CPSC 2150 Project Report

Andy Bodell

Requirements Analysis

Functional Requirements:

1. As a player I need to choose which column to place my marker in so I can beat my opponent.
2. As a player I need to see what the game board looks like before selecting a column, so I make a logical decision.
3. As a player I want to know the result of the game, so I know if I won, lost, or tied the opponent
4. As a player I can see where my markers are on the board so that I can implement a strategy to win
5. As a player I can only place one marker at a time so that it is a fair game between the opponent and I
6. As a player I need to alternate turns with the opponent so that the game is played fairly
7. As a player I need to see where the opponent placed their marker so that I can try and stop them from winning
8. As a player I need to know if a spot is blocked so I do not try and place a marker there
9. As a player I need to know the column values so I can place my marker in the desired spot
10. As a player I can pick again if I pick a column that does not exist, so I do not lose my turn
11. As a player, I can end the game in a tie by taking the last space on the board without getting enough in a row, so the game can end
12. As a player I can choose to play again, so I can play again
13. As a player I can choose how many consecutive tokens in a row to win, so I can use different strategies
14. As a player I can play with more than two players, so I can play with more of my friends
15. As a player I can make the board larger, so I can play longer games
16. As a player I can make the board smaller, so I can play shorter games
17. As a player I can choose what character my marker is, so I can play with something other than X and O
18. As a player I can choose a fast gameboard so the game runs faster on my computer
19. As a player I can choose a memory efficient gameboard so the game does not take up as much memory on my computer
20. As a player I can change the settings, so I can play different styles of games

Non-Functional Requirements

1. The game must be implemented in Java
2. The game must run on a Unix machine
3. The game must be a command line application
4. The game will be visible on the command line interface

5. The largest gameboard can be up to 100x100
6. The smallest gameboard can be as small as 3x3
7. The largest number in a row to win can be 25
8. The smallest number in a row to win can be 3
9. The max number of players is 10
10. The minimum number of players is 2
11. The documentation must be done in JavaDoc
12. There must be a GUI used to play the game
13. The game must implement the MVC architectural pattern

Deployment Instructions

Details in Projects 2-5.

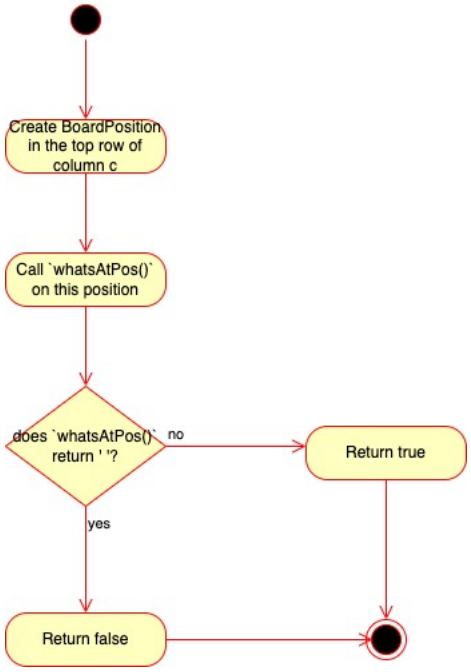
System Design

Interface: IGameBoard

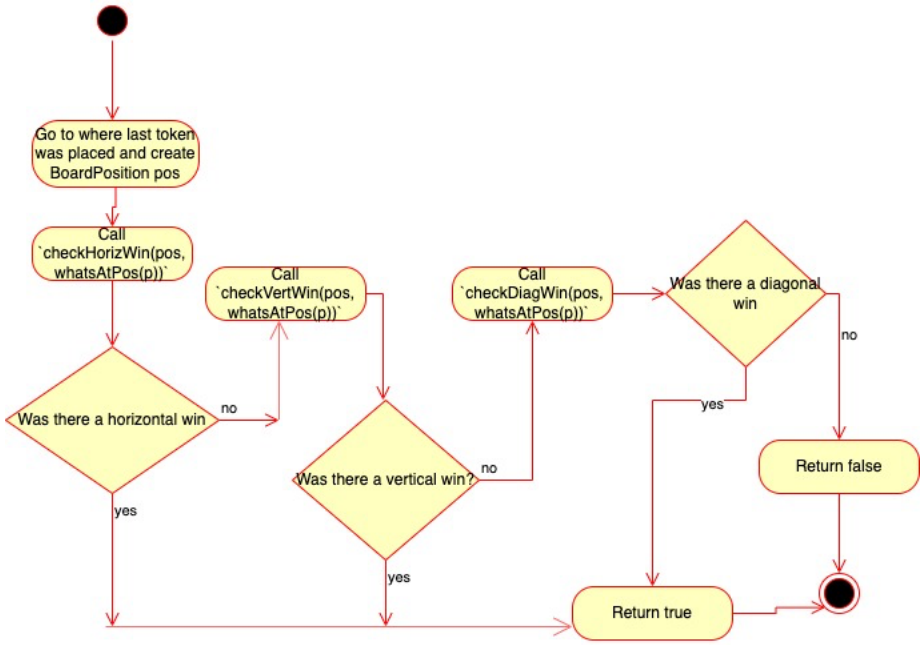
Class diagram

Activity diagrams:

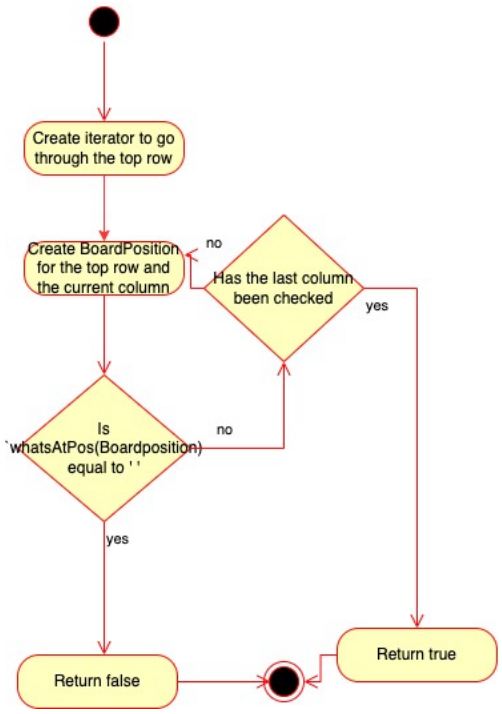
public default boolean checkIfFree(int c):



