MATTHEW FRANCHI

SOFTWARE DEVELOPMENT FOUNDATIONS

APRIL 7, 2020

ConnectX Project Requirements Report

CONTENTS

PROJECT OVERVIEW

FUNCTIONAL REQUIREMENTS

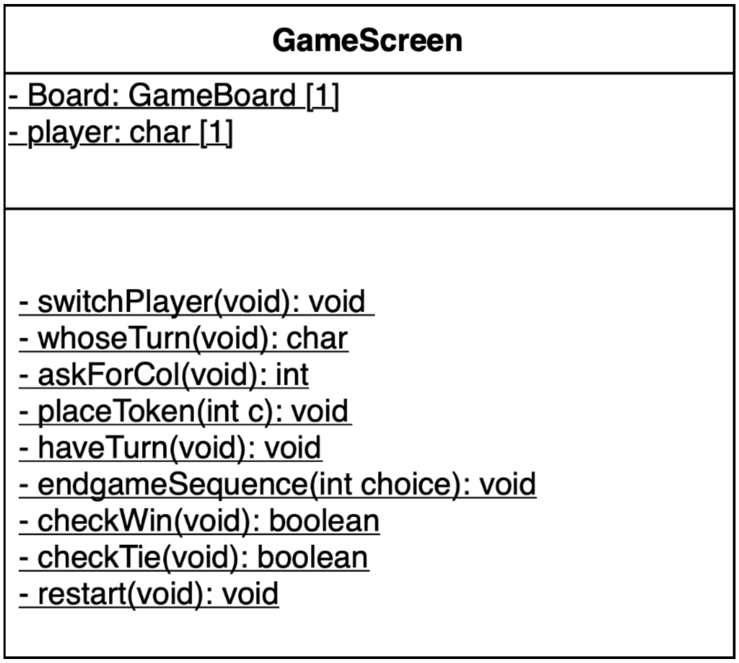
I. As a user, I can place my token in any available column, so that I can play the game
II. As a user, I can set the number of rows on the board, so that I have the ability to customize my game
III. As a user, I can set the number of columns on the board, so that I have the ability to customize my game
IV. As a user, I can set the (number to win) tokens, so that I have the ability to customize my game
V. As a user, I can set the number of players, so that I have the ability to customize my game
VI. As a user, I can set the token for each player, so that I know who is who
VII. As a user, I can place (number to win) tokens in vertical order
VIII. As a user, I can place (number to win) tokens in horizontal order
IX. As a user, I can place (number to win) tokens in diagonal order
X. As a user, I can see whose turn it is before each play, so that I know when it is and isn't my turn to play.
XI. As a user, I can see the current state of the board after each play, so that I can plan my next move.
XII. As a user, I can start a new game after the current one ends, so that I can have another chance to play from the beginning.
XIII. As a user, I can try to place four of my tokens in the same row, column, or diagonal order consecutively, so that I can win the game.
XIV. As a user, I can see if the game ended in a win
XV. As a user, I can see if the game ended in a tie
XVI. As a user, I can only enter my tokens in columns with space available, so that I can follow the rules of the game.
XVII. As a user, I can only enter tokens in the first available row in a column, so that I do not overwrite existing tokens.
XVIII. As a user, if I am the first one to specify a player token, then I will go first.
XIX. As a user, if someone wins the game, I can see the winning board.
XX. As a user, I can choose to not play a new game
XXI. As a user, I can choose to play a new game
XXII. As a user, I can choose the fast implementation / mode
XXIII. As a user, I can choose the memory-efficient implementation / mode
XXIV. As a user, I can be prompted to enter a new column if my first choice is invalid
XXV. As a user, I can be prompted to enter a new player token if my first choice was already taken
XXVI. As a user, I can be prompted to enter a new number of rows if my first input was out of range
XXVII. As a user, I can be prompted to enter a new number of columns if my first input was out of range

NON-FUNCTIONAL REQUIREMENTS

I. Must be implemented with the Java coding language
II. Must be able to run on the Clemson School of Computing Unix Environment
III. Must be able to run on a command-line interface
IV. Must be completely reliable; no crashes mid-game, when starting a new game, etc.
V. There should be a minimal, unnoticeable processing time between each turn
VI. The GameBoard and BoardPosition classes must follow the exact method signatures specified in the project guidelines document.
VII. The GameScreen class is the only class that can get input from the user or print to the console.
VIII. The project should have a high degree of adaptability and modularity, so that future additions are less complicated and easier.
IX. The project should keep the contents of the board private, as to avoid tampering.
X. The game should be extremely easy to play, and straightforward – in other words, someone with no prior experience with Connect4 (X) should be able to play the game.
XI. The project should be compiled using a makefile.
XII. Any prompts for user input should be clear and easy to understand
XIII. The game board must be an upright grid
XIV. All code must follow all best practices discussed in class
XV. All function signatures specified in the requirements document should be followed exactly
XVI. The number of rows on the board is greater than 3
XVII. The number of rows on the board is less than 100
XVIII. The number of columns on the board is greater than 3
XIX. The number of columns on the board is less than 100
XX. The number of tokens needed to win is greater than or equal to 3
XXI. The number of tokens needed to win is less than or equal to 25
XXII. The number of players is greater than or equal to 2
XXIII. The number of players is less than or equal to 10
XXIV. ConnectX should have a fast implementation
XXV. ConnectX should have a memory-efficient implementation
XXVI. The game should work with 2 to 10 players
XXVII. The program should not have any magic numbers
XXVIII. The memory-efficient (Map) implementation should not create keys for the blank space [‘ ‘]
XXIX. The game should not allow for a number of tokens needed to win greater than the number of rows
XXX. The game should not allow for a number of tokens needed to win greater than the number of columns
XXXI. The program should have descriptive comments
XXXII. The program should follow the principles of design by contract, utilizing Javadoc comments

GAMESCREEN CLASS

UML CLASS DIAGRAM



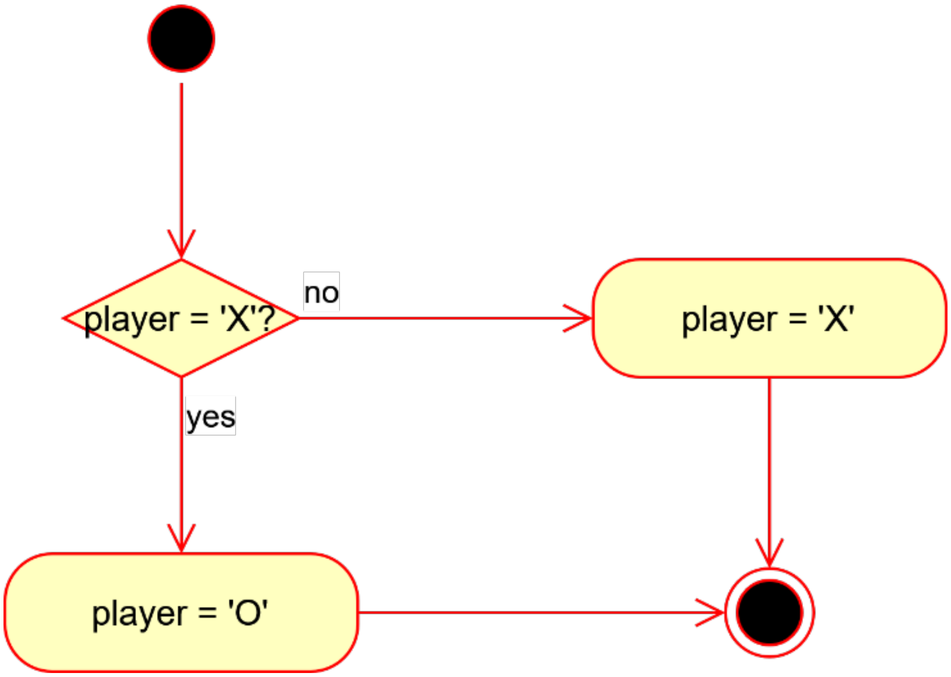
GameScreen

- Board: GameBoard [1]
- player: char [1]

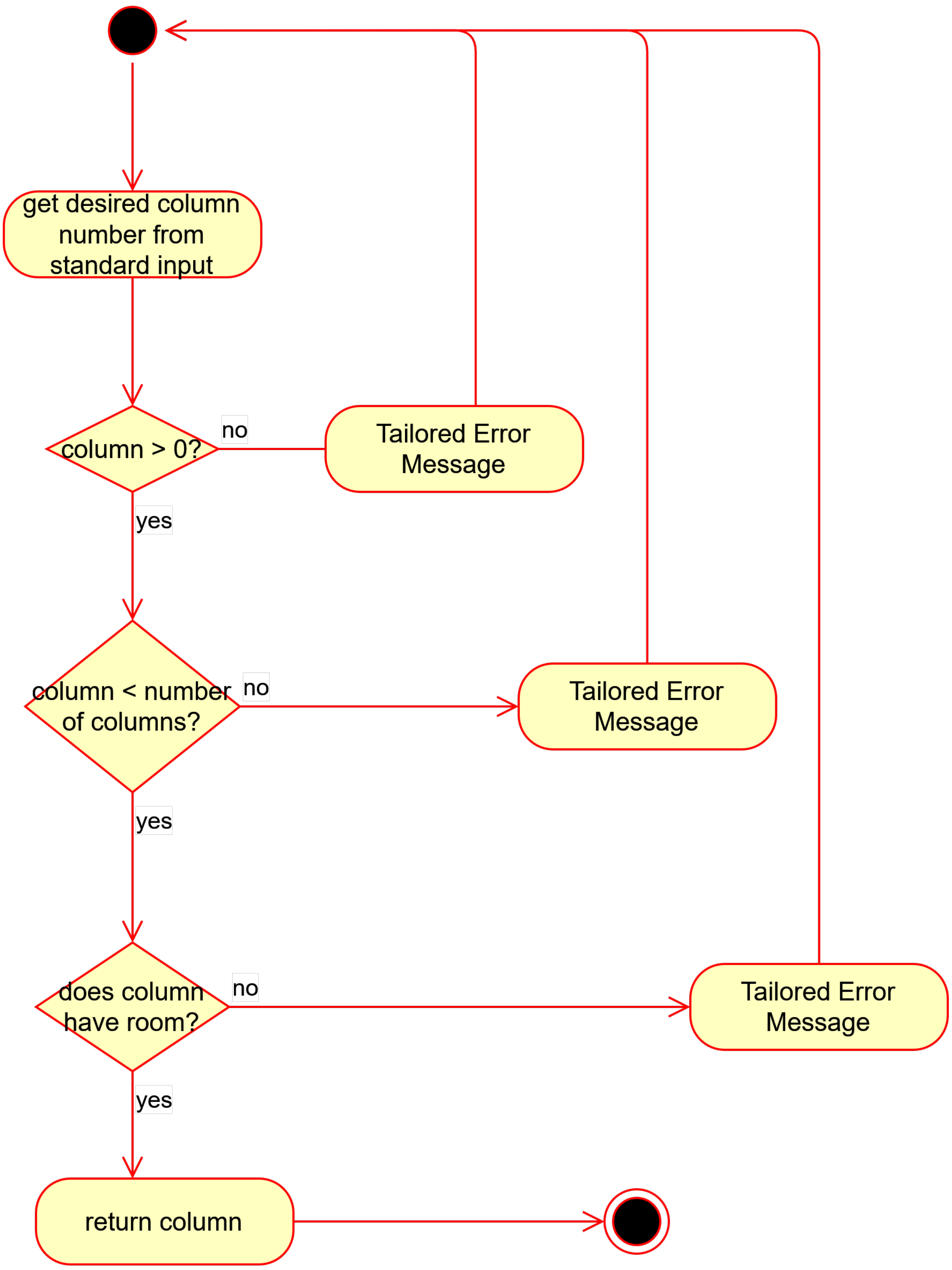
- switchPlayer(void): void
- whoseTurn(void): char
- askForCol(void): int
- placeToken(int c): void
- haveTurn(void): void
- endgameSequence(int choice): void
- checkWin(void): boolean
- checkTie(void): boolean
- restart(void): void

