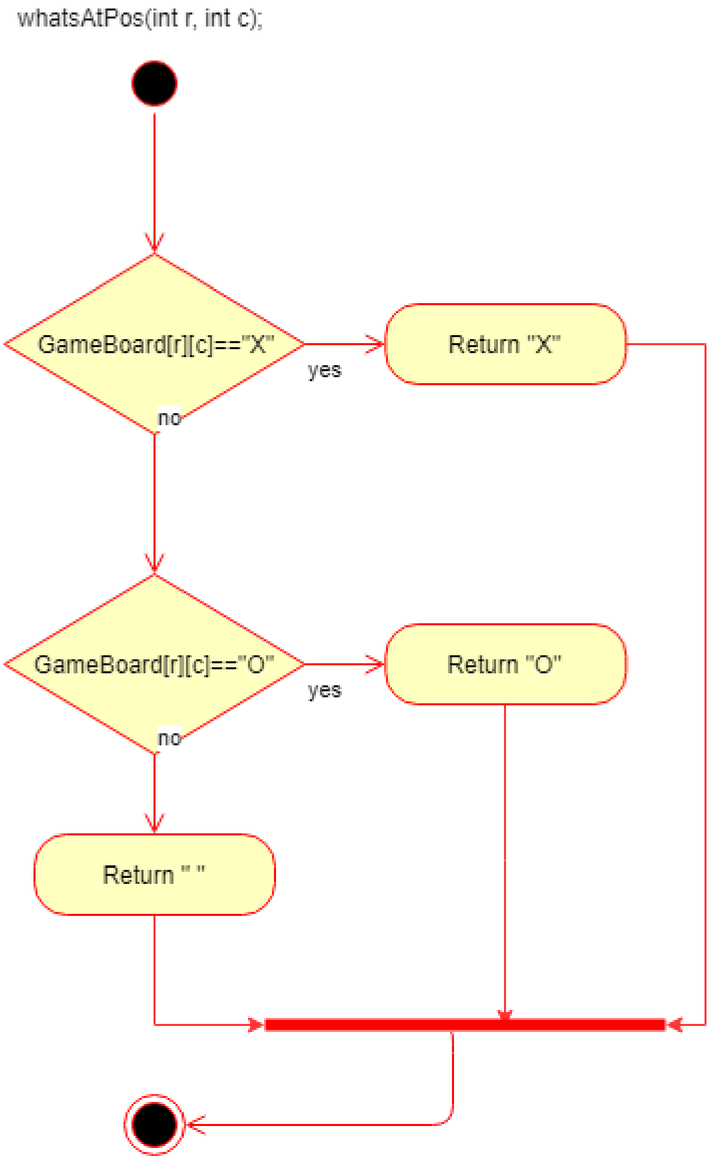
checkVertWin



checkHorizWin

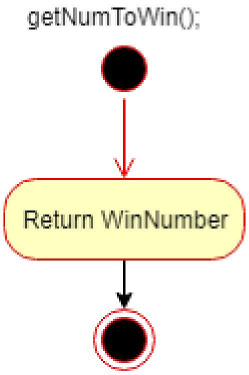


whatsAtPos


whatsAtPos(int r, int c);
GameBoard[r][c]=="X"
yes
no
Return "X"
GameBoard[r][c]=="O"
yes
no
Return "O"
Return " "

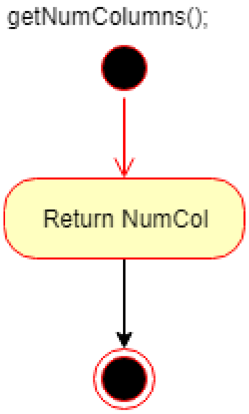
v. Below are the Methods implemented in the GameBoard Class

vi. getNumToWin



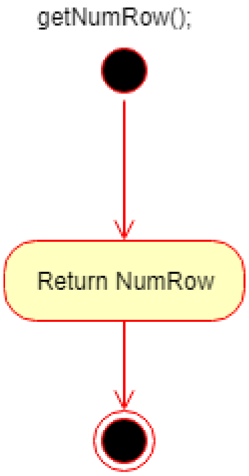
vii.