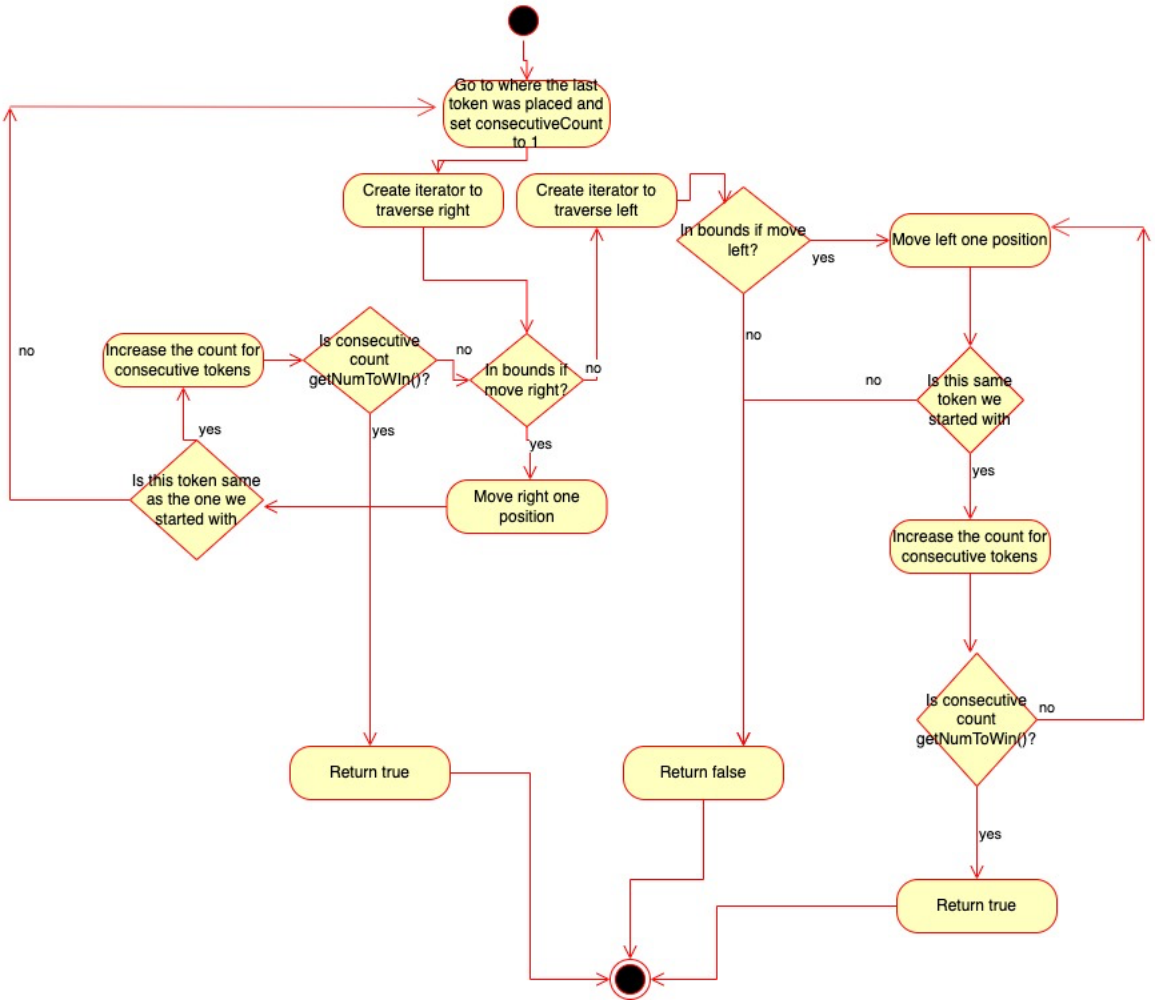
public default boolean checkForWin(int c):



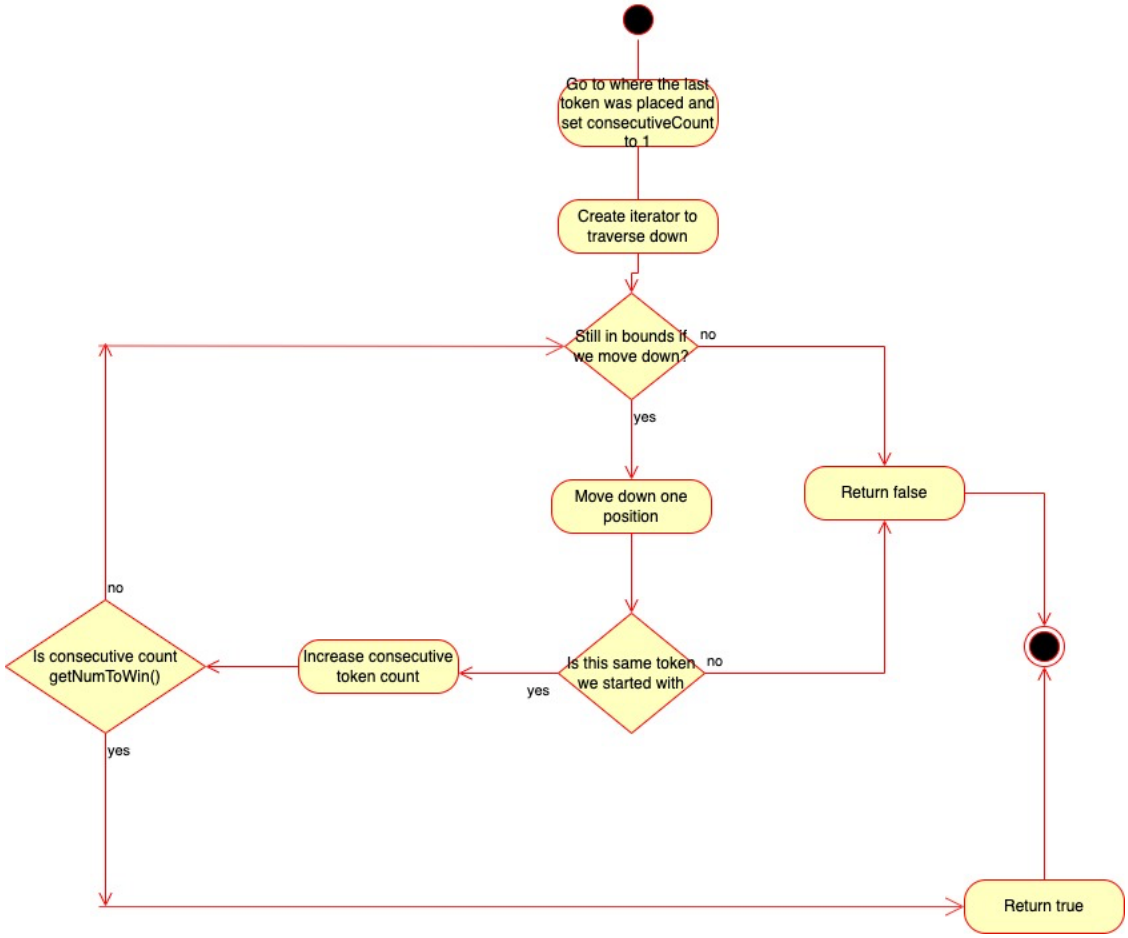
public default boolean checkTie():