UML ACTIVITY DIAGRAMS

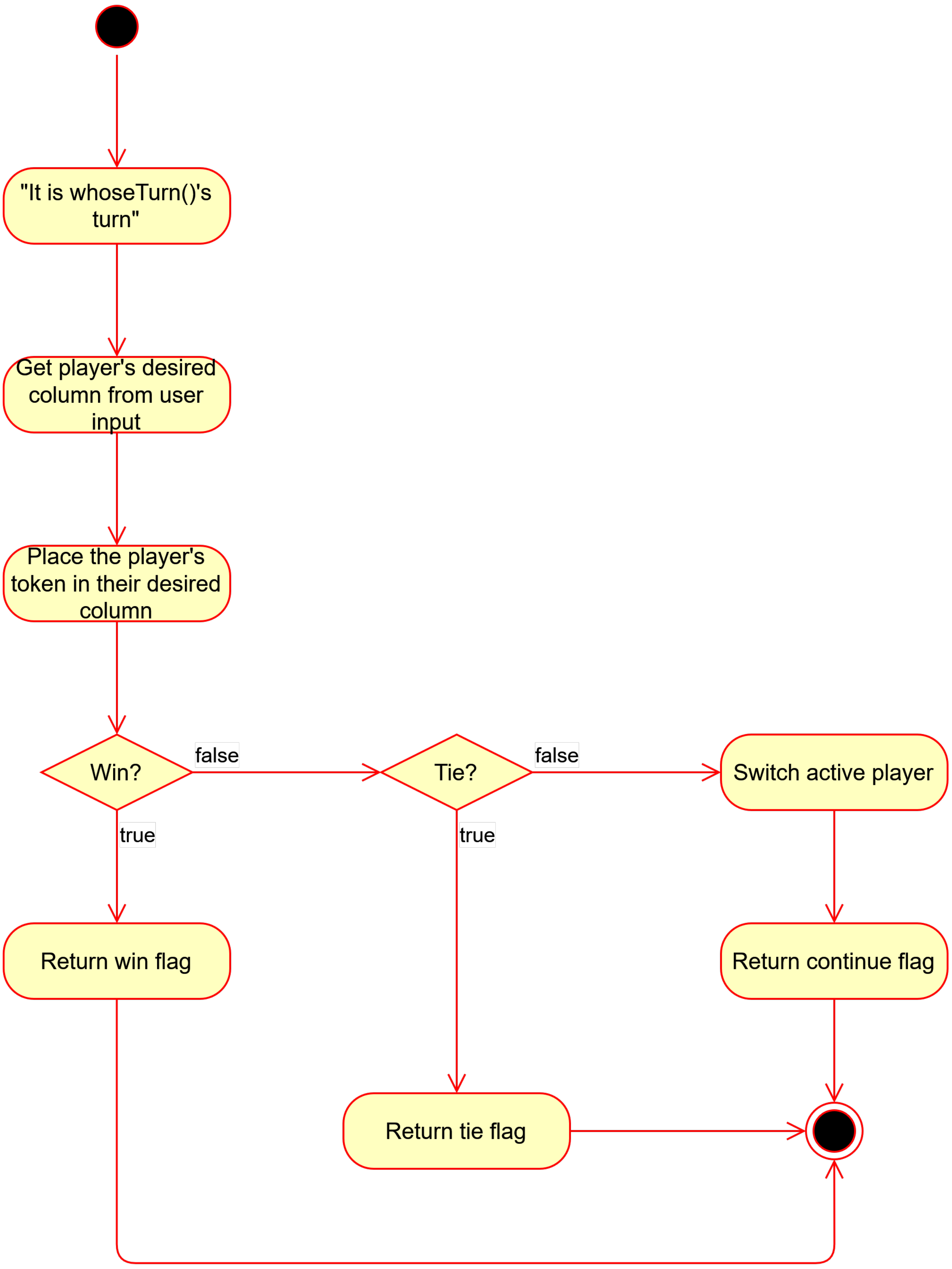
SWITCHPLAYER



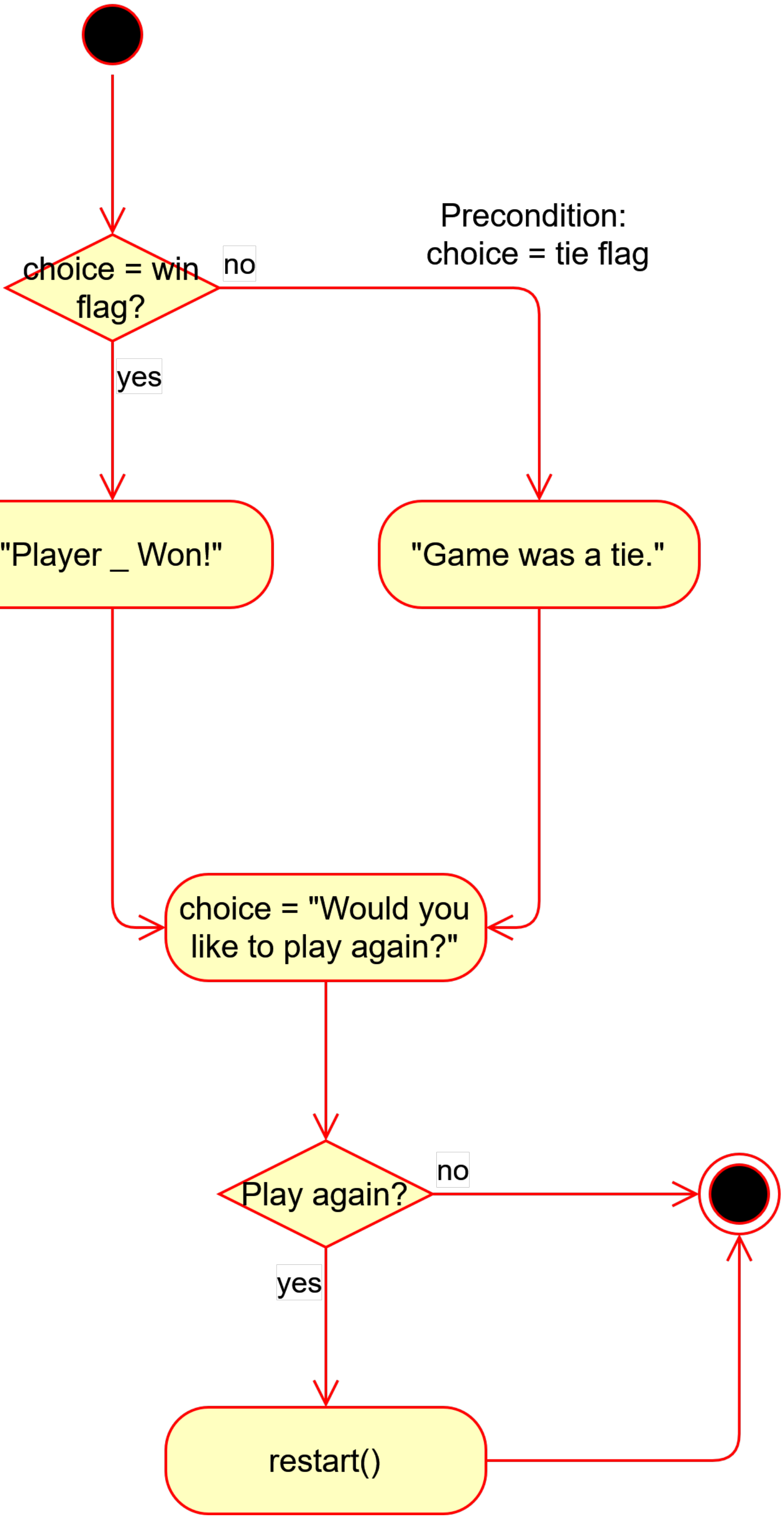
ASKFORCOL


get desired column number from standard input
column > 0?
no
Tailored Error Message
yes
column < number of columns?
no
Tailored Error Message
yes
does column have room?
no
Tailored Error Message
yes
return column

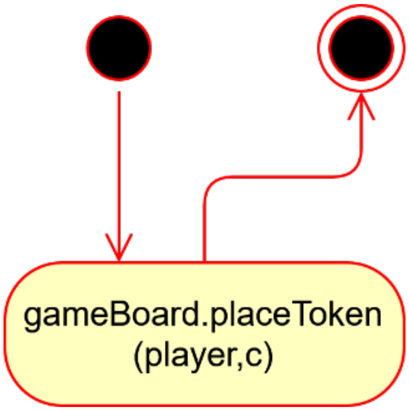
HAVETURN


"It is whoseTurn()'s turn"
Get player's desired column from user input
Place the player's token in their desired column
Win?
false
true
Tie?
false
true
Switch active player
Return win flag
Return continue flag
Return tie flag