viii. getNumColumns



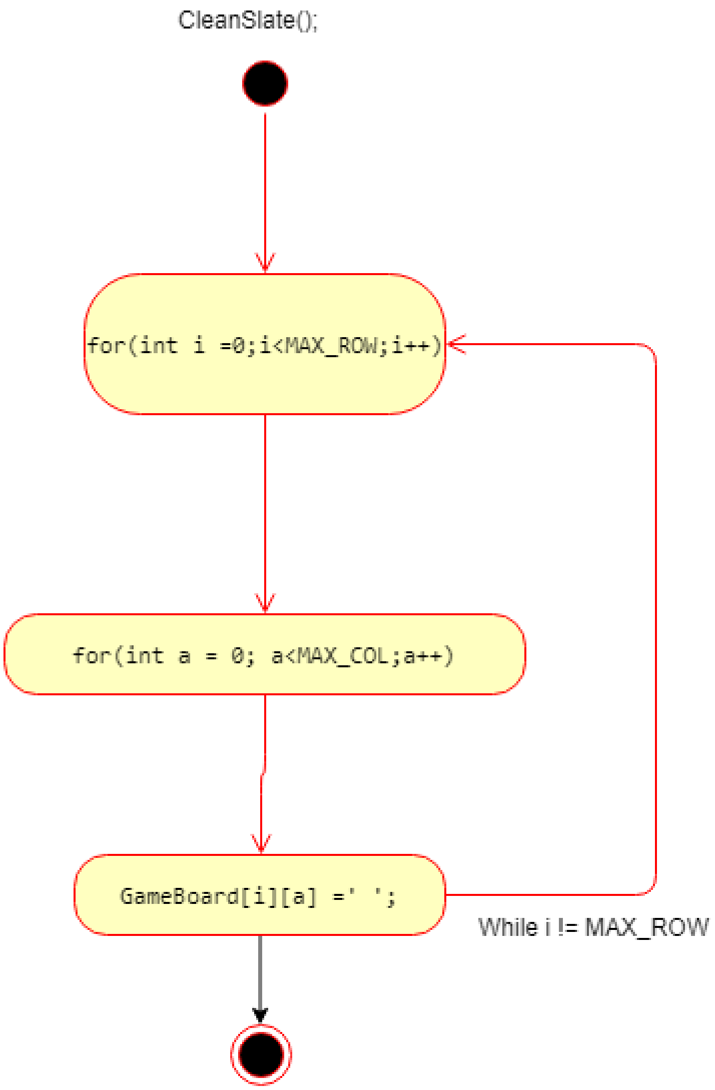
ix.

x. getNumRow



xi.