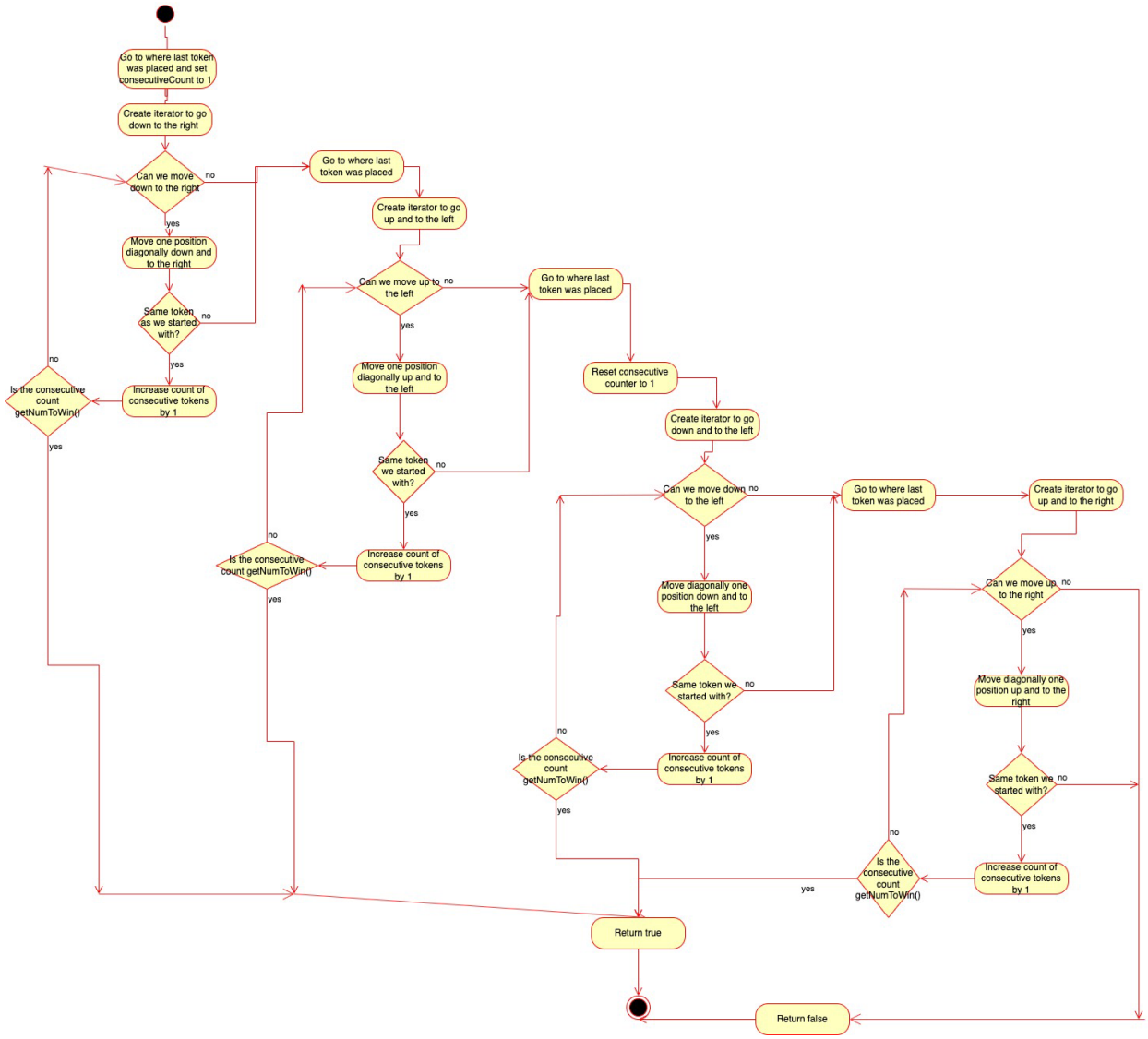
public default boolean checkHorizWin(BoardPosition pos, char p):



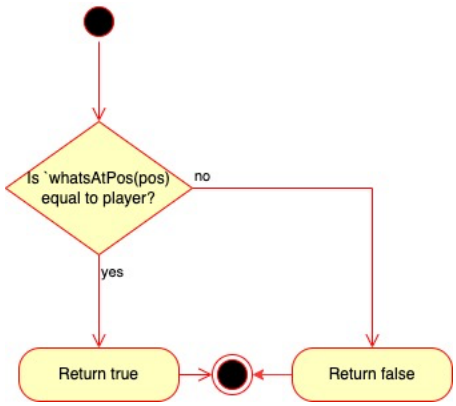
public default boolean checkVertWin(BoardPosition pos, char p):



public default boolean checkDiagWin(BoardPosition pos, char c):



public default boolean isPlayerAtPos(BoardPosition pos, char player):

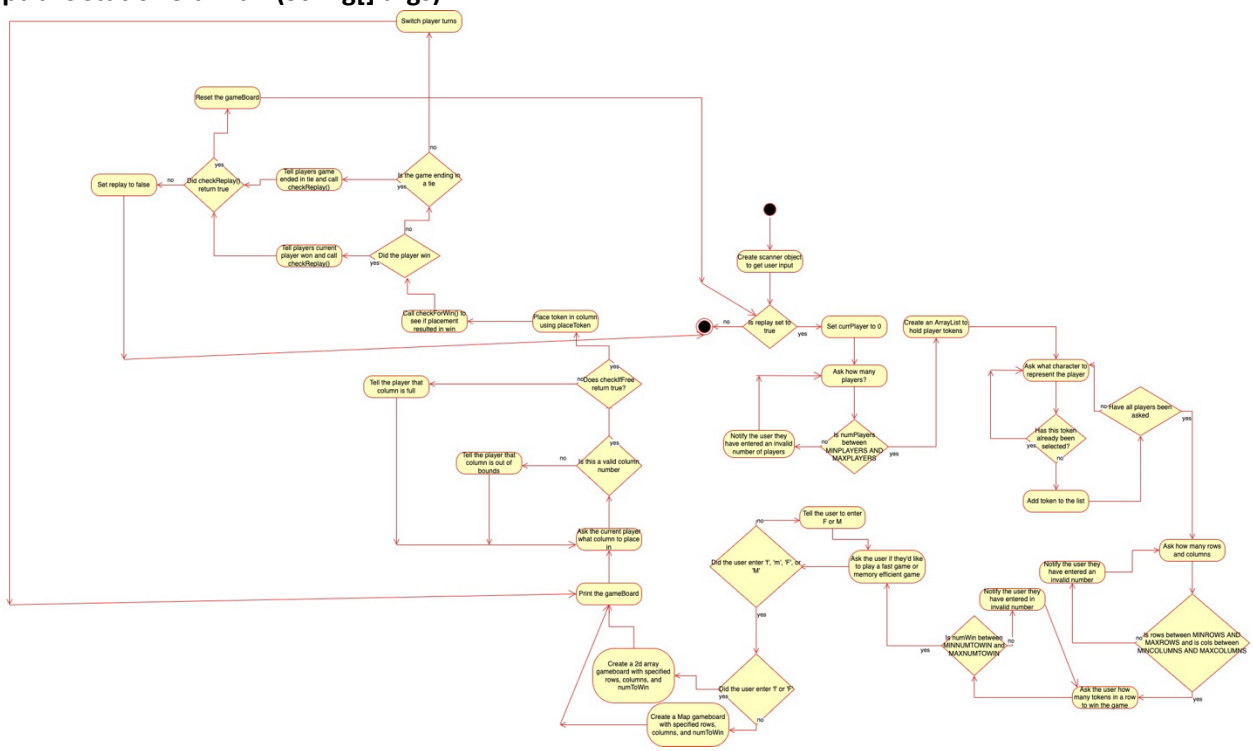


Class 1: GameScreen
Class diagram

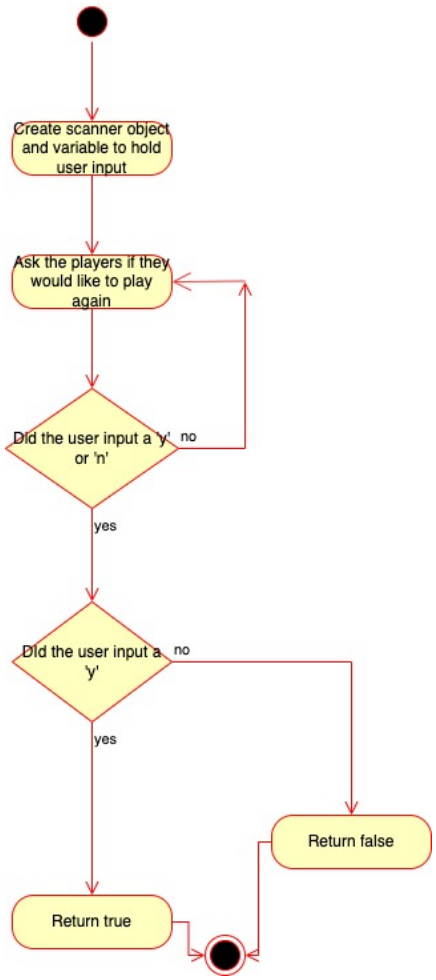
GameScreen
- replay: boolean[1] - gameBoard: IGameBoard[1]
- checkReplay(): bool + main(String[]): void