ENDGAMESEQUENCE

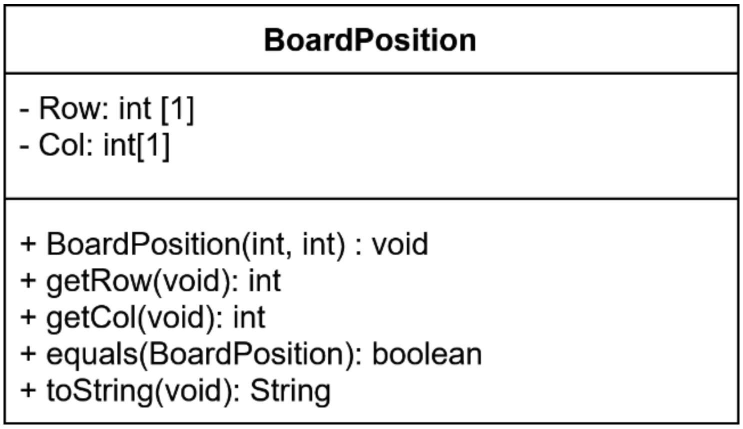


PLACETOKEN



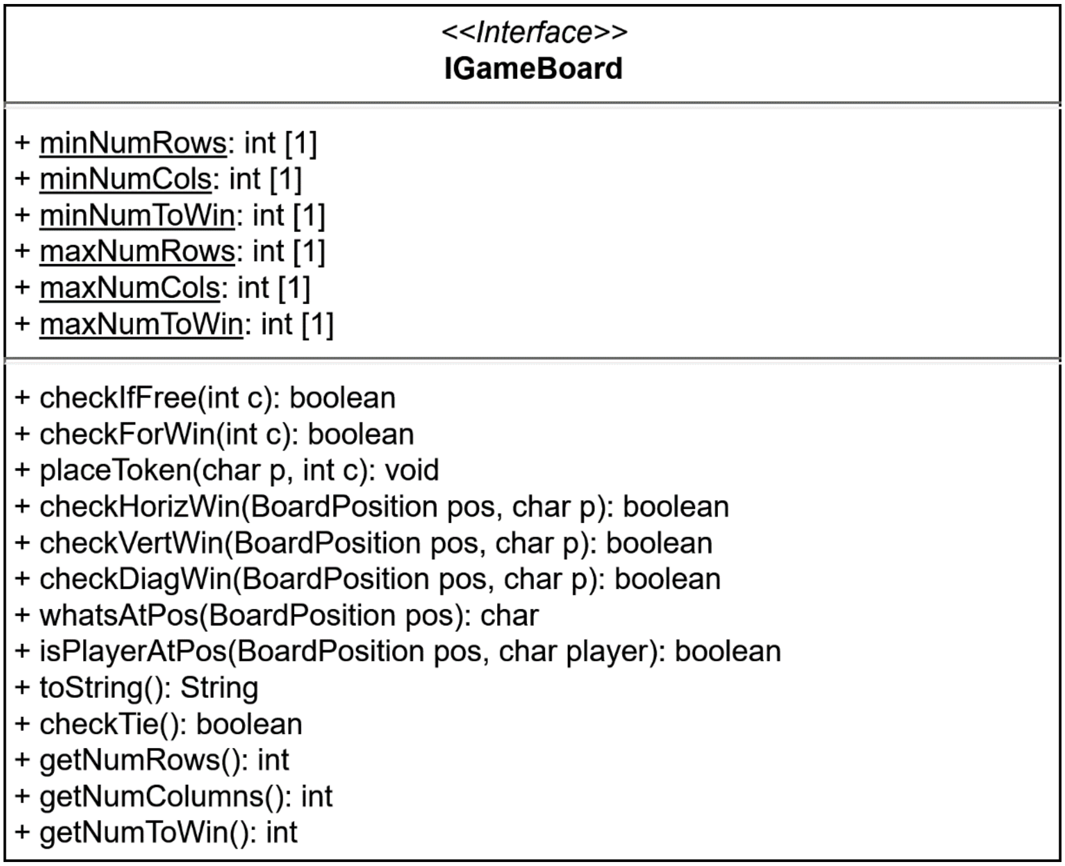
BOARDPOSITION CLASS

UML CLASS DIAGRAM



IGAMEBOARD INTERFACE

UML INTERFACE DIAGRAM



<<Interface>>

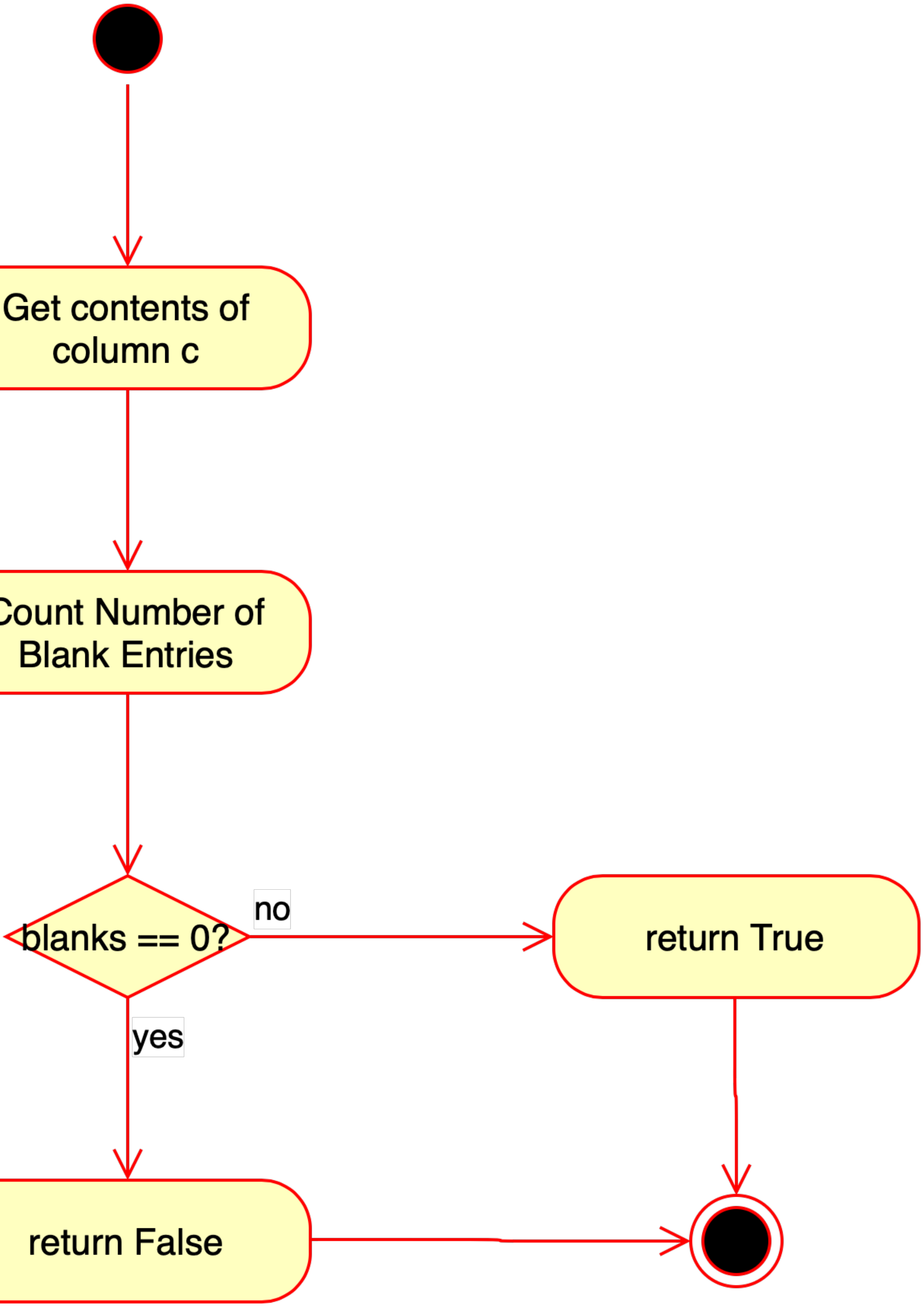
IGameBoard

+ minNumRows: int [1]
+ minNumCols: int [1]
+ minNumToWin: int [1]
+ maxNumRows: int [1]
+ maxNumCols: int [1]
+ maxNumToWin: int [1]

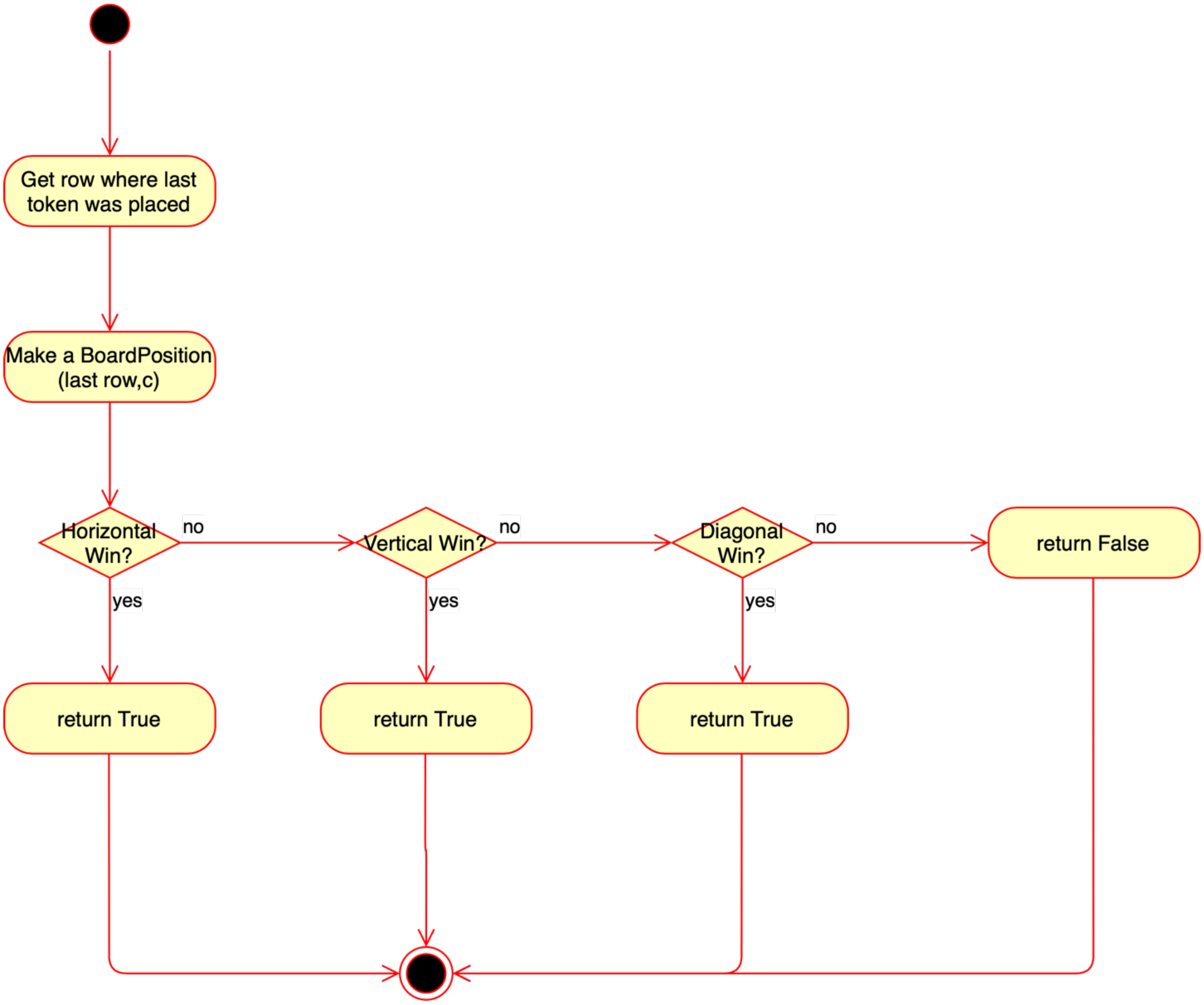
+ checkIfFree(int c): boolean
+ checkForWin(int c): boolean
+ placeToken(char p, int c): void
+ checkHorizWin(BoardPosition pos, char p): boolean
+ checkVertWin(BoardPosition pos, char p): boolean
+ checkDiagWin(BoardPosition pos, char p): boolean
+ whatsAtPos(BoardPosition pos): char
+ isPlayerAtPos(BoardPosition pos, char player): boolean
+ toString(): String
+ checkTie(): boolean
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int