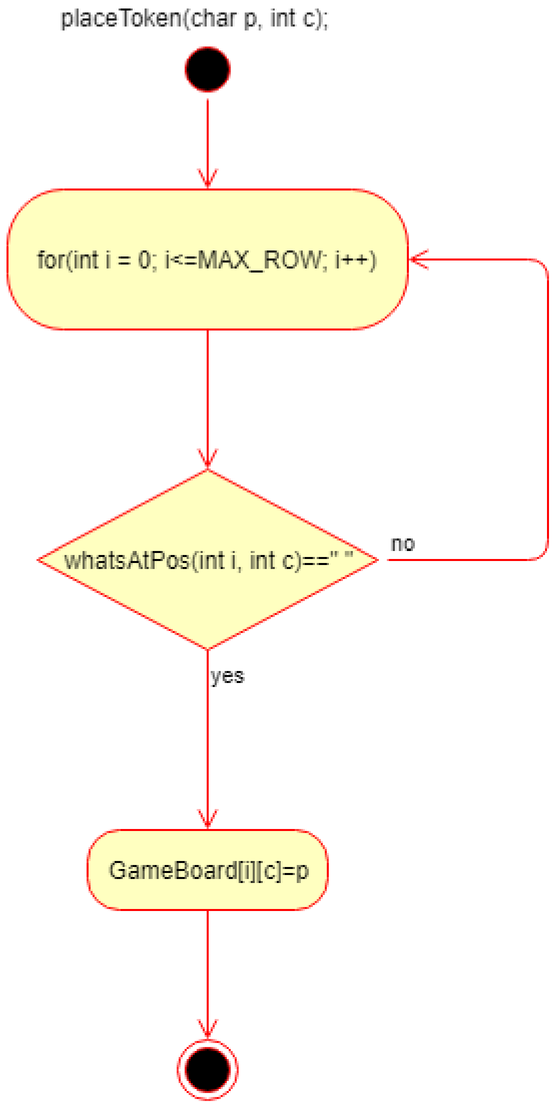
xii. CleanSlate



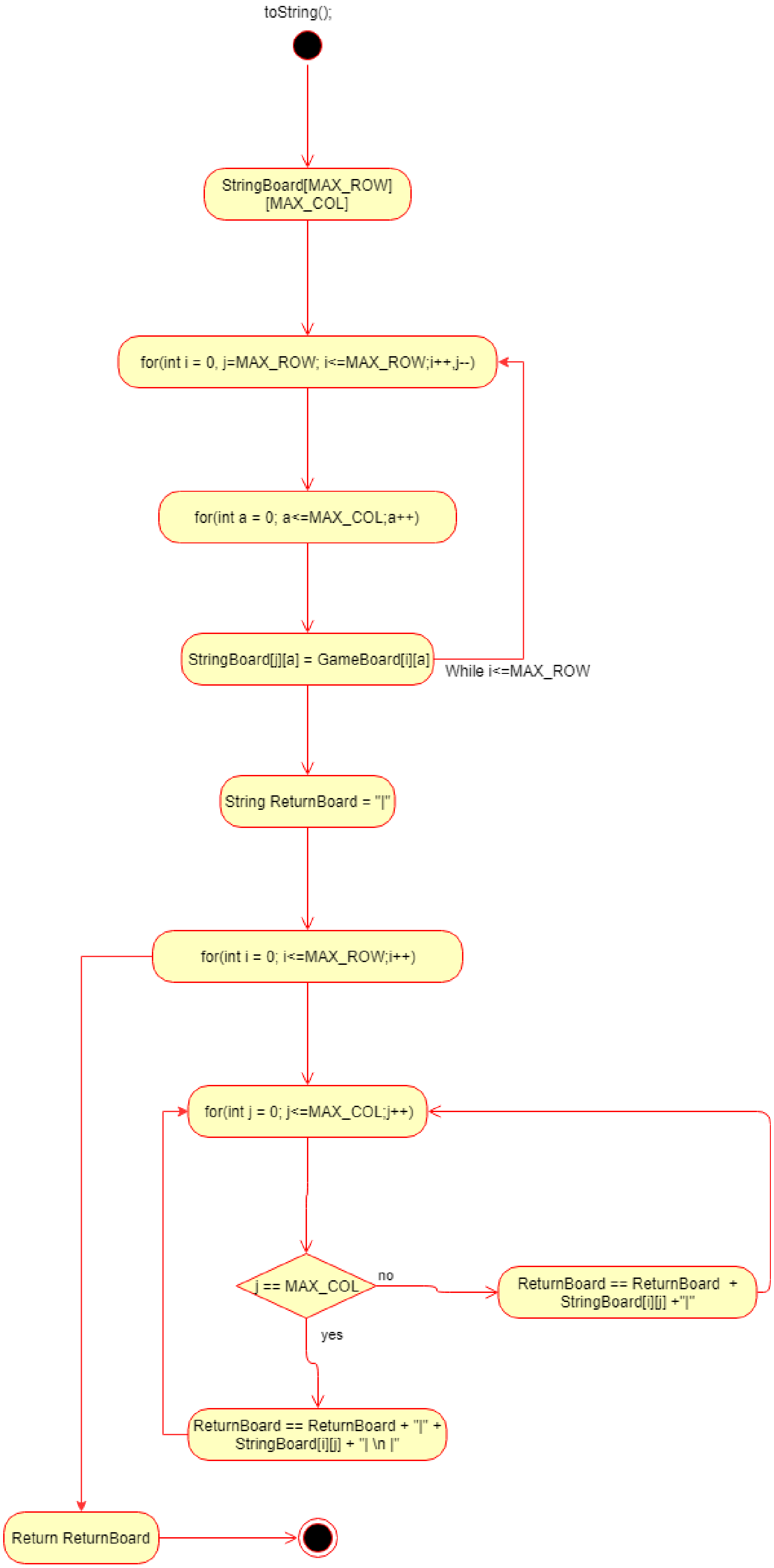
xiii.