Activity diagrams

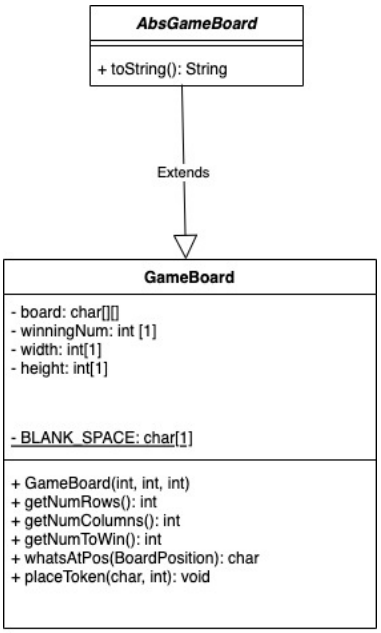
public static void main(String[] args):



private static boolean checkReplay():

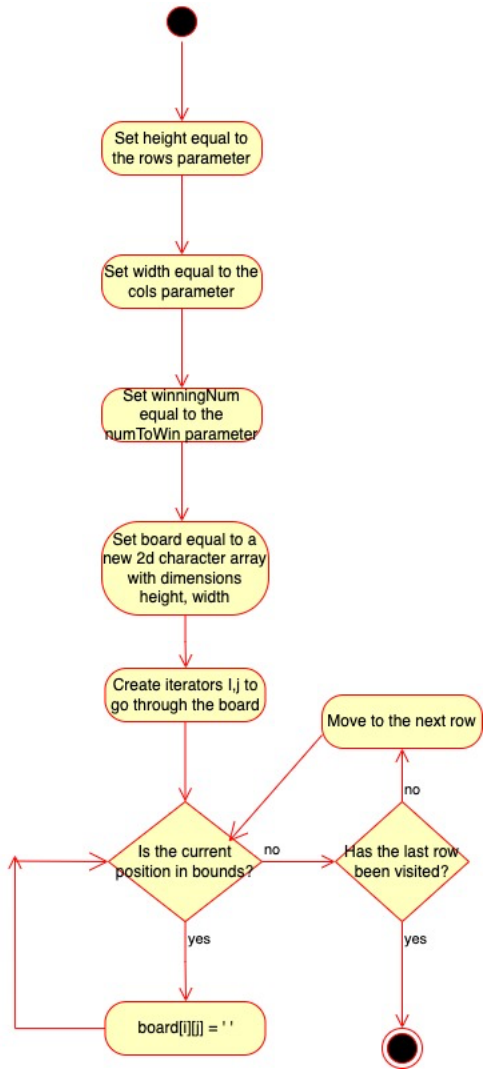

Create scanner object and variable to hold user input
Ask the players if they would like to play again
Did the user input a 'y' or 'n'
no
yes
Did the user input a 'y'
no
yes
Return false
Return true

Class 2: GameBoard

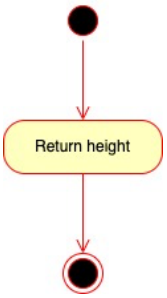


Activity Diagrams:

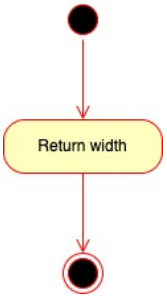
public GameBoard(int rows, int cols, int numToWin):



public int getNumRows():

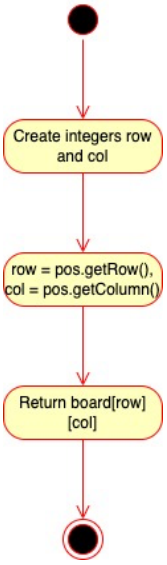


public int getNumColumns():

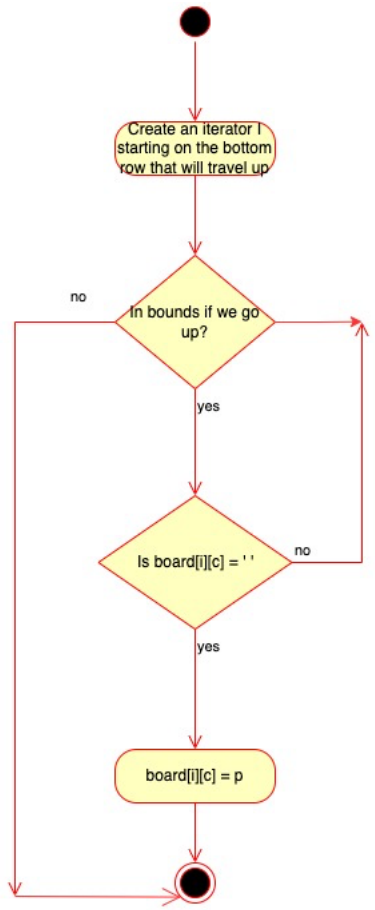


public int getNumToWin():

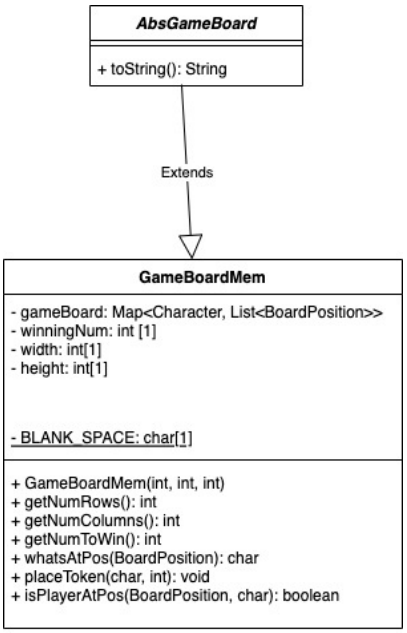
public char whatsAtPos(BoardPosition pos):



public void placeToken(char p, int c):

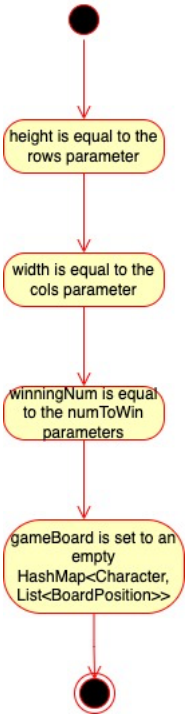


Class 3: GameBoardMem:

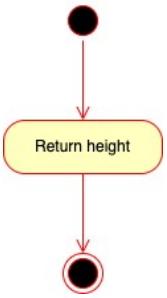


Activity Diagrams:

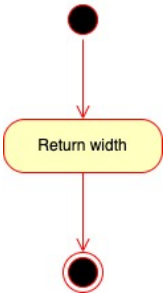
public GameBoardMem(int rows, int cols, int numToWin):



public int getNumRows():

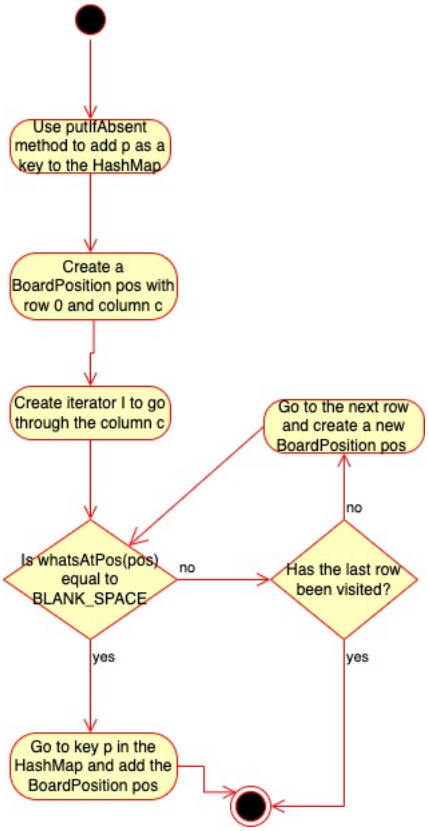


public int getNumColumns():