ACTIVITY DIAGRAMS

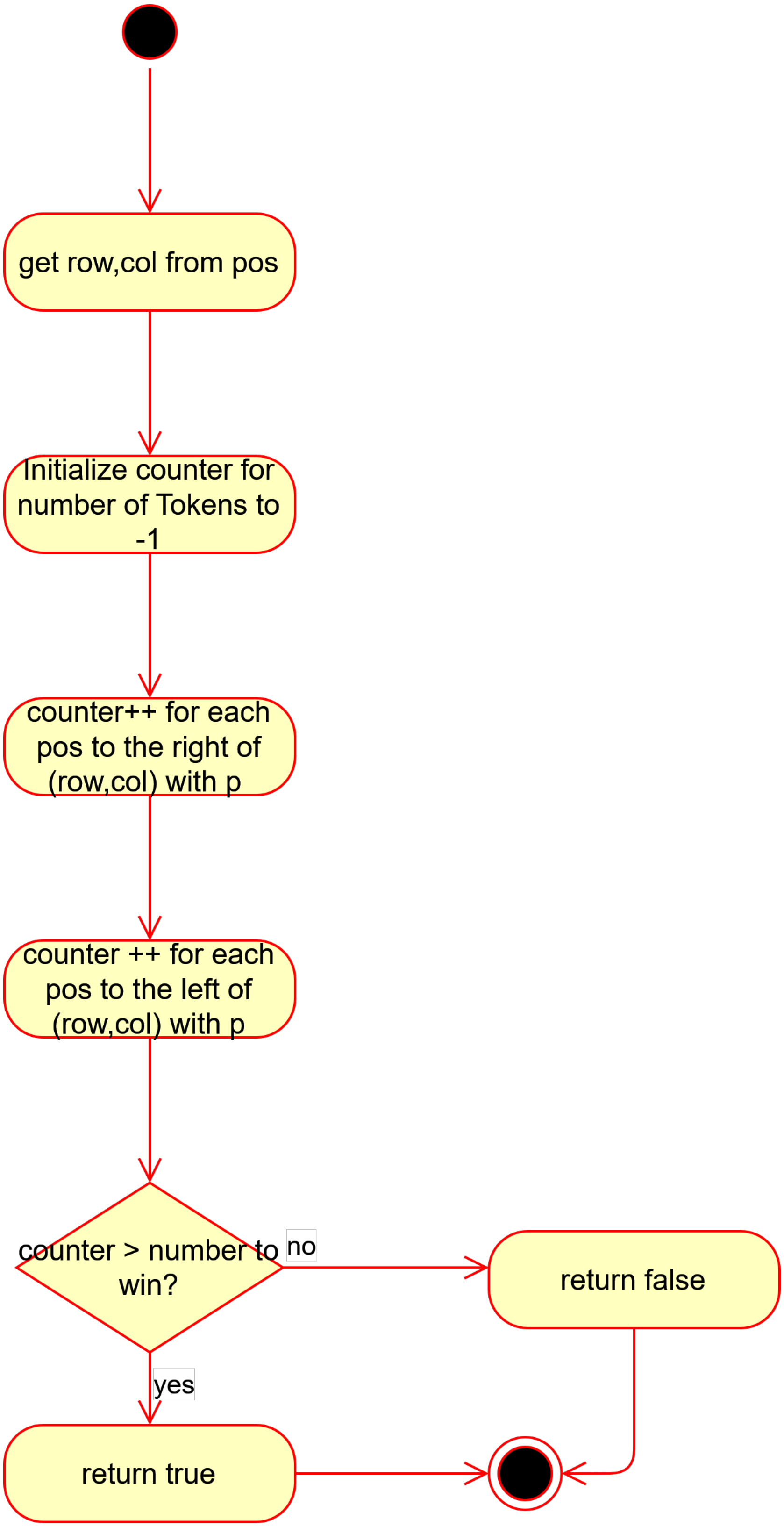
CHECKIFFREE



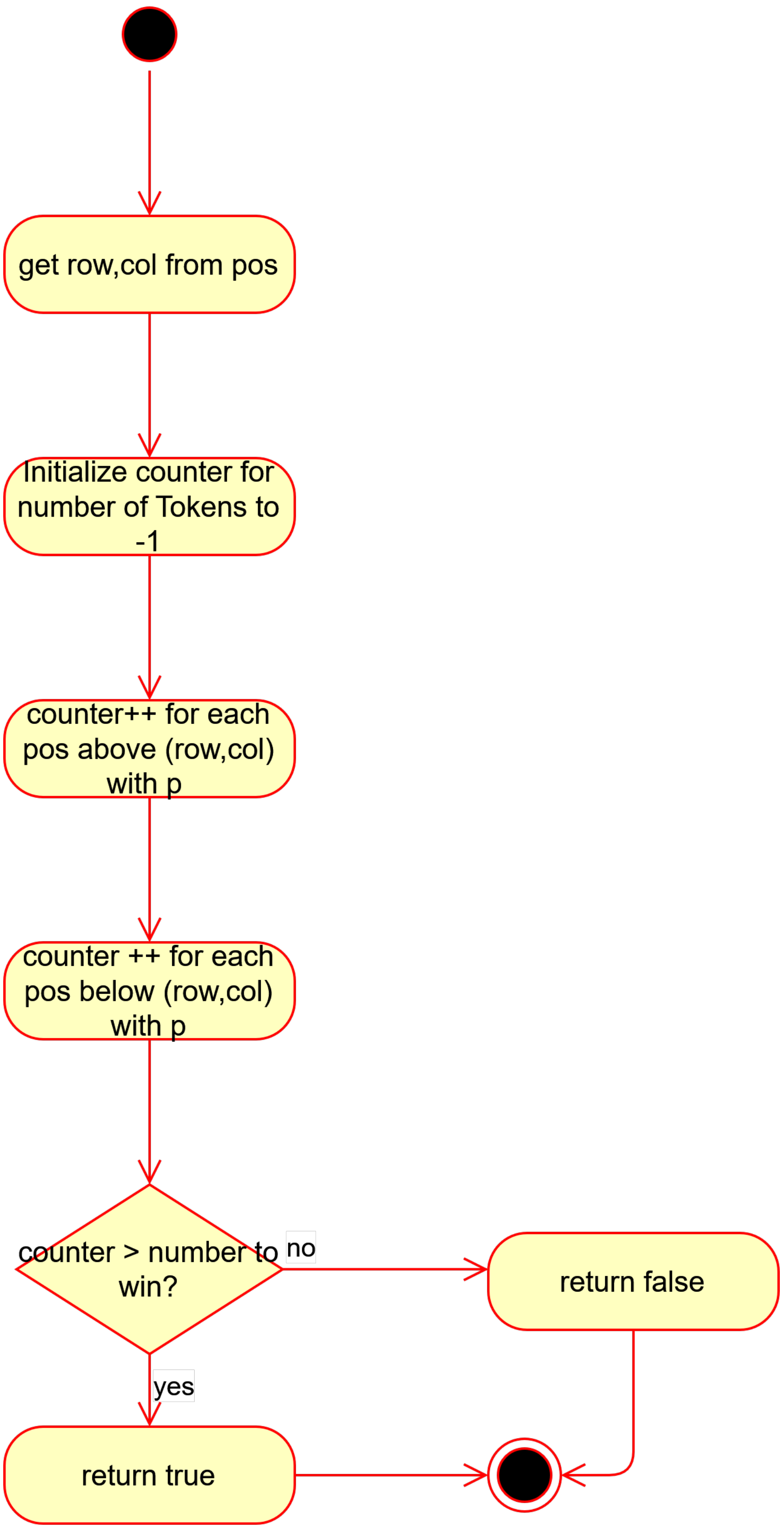
CHECKFORWIN



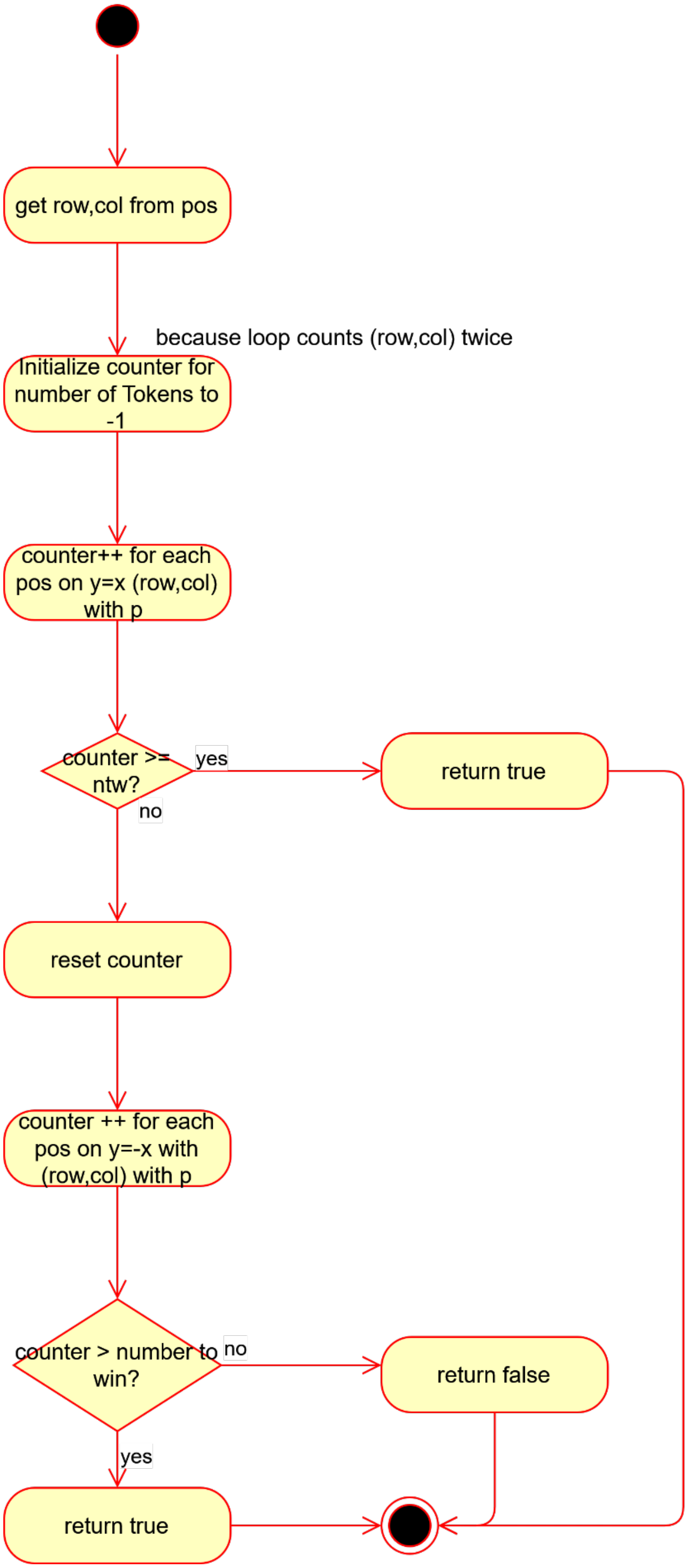
CHECKHORIZWIN


get row,col from pos
Initialize counter for number of Tokens to -1
counter++ for each pos to the right of (row,col) with p
counter ++ for each pos to the left of (row,col) with p
counter > number to win?
no
return false
yes
return true

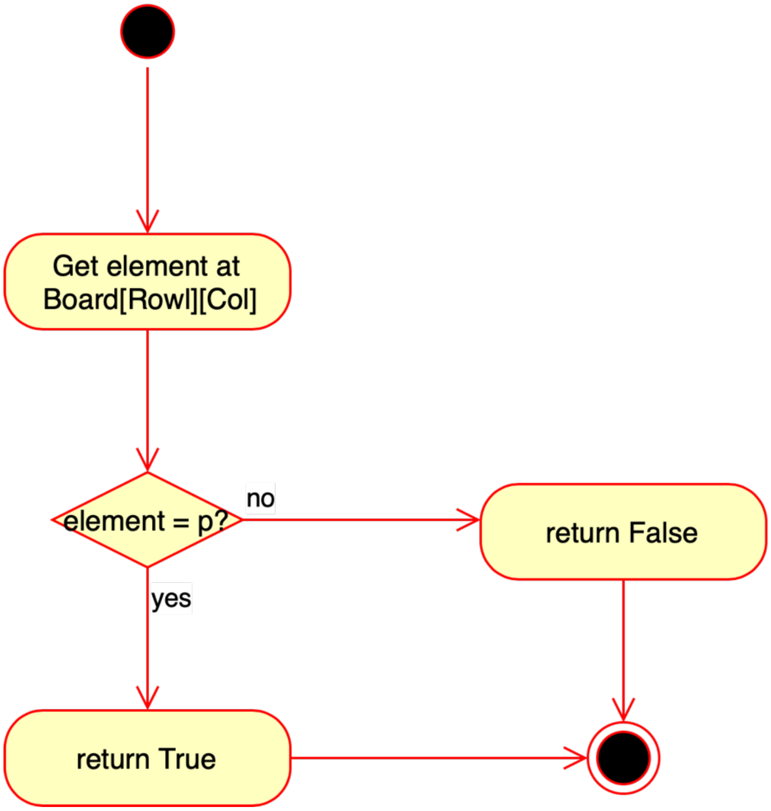
CHECKVERTWIN


get row,col from pos
Initialize counter for number of Tokens to -1
counter++ for each pos above (row,col) with p
counter ++ for each pos below (row,col) with p
counter > number to win?
no
return false
yes
return true

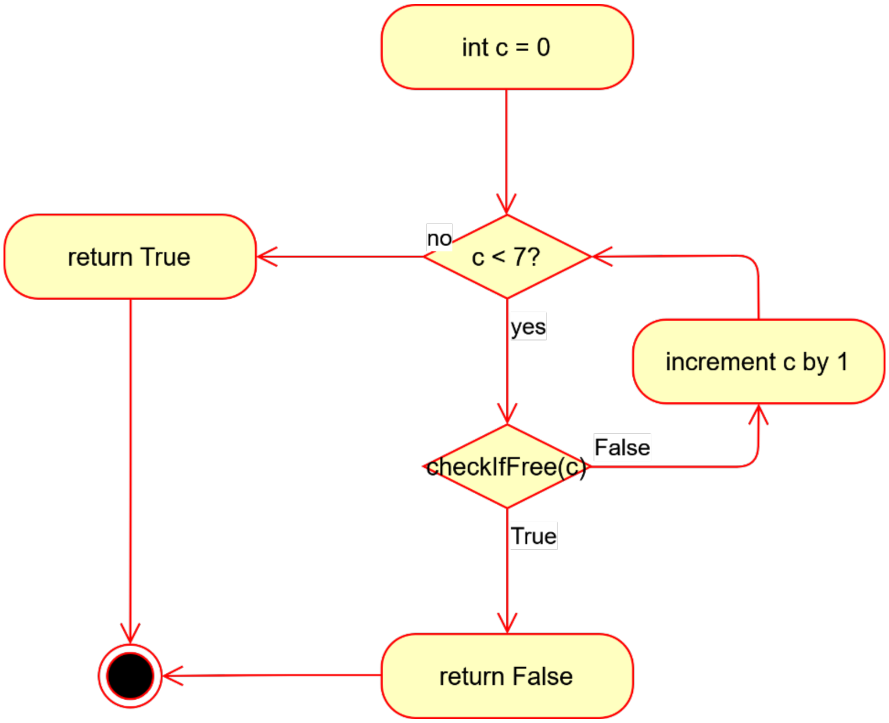
CHECKDIAGWIN


get row,col from pos
because loop counts (row,col) twice
Initialize counter for number of Tokens to -1
counter++ for each pos on y=x (row,col) with p
counter >= ntw?
yes
no
return true
reset counter
counter ++ for each pos on y=-x with (row,col) with p
counter > number to win?
no
yes
return false
return true