xiv. placeToken

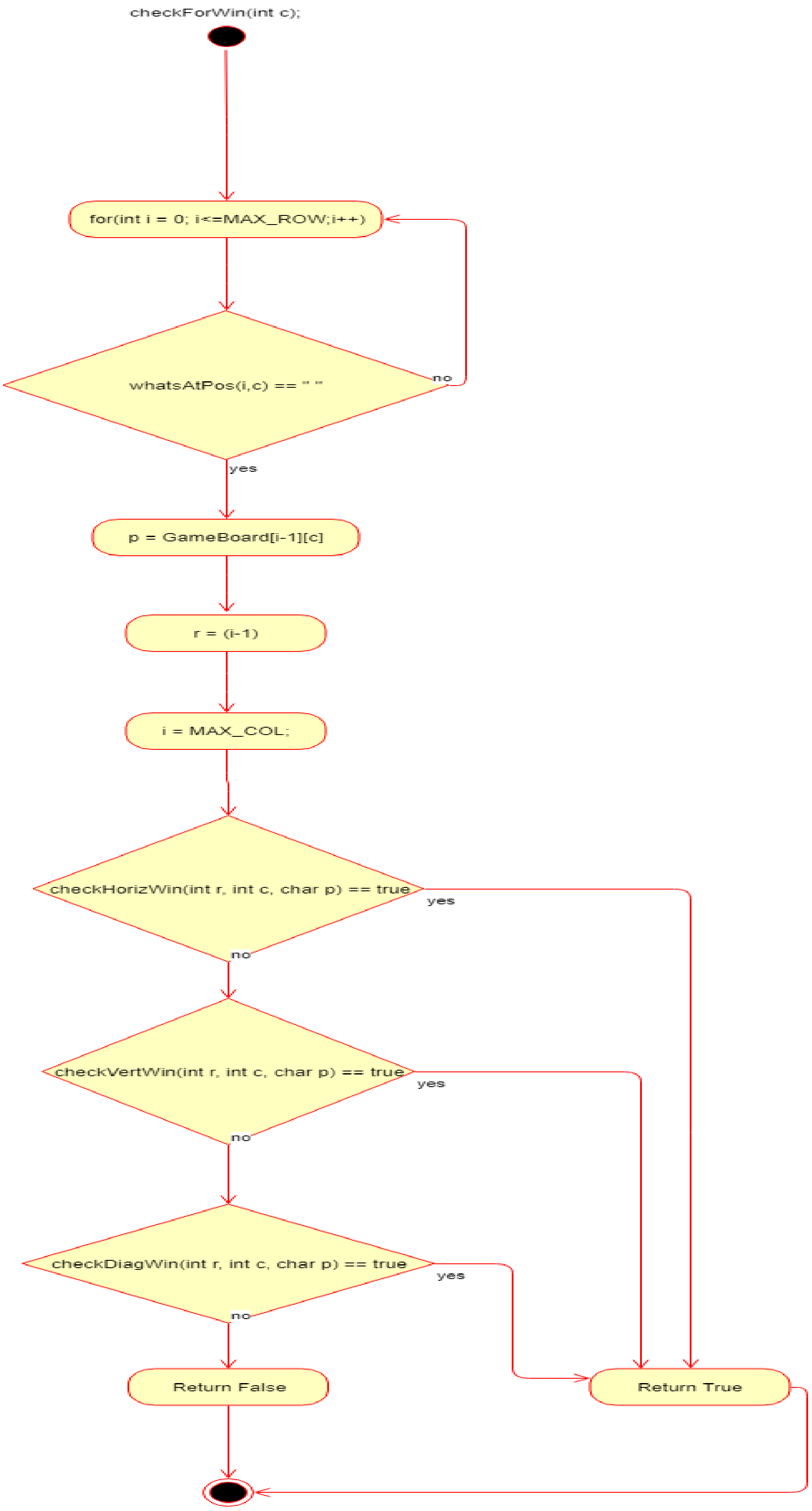
xv.