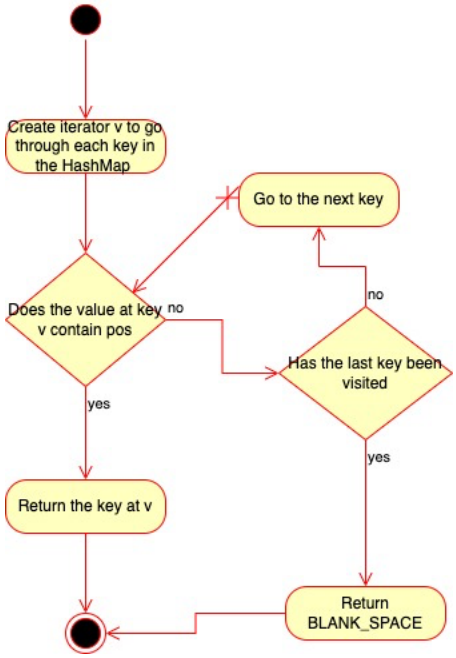


public int getNumToWin():

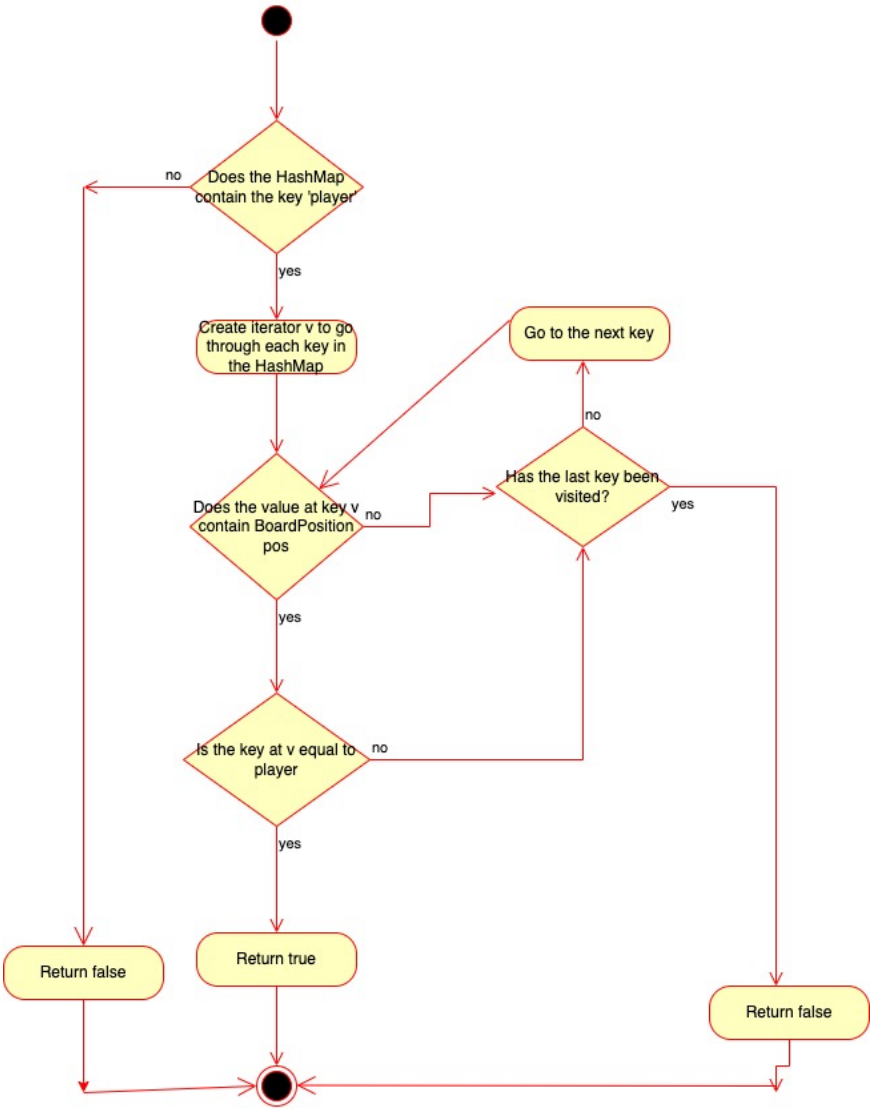
public void placeToken(char p, int c):



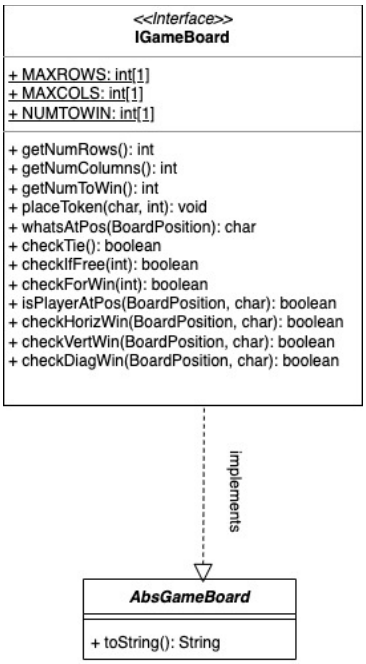
public char whatsAtPos(BoardPosition pos):



public Boolean isPlayerAtPos(BoardPosition pos, char player):