ISPLAYERATPOS

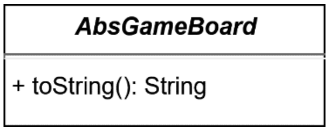


CHECKTIE



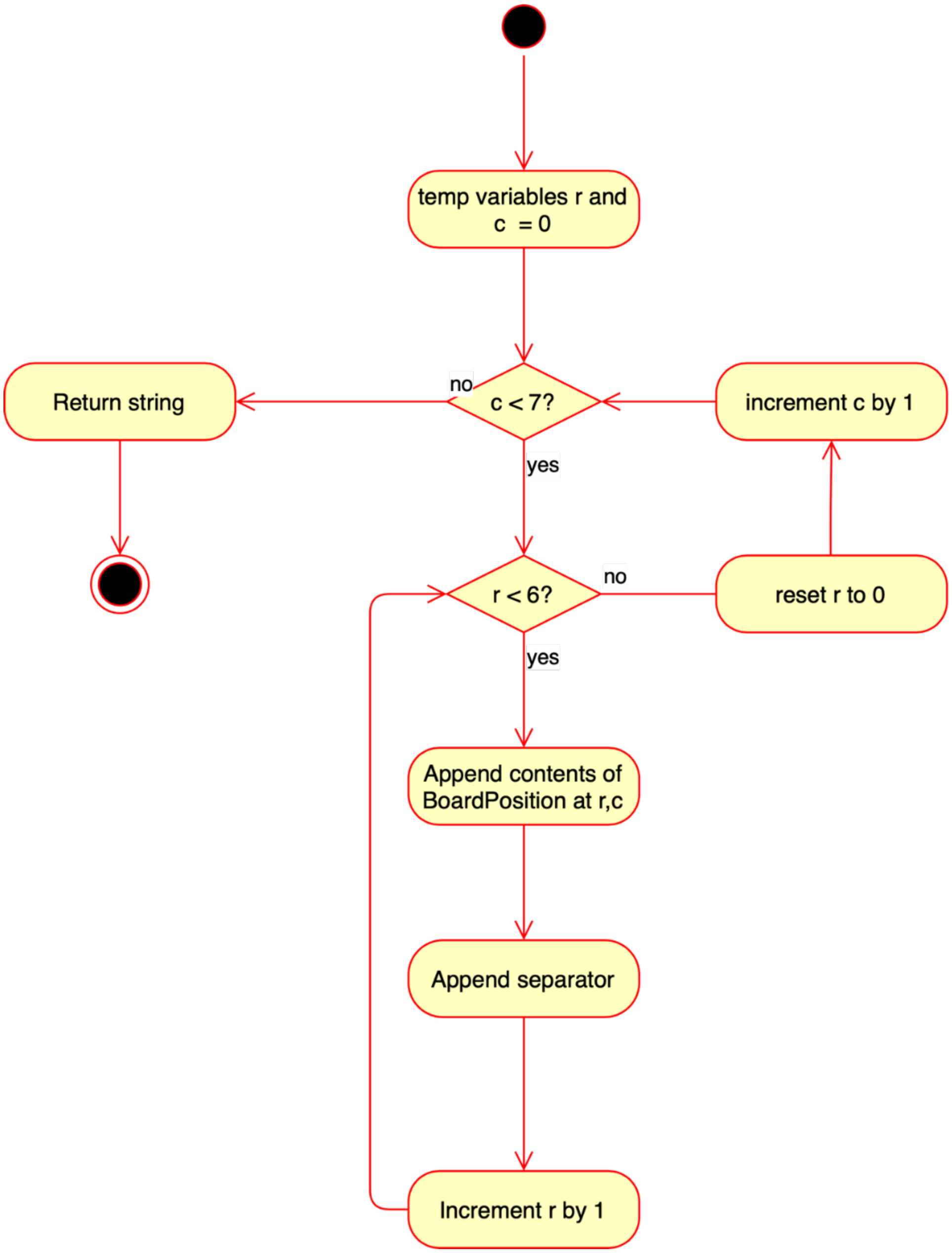
ABSGAMEBOARD CLASS

UML CLASS DIAGRAM



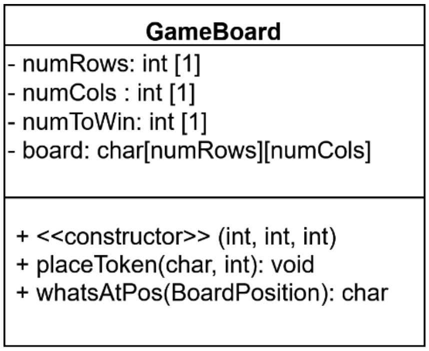
UML ACTIVITY DIAGRAM

TOSTRING



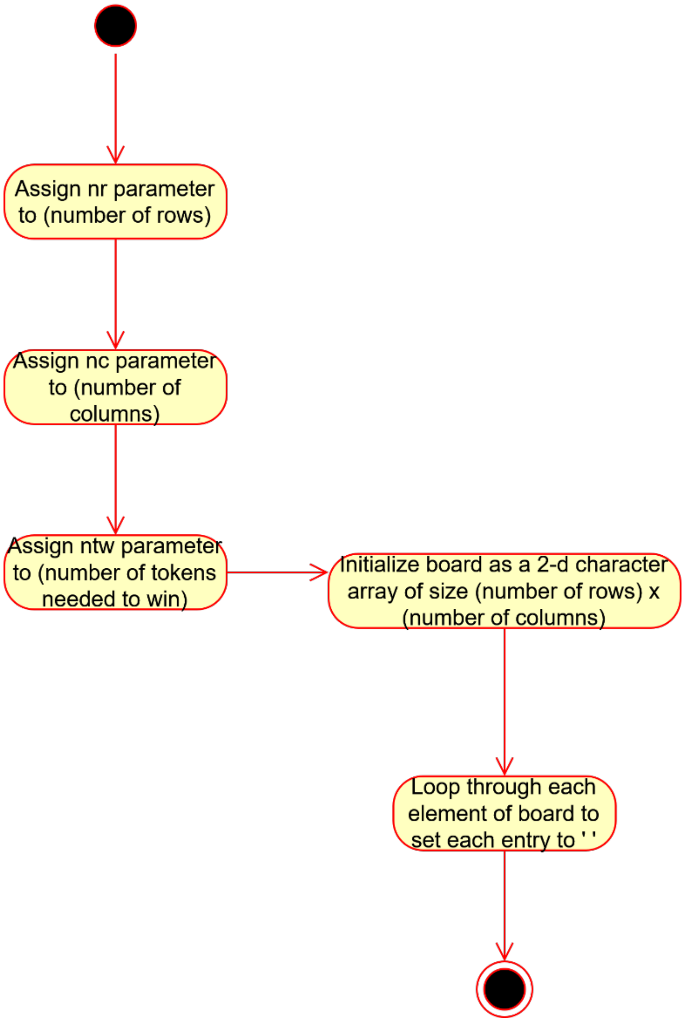
GAMEBOARD CLASS

UML CLASS DIAGRAM

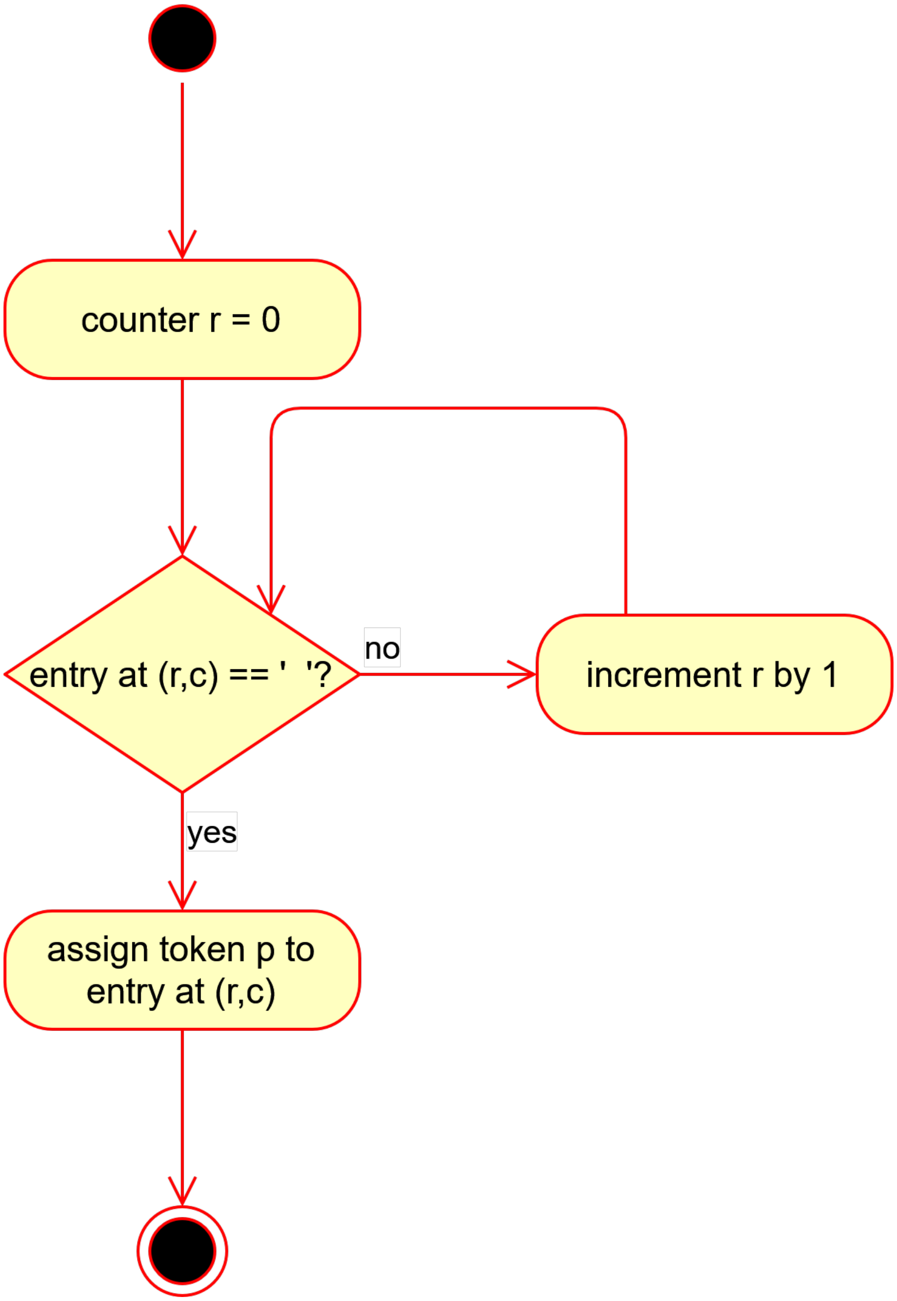

GameBoard
- numRows: int [1]
- numCols : int [1]
- numToWin: int [1]
- board: char[numRows][numCols]
+ <<constructor>> (int, int, int)
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char

UML ACTIVITY DIAGRAMS

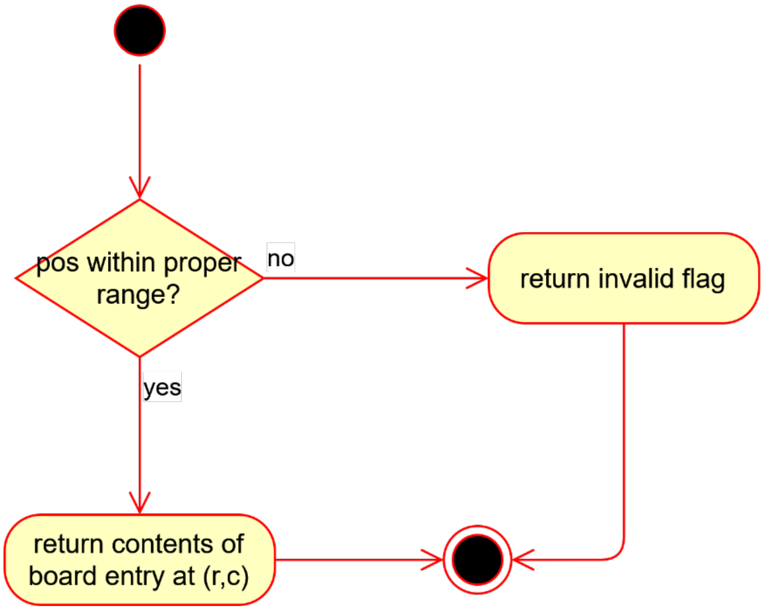
CONSTRUCTOR


Assign nr parameter to (number of rows)
Assign nc parameter to (number of columns)
Assign ntw parameter to (number of tokens needed to win)
Initialize board as a 2-d character array of size (number of rows) x (number of columns)
Loop through each element of board to set each entry to ' '