placeToken(char p, int c);
for(int i = 0; i<=MAX_ROW; i++)
whatsAtPos(int i, int c)==" "
no
yes
GameBoard[i][c]=p

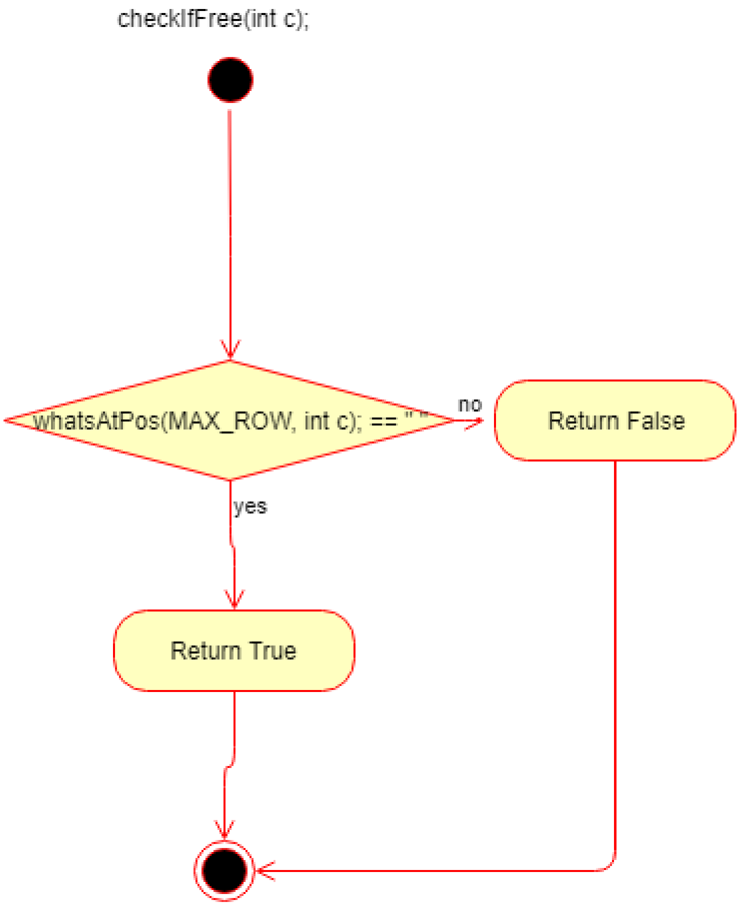
toString


toString();
StringBoard[MAX_ROW]
[MAX_COL]
for(int i = 0, j=MAX_ROW; i<=MAX_ROW;i++,j--)
for(int a = 0; a<=MAX_COL;a++)
StringBoard[j][a] = GameBoard[i][a]
While i<=MAX_ROW
String ReturnBoard = "|"
for(int i = 0; i<=MAX_ROW;i++)
for(int j = 0; j<=MAX_COL;j++)
j == MAX_COL
no
ReturnBoard == ReturnBoard +
StringBoard[i][j] +"|"
yes
ReturnBoard == ReturnBoard + "|" +
StringBoard[i][j] + "| \n |"
Return ReturnBoard