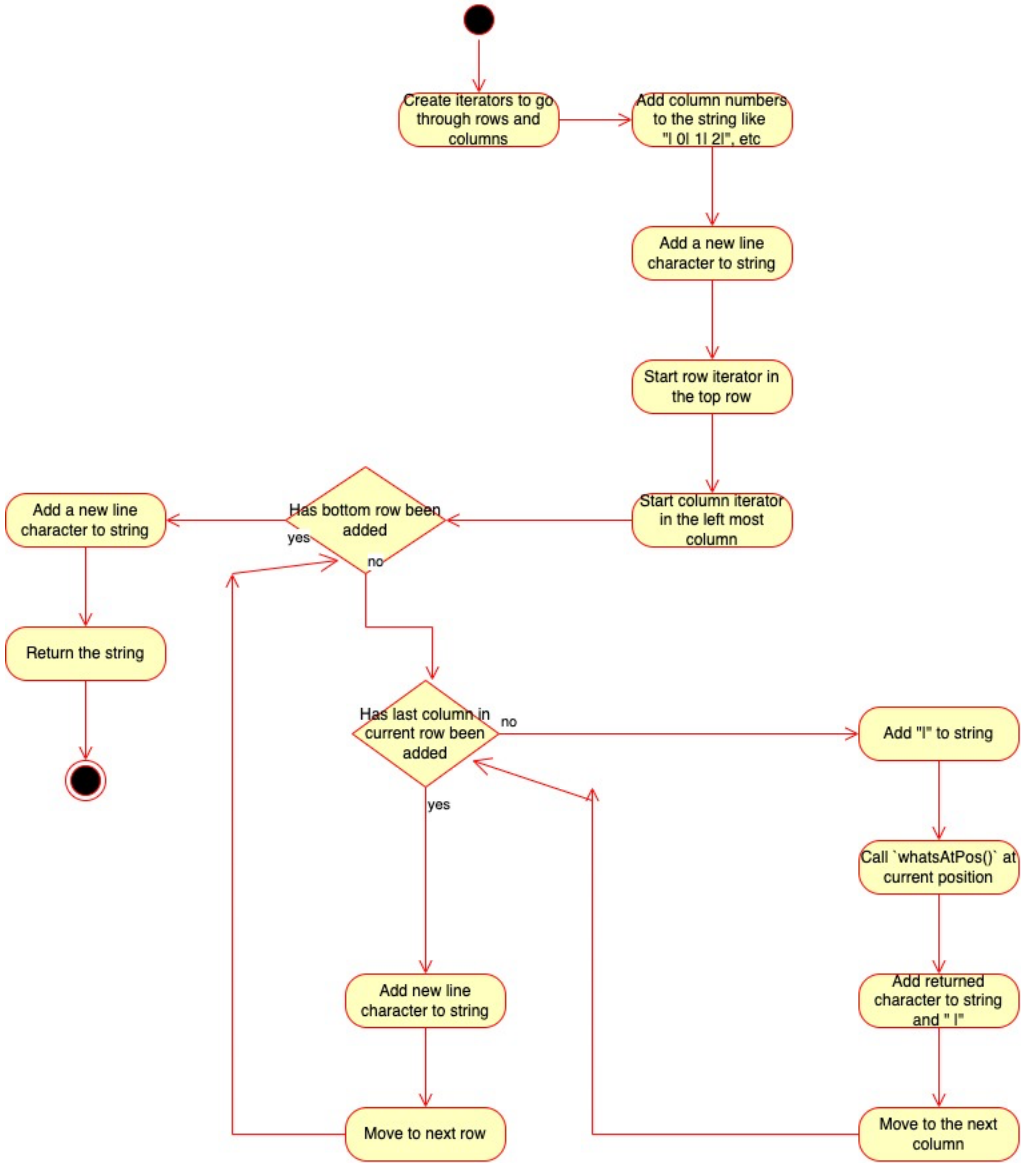


Class 4: AbsGameBoard



Activity Diagrams:

public String toString():



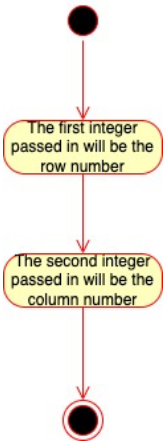
Class 5: BoardPosition

Class Diagram

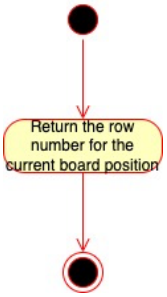
BoardPosition
- ROW_POS: int[1] {read-only} - COL_POS: int[1] {read-only}
+ BoardPosition(int, int) + getRow(): int + getColumn(): int + toString(): String + equals(Object): boolean

Activity Diagram

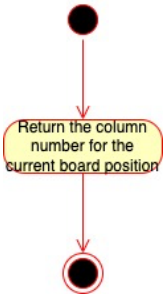
public BoardPosition(int row, int column):



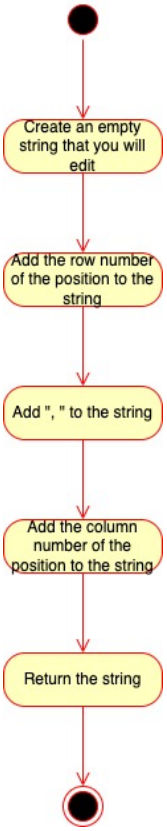
public int getRow():



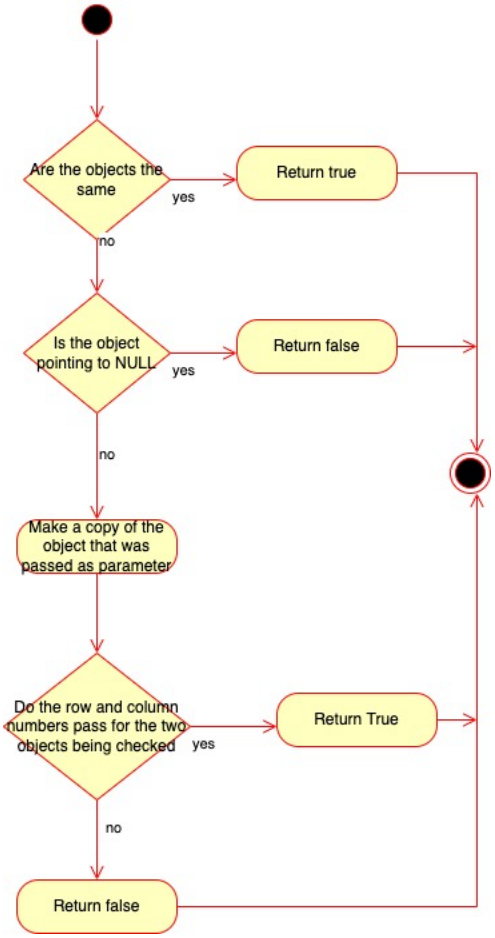
public int getColumn():



public String toString():



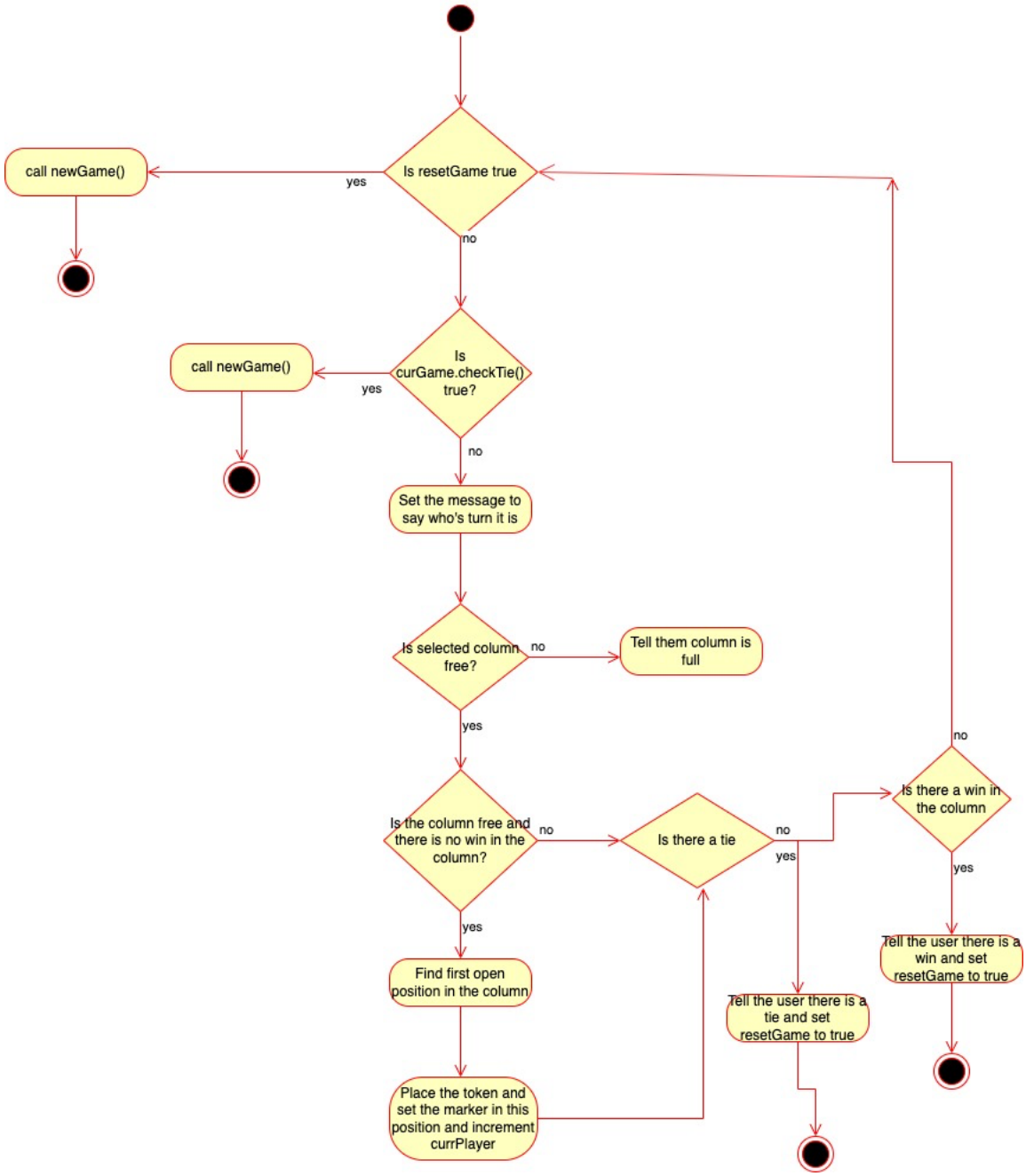
public equals(Object o):



class ConnectXController:

ConnectXController
- curGame: IGameBoard[1] - screen: ConnectXView[1] - possibleTokens: char[10] - numPlayers: int[1] - resetGame: boolean[1] - currPlayer: int[1] + MAX_PLAYERS: int[1]
+ ConnectXController(IGameBoard, ConnectXView, int) + processButtonClick(int): void - newGame(): void

Activity Diagrams:
public void processButtonClick(int col)



Test Cases

Details in Project 4.