PLACETOKEN

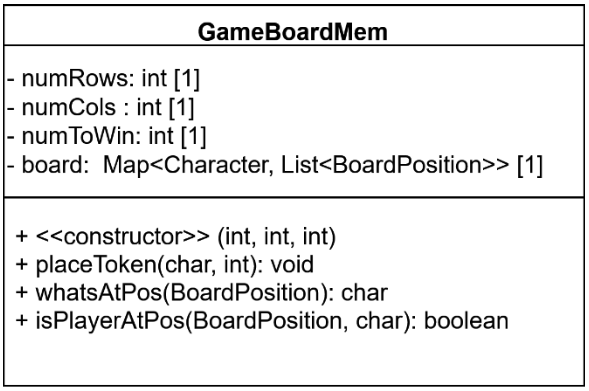

counter r = 0
entry at (r,c) == ' '?
no
increment r by 1
yes
assign token p to entry at (r,c)

WHATSATPOS



GAMEBOARDMEM CLASS

UML CLASS DIAGRAM



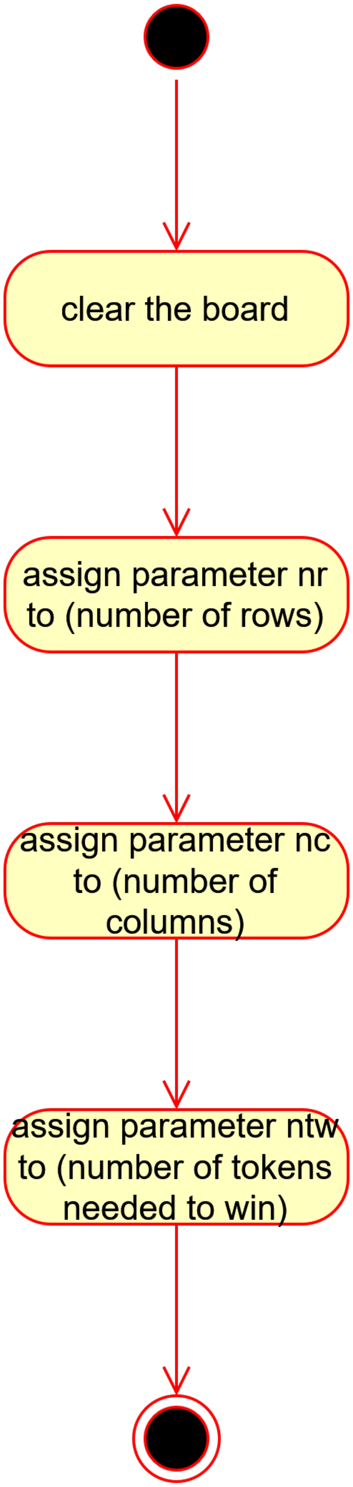
GameBoardMem

- numRows: int [1]
- numCols : int [1]
- numToWin: int [1]
- board: Map<Character, List<BoardPosition>> [1]

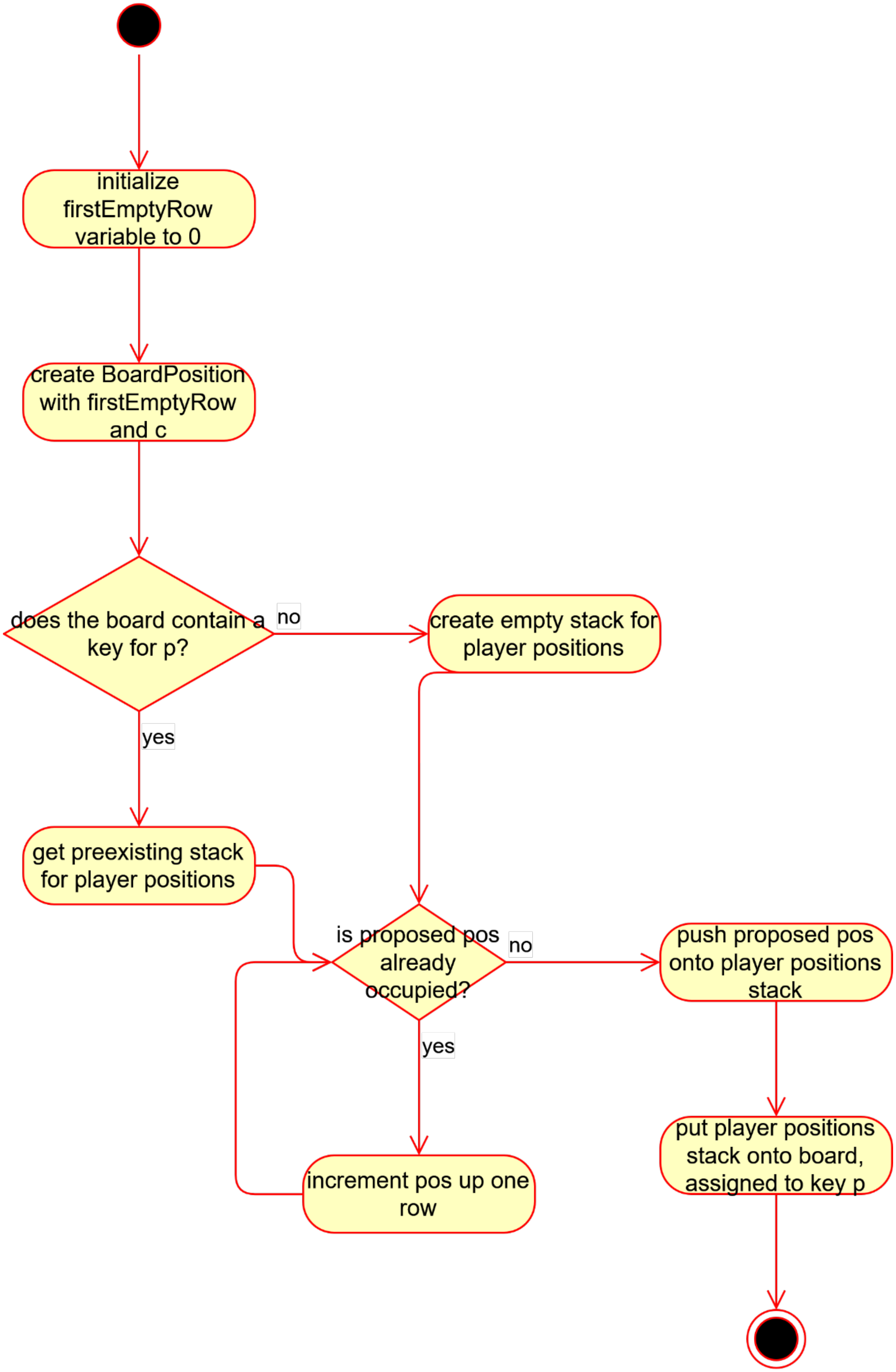
+ <<constructor>> (int, int, int)
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean

UML ACTIVITY DIAGRAMS

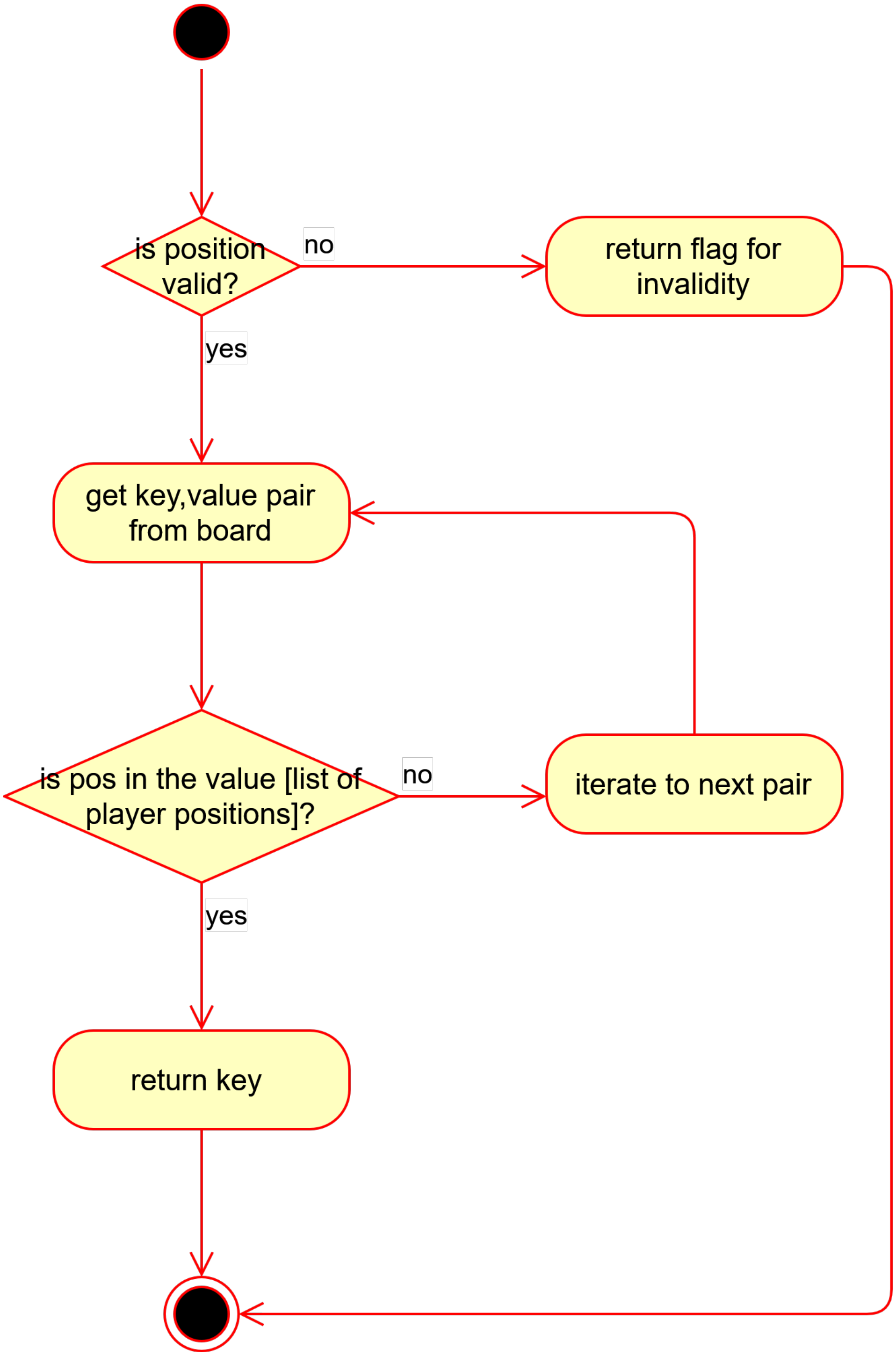
CONSTRUCTOR


clear the board
assign parameter nr to (number of rows)
assign parameter nc to (number of columns)
assign parameter ntw to (number of tokens needed to win)