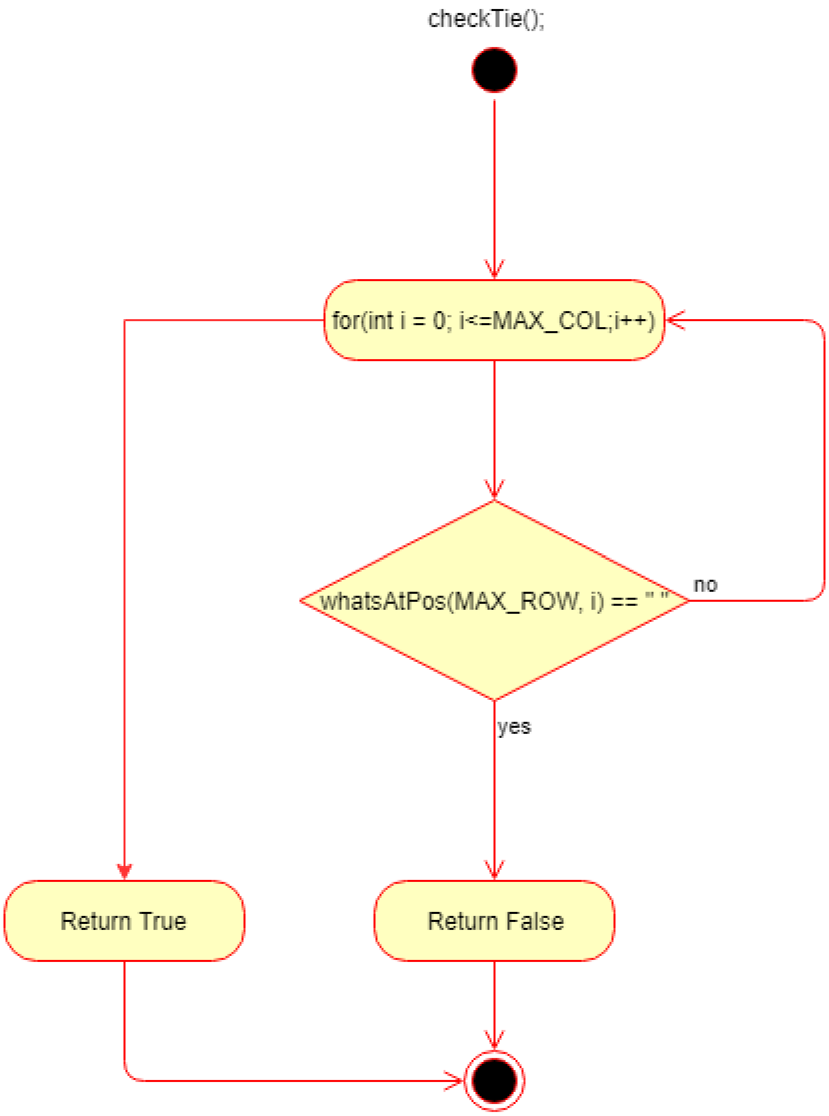
checkForWin


checkForWin(int c);
for(int i = 0; i<=MAX_ROW;i++)
whatsAtPos(i,c) == " "
no
yes
p = GameBoard[i-1][c]
r = (i-1)
i = MAX_COL;
checkHorizWin(int r, int c, char p) == true
yes
no
checkVertWin(int r, int c, char p) == true
yes
no
checkDiagWin(int r, int c, char p) == true
yes
no
Return False
Return True

checkIfFree



checkTie


checkTie();
for(int i = 0; i<=MAX_COL;i++)
whatsAtPos(MAX_ROW, i) == " "
no
yes
Return True
Return False