GameBoard(int row, int col, int numToWin)/GameBoardMem(int row, int col, int numToWin)

<table>
<tr>
<td>Input:
No state, object has not been created yet

Row = 5
Col = 5
numToWin = 5</td>
<td>Output:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>
5x5 board created with all spaces initialized to blank</td>
<td>Reason:
This is a unique test because it tests the constructor call for parameters that aren’t on the boundary for the respective number of rows, columns, and number to win

Function name:
test_constructor</td>
</tr>
</table>

GameBoard(int row, int col, int numToWin)/GameBoardMem(int row, int col, int numToWin)

<table>
<tr>
<td>Input:
No state, object has not been created yet

Row = 3
Col = 3
numToWin = 3</td>
<td>Output:
<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>
3x3 board created with all spaces initialized to blank</td>
<td>Reason:
This is a unique test because it tests the constructor call for the minimum parameters that are allowed for row, col, and numToWin

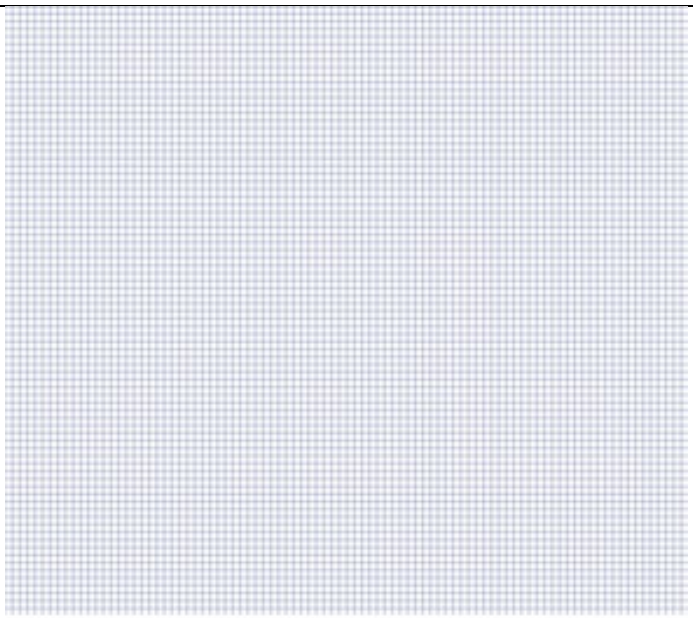
Function name:
test_min_constructor</td>
</tr>
</table>

GameBoard(int row, int col, int numToWin)/GameBoardMem(int row, int col, int numToWin)

<table>
<tr>
<td>Input:
No state, object has not been created yet

Row = 100
Col = 100
numToWin = 25</td>
<td>Output:</td>
<td>Reason:
This is a unique test because it tests the constructor call for the maximum parameters that are allowed for row, col, and numToWin

Function name:
test_max_constructor</td>
</tr>
</table>

	 shutterstock.com · 1541684540 100x100 board created with all spaces initialized to blank (can't make a 100x100 table in word so this picture will have to do)	

boolean checkIfFree(int c)

Input: State: c = 0	Output: checkIfFree = true state of the board does not change	Reason: This is a unique test case because it tests a column that is not empty or full and it is also not close to being full Function name: test_checkIfFree_partialFull

O				
X				
O				

boolean checkIfFree(int c)

<table>
<tr>
<td>Input:
State:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>
c = 0</td>
<td>Output:

checkIfFree = true

state of the board does not change</td>
<td>Reason:
This is a unique test case because it calls the function when the board is completely empty which is an edge case.

Function name:
test_checkIfFree_empty</td>
</tr>
</table>

boolean checkIfFree(int c)

<table>
<tr>
<td>Input:
State:
<table><tr><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td><td></td></tr></table>
c = 0</td>
<td>Output:

checkIfFree = false

state of the board does not change</td>
<td>Reason:
This is a unique test case because it tests a column which is completely full which is another edge case.

Function name:
test_checkIfFree_full</td>
</tr>
</table>

boolean checkHorizWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 5
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr></table>
pos.getRow = 0
pos.getCol = 0
p = ‘O’</td>
<td>Output:

checkHorizWin = true

state of the board does not change</td>
<td>Reason:
This is a unique test case because it starts from the beginning of the row so it will have to traverse right in order to find the consecutive tokens

Function name:
test_checkHorizWin_beginning</td>
</tr>
</table>

boolean checkHorizWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 5
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr></table>
pos.getRow = 0
pos.getCol = 4
p = ‘O’</td>
<td>Output:

checkHorizWin = true

state of the board does not change</td>
<td>Reason:
This is a unique test case because it starts at the end of the row so it will have to traverse left in order to find the consecutive tokens

Function name:
test_checkHorizWin_end</td>
</tr>
</table>

boolean checkHorizWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 5
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td>O</td><td>X</td><td></td></tr></table>
pos.getRow = 0
pos.getCol = 0
p = ‘O’</td>
<td>Output:

checkHorizWin = false

state of the board does not change</td>
<td>Reason:
This is a unique test case because it has to traverse right but it is also testing to make sure that the function will return false if it does not find consecutive positions with the same character

Function name:
test_checkHorizWin_begin_noWin</td>
</tr>
</table>

boolean checkHorizWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 5
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr></table>
pos.getRow = 0
pos.getCol = 1
p = ‘O’</td>
<td>Output:

checkHorizWin = true

state of the board does not change</td>
<td>Reason:
This is a unique test case because it starts from a position that is in the middle of the row so it will have to traverse to the left and to the right in order to find consecutive tokens

Function name:
test_checkHorizWin_fromMiddle</td>
</tr>
</table>

boolean checkVertWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>
pos.getRow = 0
pos.getCol = 0
p = ‘X’</td>
<td>Output:

checkVertWin = false

state of the board does not change</td>
<td>Reason:
This is a unique test case because it tests when the board is completely empty which is an edge case because there are no tokens on the board

Function name:
test_checkVertWin_emptyBoard</td>
</tr>
</table>

boolean checkVertWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td><td></td></tr></table>
pos.getRow = 4
pos.getCol = 0
p = ‘O’</td>
<td>Output:

checkVertWin = false

state of the board does not change</td>
<td>Reason:
This is a unique test case because it tests that the function will return false if it does not find enough consecutive characters in a row from the starting position

Function name:
test_checkVertWin_inBetween</td>
</tr>
</table>

boolean checkVertWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td><td></td><td></td></tr></table>
pos.getRow = 2
pos.getCol = 0
p = ‘O’</td>
<td>Output:

checkVertWin = true

state of the board does not change</td>
<td>Reason:
This is a unique test case because it tests that our function will traverse down the board and if there are numToWin consecutive tokens then it will return true