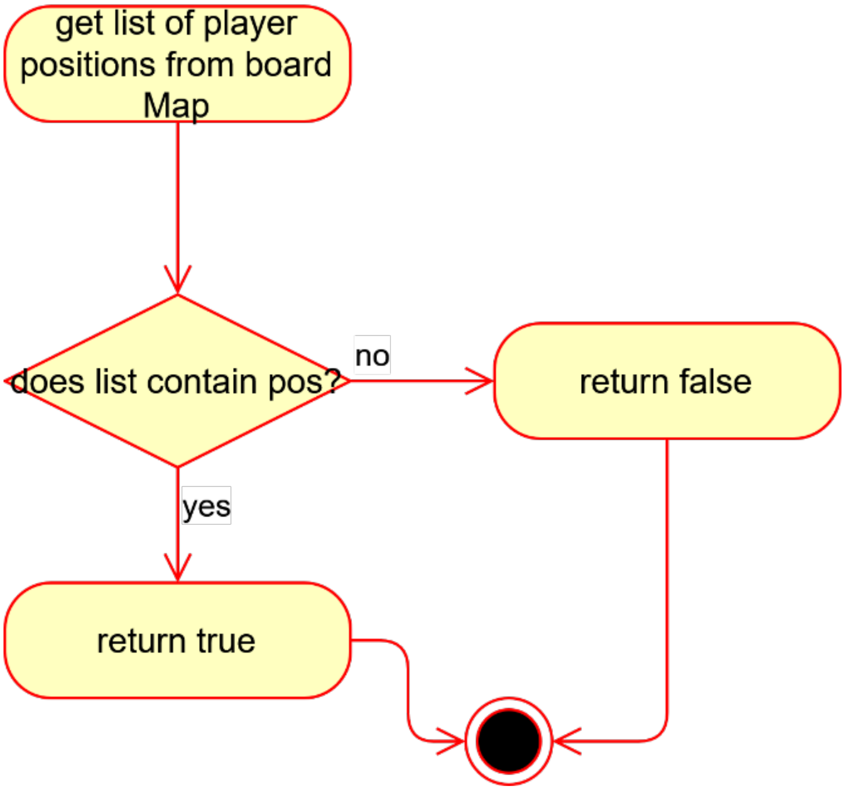
PLACETOKEN


initialize firstEmptyRow variable to 0
create BoardPosition with firstEmptyRow and c
does the board contain a key for p?
no
create empty stack for player positions
yes
get preexisting stack for player positions
is proposed pos already occupied?
no
push proposed pos onto player positions stack
yes
increment pos up one row
put player positions stack onto board, assigned to key p

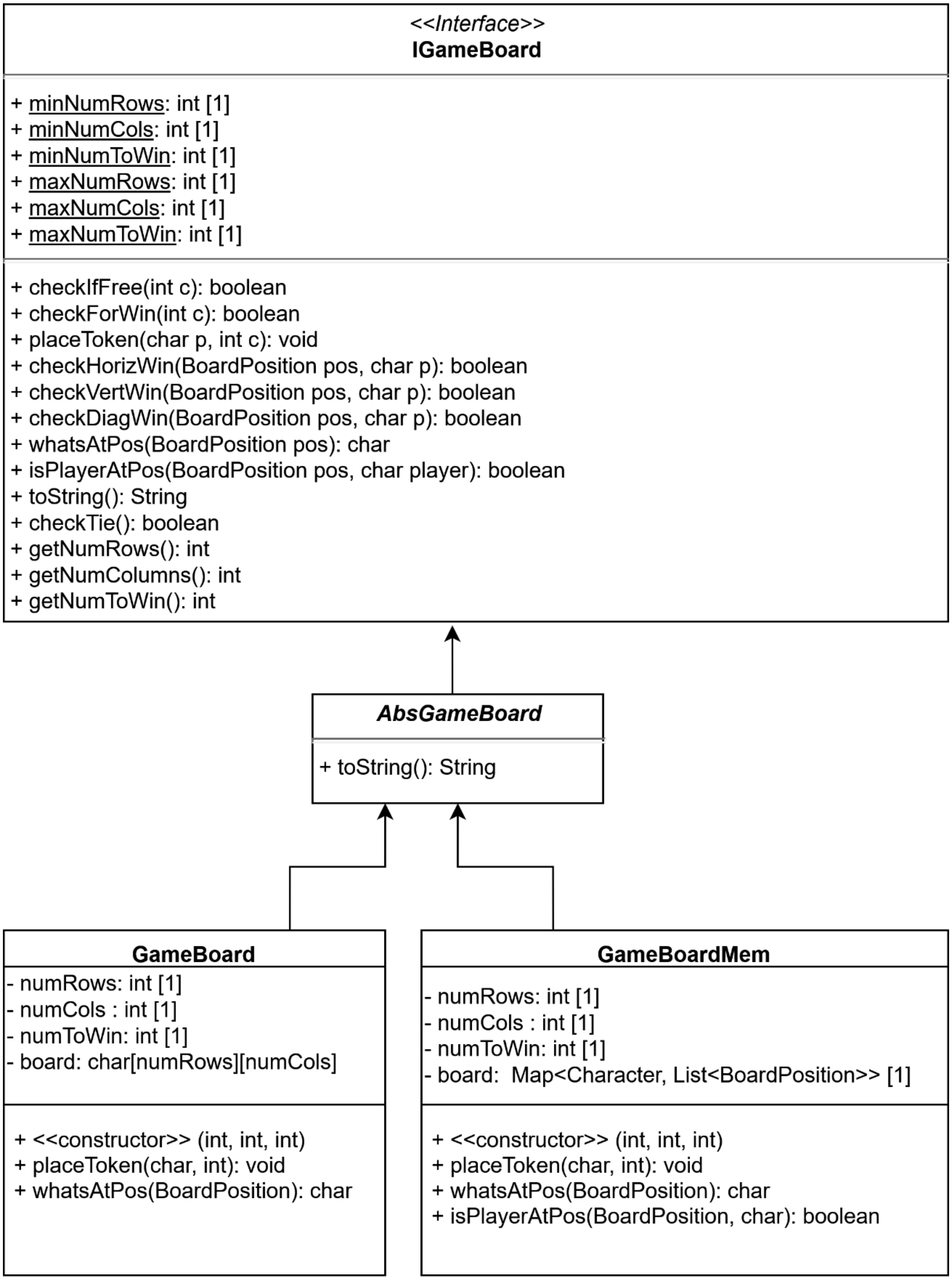
WHATSATPOS


is position valid?
no
return flag for invalidity
yes
get key,value pair from board
is pos in the value [list of player positions]?
no
iterate to next pair
yes
return key

ISPLAYERATPOS



UML CLASS-RELATIONSHIPS DIAGRAM



<<Interface>>
IGameBoard

+ minNumRows: int [1]
+ minNumCols: int [1]
+ minNumToWin: int [1]
+ maxNumRows: int [1]
+ maxNumCols: int [1]
+ maxNumToWin: int [1]

+ checkIfFree(int c): boolean
+ checkForWin(int c): boolean
+ placeToken(char p, int c): void
+ checkHorizWin(BoardPosition pos, char p): boolean
+ checkVertWin(BoardPosition pos, char p): boolean
+ checkDiagWin(BoardPosition pos, char p): boolean
+ whatsAtPos(BoardPosition pos): char
+ isPlayerAtPos(BoardPosition pos, char player): boolean
+ toString(): String
+ checkTie(): boolean
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int

AbsGameBoard

+ toString(): String

GameBoard

- numRows: int [1]
- numCols : int [1]
- numToWin: int [1]
- board: char[numRows][numCols]

+ <<constructor>> (int, int, int)
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char

GameBoardMem

- numRows: int [1]
- numCols : int [1]
- numToWin: int [1]
- board: Map<Character, List<BoardPosition>> [1]

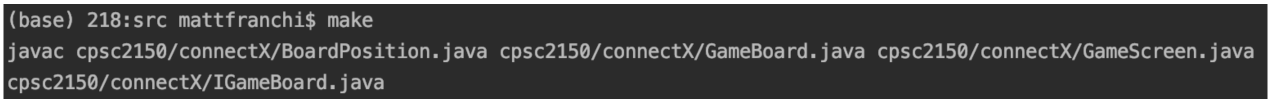
+ <<constructor>> (int, int, int)
+ placeToken(char, int): void
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean

PROJECT COMPILING INSTRUCTIONS

ConnectX comes bundled with a GNU makefile that provides the following functionalities:

MAKE DEFAULT

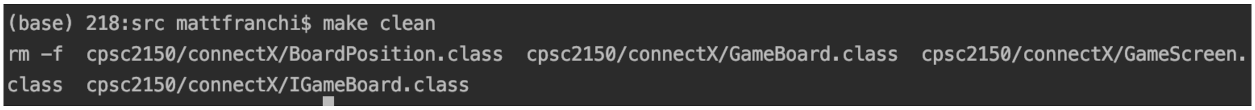
The default routine compiles all the project's .java files into .class files.



```
(base) 218:src mattfranchi$ make
javac cpsc2150/connectX/BoardPosition.java cpsc2150/connectX/GameBoard.java cpsc2150/connectX/GameScreen.java
cpsc2150/connectX/IGameBoard.java
```

MAKE RUN

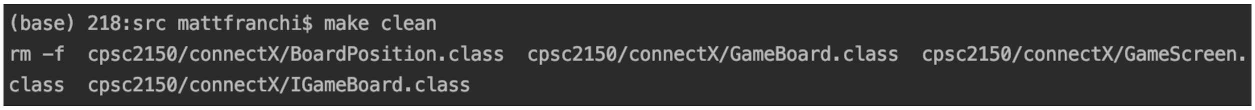
The run command executes the project's GameScreen class, which starts the ConnectX game. NOTE: the *default make* command needs to be run before *make run*.



```
(base) 218:src mattfranchi$ make clean
rm -f  cpsc2150/connectX/BoardPosition.class  cpsc2150/connectX/GameBoard.class  cpsc2150/connectX/GameScreen.
class  cpsc2150/connectX/IGameBoard.class
```

MAKE CLEAN

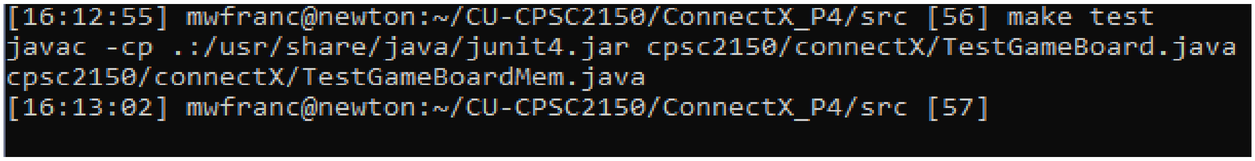
The clean command deletes all .class files in the project directory; NOTE: the code will have to be recompiled with the *make* command following the execution of this command.



```
(base) 218:src mattfranchi$ make clean
rm -f  cpsc2150/connectX/BoardPosition.class  cpsc2150/connectX/GameBoard.class  cpsc2150/connectX/GameScreen.
class  cpsc2150/connectX/IGameBoard.class
```

MAKE TEST

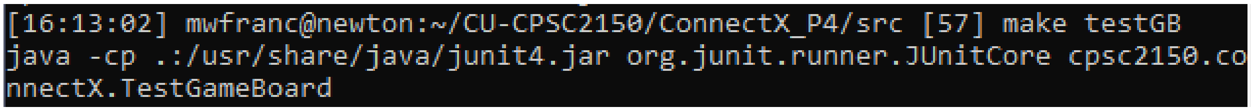
The test command compiles all of ConnectX's testing .java files into .class files.



```
[16:12:55] mwfranc@newton:~/CU-CPSC2150/ConnectX_P4/src [56] make test
javac -cp .:/usr/share/java/junit4.jar cpsc2150/connectX/TestGameBoard.java
cpsc2150/connectX/TestGameBoardMem.java
[16:13:02] mwfranc@newton:~/CU-CPSC2150/ConnectX_P4/src [57]
```

MAKE TESTGB

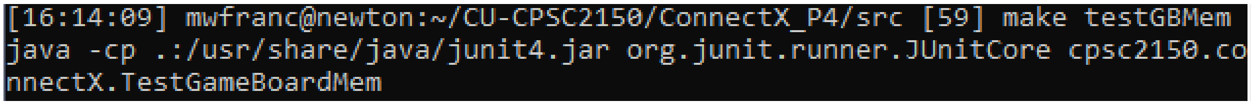
The testGB command runs the junit tests contained within the TestGameBoard.class file.



```
[16:13:02] mwfranc@newton:~/CU-CPSC2150/ConnectX_P4/src [57] make testGB
java -cp .:/usr/share/java/junit4.jar org.junit.runner.JUnitCore cpsc2150.co
nnectX.TestGameBoard
```

MAKE TESTGBMEM

The testGBMem command runs the junit tests contained within the TestGameBoardMem.class file.



```
[16:14:09] mwfranc@newton:~/CU-CPSC2150/ConnectX_P4/src [59] make testGBMem
java -cp .:/usr/share/java/junit4.jar org.junit.runner.JUnitCore cpsc2150.co
nnectX.TestGameBoardMem
```

TESTING

public GameBoard(int nr, int nc, int ntw)

<table>
<tr>
<td><b><i>Input</i></b>

nr = minNumRows [3]
nc = minNumCols [3]
ntw = minNumToWin [3]</td>
<td><b><i>Output</i></b>

3x3 GameBoard<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></td>
<td><b><i>Reason:</i></b>
This test case is unique and distinct because it is a boundary case; it evaluates whether or not the constructor can produce the smallest allowed game board.</td>
</tr>
<tr>
<td><b><i>Input</i></b>

nr = maxNumRows [100]
nc = maxNumCols [100]
ntw = maxNumToWin [25]</td>
<td><b><i>Output</i></b>

100x100 GameBoard<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></td>
<td><b><i>Reason:</i></b>
This test case is unique and distinct because it is a boundary case; it evaluates whether or not the constructor can produce the largest allowed game board.</td>
</tr>
<tr>
<td><b><i>Input</i></b>

nr = 6
nc = 7
ntw = 4</td>
<td><b><i>Output</i></b>

6x7 GameBoard<table><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td><b><i>Reason:</i></b>
This test case is unique and distinct because it is a routine case; it tests whether or not the constructor can produce a common-sized game board</td>
</tr>
</table>

public boolean checkIfFree(int c)

<table>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
<table>
<tr><td>R = 5</td><td>C = 0</td><td>P = X</td></tr>
</table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
checkIfFree(C) returns <b><i>false</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, because it tests whether or not checkIfFree() can correctly process a full column.</td>
</tr>
<tr>
<td>Input
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
<table>
<tr><td>R = N/A</td><td>C = N/A</td><td>P = N/A</td></tr>
</table>
</td>
<td>Output
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
checkIfFree(C) returns <b><i>true</i></b>
</td>
<td>Reason:
This test cause is a unique and distinct boundary case, because it tests whether or not checkIfFree() can correctly process an empty column.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td></tr>
</table>
<table>
<tr><td>R = 2</td><td>C = 2</td><td>P = X</td></tr>
</table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td></tr>
</table>
checkIfFree(C) returns <b><i>true</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct routine case, as it tests whether or not checkIfFree() can handle typical usage; the majority of the time, checkIfFree() will be given a column that is partially full.</td>
</tr>
</table>

public boolean checkHorizWin(int c)

<table>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td></td><td></td><td></td><td></td></tr>
</table>
<table><tr><td>R = 0</td><td>C = 3</td><td>P = X</td></tr></table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td><b>X</b></td><td></td><td></td><td></td></tr>
</table>
checkHorizWin(C,P) = <b><i>true</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct routine case, because it tests whether or not checkHorizWin() can correctly identify a typical horizontal win scenario.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td></td><td></td><td></td><td></td></tr>
</table>
<table><tr><td>R = 0</td><td>C = 3</td><td>P = O</td></tr></table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td><b>O</b></td><td></td><td></td><td></td></tr>
</table>
checkHorizWin(C,P) = <b><i>false</i></b>
</td>
<td><b><i>Reason:</i></b>
This test cause is a unique and distinct routine case, because it tests whether or not checkHorizWin() can correctly identify a typical scenario where no horizontal win will occur.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td>X</td><td></td><td>X</td><td></td><td></td></tr>
</table>
<table><tr><td>R = 0</td><td>C = 3</td><td>P = X</td></tr></table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td>X</td><td><b>X</b></td><td>X</td><td></td><td></td></tr>
</table>
checkHorizWin(C,P) = <b><i>true</i></b>
</td>
<td><b><i>Reason:</i></b>
This test is a unique and distinct challenging case, because it tests whether or not checkHorizWin() can correctly identify successive tokens in both the left and right directions.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>X</td><td>X</td><td>X</td><td></td></tr>
</table>
<table><tr><td>R = 0</td><td>C = 6</td><td>P = X</td></tr></table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>X</td><td>X</td><td>X</td><td><b>X</b></td></tr>
</table>
checkHorizWin(C,P) = <b><i>true</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, because it tests whether or not checkHorizWin() can correctly identify a win when the winning token is placed in the rightmost possible row.</td>
</tr>
</table>

public boolean checkVertWin(int c)

<table>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
<table><tr><td>R = 3</td><td>C = 0</td><td>P = X</td></tr></table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>X</b></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
checkVertWin(C,P) = <b><i>true</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct routine case, as it tests whether or not checkVertWin() can correctly identify a typical vertical win scenario.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
</table>
<table><tr><td>R = 3</td><td>C = 3</td><td>P = O</td></tr></table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td><b>O</b></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
</table>
checkVertWin(C,P) = <b><i>false</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct routine case, as it tests whether or not checkVertWin() can correctly identify a scenario where a vertical win will not occur.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
<table><tr><td>R = 6</td><td>C = 0</td><td>P = X</td></tr></table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td><b>X</b></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
checkVertWin (C,P) = <b><i>true</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not checkVertWin() can correctly identify a win when the winning token is placed in the uppermost possible row.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
<table><tr><td>R = 0</td><td>C = 1</td><td>P = X</td></tr></table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td><b>X</b></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
checkVertWin (C,P) = <b><i>false</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct challenging case, as it tests whether or not checkVertWin() functions correctly when the board only has one token, in the column that checkVertWin() is called on.</td>
</tr>
</table>

public boolean checkDiagWin(int c)

Input

State: (number to win = 4)

		X	O			
	X	O	O			
X	O	O	O			

R = 3	C = 3	P = X

Output

6x7 GameBoard

			X			
		X	O			
	X	O	O			
X	O	O	O			

checkDiagWin(C,P) = ***true***

Reason:

This test case is a unique and distinct routine case, as it tests whether or not checkDiagWin() can correctly identify a typical win with the successive tokens in the upwards diagonal direction.

Input

State: (number to win = 4)

X						
O	X					
O	O	X				
O	O	O				

R = 0	C = 3	P = O

Output

6x7 GameBoard

X						
O	X					
O	O	X				
O	O	O	**X**			

checkDiagWin(C,P) = ***true***

Reason:

This test case is a unique and distinct routine case, as it tests whether or not checkDiagWin() can correctly identify a typical win with the successive tokens in the downwards diagonal direction.

Input

State: (number to win =4)

X						
O						
O	O	X				
O	O	O	X			

R = 2	C = 1	P = X

Output

6x7 GameBoard

X						
O	**X**					
O	O	X				
O	O	O	X			

checkDiagWin(C,P) = ***true***

Reason:

This test case is a unique and distinct challenging case, as it tests whether or not checkDiagWin() can correctly identify a win when the winning token is placed in the middle of a upwards-diagonal succession.

Input

State: (number to win =4)

			X			
		X	O			
		O	O			
X	O	O	O			

R = 1	C = 1	P = X

Output

6x7 GameBoard

			X			
		X	O			
	X	O	O			
X	O	O	O			

checkDiagWin(C,P) = ***true***

Reason:

This test case is a unique and distinct challenging case, as it tests whether or not checkDiagWin() can correctly identify a win when the winning token is placed in the middle of a downwards-diagonal succession.

Input
State: (number to win =4)

R = 0	C = 1	P = X

Output
6x7 GameBoard

	X					

checkDiagWin(C,P) = ***false***

Reason:
This test case is a unique and distinct boundary case, as it tests whether or not checkDiagWin() can correctly process a column with only one token in it.

Input
State: (number to win =4)

	X					
	X					
	X	X	X			
X	X	X	X			

R = 2	C = 3	P = X

Output
6x7 GameBoard

	X					
	X	**X**				
	X	X	X			
X	X	X	X			

checkDiagWin(C,P) = ***false***

Reason:

This test case is a unique and distinct challenging case, as it tests whether or not the algorithm to detect successive diagonal tokens works correctly.

Input
State: (number to win =4)

		X	O			
	X	O	O			
X	O	O	X			

R = 0	C = 1	P = O

Output
6x7 GameBoard

			O			
		X	O			
	X	O	O			
X	O	O	X			

checkDiagWin(C,P) = ***false***

Reason:

This test case is a unique and distinct routine case, as it tests whether or not checkDiagWin() can correctly identify a typical no-win scenario.

public boolean checkTie()

<table>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>
checkTie() = <b><i>true</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not checkTie() can correctly identify a tie, i.e. when the board is completely full.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
checkTie() = <b><i>false</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not checkTie() can correctly process an empty board.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
checkTie() = <b><i>false</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not checkTie() can correctly process a board with only one token on it.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>O</td><td>X</td><td></td><td></td><td></td></tr>
</table>
</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>O</td><td>X</td><td></td><td></td><td></td></tr>
</table>
checkTie() = <b><i>false</i></b>
</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct routine case, as it tests whether or not checkTie() can correctly identify a typical no-tie situation.</td>
</tr>
</table>

public char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos = (3,5)</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
whatsAtPos(pos) = ‘ ‘</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not whatsAtPos() can correctly process a blank space on the board.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos = (0,0)</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
whatsAtPos(pos) = ‘X’</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not whatsAtPos() works on the leftmost and lowest board position.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
</table>
Pos = (5, 6)</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
</table>
whatsAtPos(pos) = ‘X’</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not whatsAtPos() works on the rightmost and highest board position.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos = (5,0)</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
whatsAtPos(pos) = ‘X’</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not whatsAtPos() works on the leftmost and highest board position.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>O</td><td>X</td><td></td><td></td><td></td></tr>
</table>
Pos = (2,1)</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>X</td><td>O</td><td>X</td><td></td><td></td><td></td></tr>
</table>
whatsAtPos(pos) = ‘X’</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct routine case, as it tests whether or not whatsAtPos() correctly extracts the contents of a position in the middle of the board.</td>
</tr>
</table>

public boolean isPlayerAtPos(BoardPosition pos, char p)

<table>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>Pos = (3,5)</td>
<td><b><i>Output</i></b>
6x7 GameBoard<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>whatsAtPos(pos, X) = <b><i>false</i></b></td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not isPlayerAtPos() can correctly process an empty board position.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>Pos = (0,0)</td>
<td><b><i>Output</i></b>
6x7 GameBoard<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>isPlayerAtPos(posdo,O) = <b><i>false</i></b></td>
<td><b><i>Reason:</i></b>
This test cause is a unique and distinct challenging case, as it tests whether or not isPlayerAtPos() can discern that a board position is (1) occupied and (2) occupied by the specific player token O.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>Pos = (1, 2)</td>
<td><b><i>Output</i></b>
6x7 GameBoard<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>isPlayerAtPos(pos, X) = <b><i>true</i></b></td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct routine case, as it tests whether or not isPlayerAtPos() can correctly identify whether a specific token p is at a specific position pos; position p is in the middle of the board.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>Pos = (0,0)</td>
<td><b><i>Output</i></b>
6x7 GameBoard<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>X</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>isPlayerAtPos(pos) = ‘X’</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not isPlayerAtPos() will work for the leftmost and lowest possible board position.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
</table>Pos = (5,6)</td>
<td><b><i>Output</i></b>
6x7 GameBoard<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>O</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>X</td></tr>
</table>isPlayerAtPos(pos) = ‘X’</td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not isPlayerAtPos() will work for the rightmost and highest possible board position.</td>
</tr>
</table>

public void placeToken(char p, int c)

<table>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos = (0,5) ; p = ‘X’</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>X</td><td></td></tr>
</table></td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not placeToken() will correctly place a token in an empty column.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win = 4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos = (5,1); p = ‘X’</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td><b>X</b></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>O</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>X</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not placeToken() will correctly place a token in a nearly-full column.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos = (0, 0); P = ‘X’</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>X</b></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not placeToken() will correctly place a token in the leftmost column.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos = (0,6); P = ‘X’</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td><b>X</b></td></tr>
</table></td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct boundary case, as it tests whether or not placeToken() will correctly place a token in the rightmost column.</td>
</tr>
<tr>
<td><b><i>Input</i></b>
State: (number to win =4)
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>O</td><td>X</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td>O</td><td></td><td></td><td></td></tr>
</table>
Pos = (2,2); p = ‘X’</td>
<td><b><i>Output</i></b>
6x7 GameBoard
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td><b>X</b></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>O</td><td>X</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td>O</td><td></td><td></td><td></td></tr>
</table></td>
<td><b><i>Reason:</i></b>
This test case is a unique and distinct routine case, as it tests whether or not placeToken() will correctly place a token in one of the middle columns.</td>
</tr>
</table>