Function name:
test_checkVertWin_fromTop</td>
</tr>
</table>

boolean checkVertWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td><td></td><td></td></tr></table>
pos.getRow = 1
pos.getCol = 0
p = ‘O’</td>
<td>Output:

checkVertWin = false

state of the board does not change</td>
<td>Reason:
This is a unique test case because it shows that even if the function does not detect a different player token it will return false if there are not numToWin consecutive tokens in a row

Function name:
test_checkVertWin_short</td>
</tr>
</table>

boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table>
pos.getRow = 0
pos.getCol = 0
p = ‘O’</td>
<td>Output:

checkDiagWin = false

state of the board does not change</td>
<td>Reason:
This is a unique test case because it tests when the board is completely empty which is an edge case because there are no tokens on the board

Function name:
test_checkDiagWin_empty</td>
</tr>
</table>

boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td>X</td><td>O</td></tr><tr><td></td><td>O</td><td>X</td></tr><tr><td>O</td><td>X</td><td>O</td></tr></table>
pos.getRow = 0
pos.getCol = 0
p = ‘O’</td>
<td>Output:

checkDiagWin = true

state of the board does not change</td>
<td>Reason:
This is a unique test case because it is traversing the diagonal from the bottom left up to the upper right hand corner

Function name:
test_checkDiagWin_bLeft_to_uRight</td>
</tr>
</table>

boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td>X</td><td>O</td></tr><tr><td></td><td>O</td><td>X</td></tr><tr><td>O</td><td>X</td><td>O</td></tr></table>
pos.getRow = 2
pos.getCol = 2
p = ‘O’</td>
<td>Output:

checkDiagWin = true

state of the board does not change</td>
<td>Reason:
This is a unique test case because it is traversing the diagonal from the upper right corner to the bottom left corner

Function name:
test_checkDiagWin_uRight_to_bLeft</td>
</tr>
</table>

boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>O</td><td>X</td><td></td></tr><tr><td>X</td><td>O</td><td></td></tr><tr><td>O</td><td>X</td><td>O</td></tr></table>
pos.getRow = 0
pos.getCol = 2
p = ‘O’</td>
<td>Output:

checkDiagWin = true

state of the board does not change</td>
<td>Reason:
This is a unique test case because it is traversing the diagonal from the bottom right corner to the upper left corner

Function name:
test_checkDiagWin_bRight_to_uLeft</td>
</tr>
</table>

boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>O</td><td>X</td><td></td></tr><tr><td>X</td><td>O</td><td></td></tr><tr><td>O</td><td>X</td><td>O</td></tr></table>
pos.getRow = 2
pos.getCol = 0
p = ‘O’</td>
<td>Output:

checkDiagWin = true

state of the board does not change</td>
<td>Reason:
This is a unique test case because it is traversing the diagonal from the upper left corner to the bottom right corner

Function name:
test_checkDiagWin_uLeft_to_bRight</td>
</tr>
</table>

boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td>O</td></tr><tr><td>X</td><td>X</td><td>X</td></tr><tr><td>O</td><td>O</td><td>O</td></tr></table>
pos.getRow = 0
pos.getCol = 0
p = ‘O’</td>
<td>Output:

checkDiagWin = False

state of the board does not change</td>
<td>Reason:
This is a unique test case because there is a false character along the diagonal so the function will not count numToWin consecutive characters and will return false

Function name:
test_checkDiagWin_inBetween</td>
</tr>
</table>

boolean checkDiagWin(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>O</td><td>X</td><td>O</td></tr><tr><td>X</td><td>O</td><td>X</td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table>
pos.getRow = 0
pos.getCol = 0
p = ‘X’</td>
<td>Output:

checkDiagWin = false

state of the board does not change</td>
<td>Reason:
This is a unique test case because the entire board is full and the game is tied, which is an edge case because we’re testing when the entire board is full and there is still no win

Function name:
test_checkDiagWin_ifTie</td>
</tr>
</table>

boolean checkTie()

<table>
<tr>
<td>Input:
State: numToWin = 3
<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>Output:

checkTie = false

state of the board does not change</td>
<td>Reason:
This is a unique test case because we are testing on an empty board which is an edge case because the board is completely empty

Function name:
test_checkTie_empty</td>
</tr>
</table>

boolean checkTie()

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>O</td><td>X</td><td>O</td></tr><tr><td>X</td><td>O</td><td>X</td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table></td>
<td>Output:

checkTie = true

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking when the board is completely full and no player has won which is an edge case

Function name:
test_checkTie_full</td>
</tr>
</table>

boolean checkTie()

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>X</td><td></td><td></td></tr><tr><td>O</td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table></td>
<td>Output:

checkTie = false

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking when the board is partially full, but not close to full

Function name:
test_checkTie_partialFull</td>
</tr>
</table>

boolean checkTie()

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>O</td><td>X</td><td></td></tr><tr><td>X</td><td>O</td><td>X</td></tr><tr><td>O</td><td>X</td><td>O</td></tr></table></td>
<td>Output:

checkTie = false

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking when the board is very close to being full and the top row is one space away from being full

Function name:
test_checkTie_full</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>X</td><td></td><td></td></tr><tr><td>O</td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>
pos.getRow = 0
pos.getCol = 0</td>
<td>Output:

whatsAtPos = ‘X’

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking what was placed in the very first position to make sure the first token goes to the bottom row

Function name:
test_whatsAtPos_firstSpace</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td>X</td></tr><tr><td></td><td></td><td>O</td></tr><tr><td></td><td></td><td>X</td></tr></table>
pos.getRow = 2
pos.getCol = 0</td>
<td>Output:

whatsAtPos = ‘X’

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking what was placed in the very last position on the board which is an edge case, and it makes sure tokens can be placed on top of each other

Function name:
test_whatsAtPos_lastSpace</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:
State: numToWin = 3
<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>
pos.getRow = 0
pos.getCol = 0</td>
<td>Output:

whatsAtPos = ‘ ’

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking that whatsAtPos will not return a token at our edge case of the board being completely empty

Function name:
test_whatsAtPos_empty</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table>
pos.getRow = 1
pos.getCol = 0</td>
<td>Output:

whatsAtPos = ‘ ’

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking that the tokens are placed on the bottom row first, and there will still be blank spaces above them

Function name:
test_whatsAtPos_emptySpace</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>X</td><td>X</td><td></td></tr><tr><td>O</td><td>O</td><td>X</td></tr><tr><td>X</td><td>X</td><td>O</td></tr></table>
pos.getRow = 2
pos.getCol = 2</td>
<td>Output:

whatsAtPos = ‘ ’

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking that even though the board is almost full, even if there is one empty space left whatsAtPos will still return an empty space

Function name:
test_whatsAtPos_almostFull_emptySpace</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<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>
pos.getRow = 0
pos.getCol = 0
p = ‘X’</td>
<td>Output:

isPlayerAtPos = false

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking one of the edge cases when the board is completely empty

Function name:
test_isPlayerAtPos_empty</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>X</td><td></td><td></td></tr><tr><td>O</td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>
pos.getRow = 0
pos.getCol = 0
p = ‘X’</td>
<td>Output:

isPlayerAtPos = True

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking the very first position on the board which is a boundary position and makes sure tokens are placed on the bottom row first

Function name:
test_isPlayerAtPos_firstSpot</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td>X</td></tr><tr><td></td><td></td><td>O</td></tr><tr><td></td><td></td><td>X</td></tr></table>
pos.getRow = 2
pos.getCol = 2
p = ‘X’</td>
<td>Output:

isPlayerAtPos = True

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking the very last position on the board which is a boundary position and makes sure tokens can be stacked on top of each other

Function name:
test_isPlayerAtPos_empty</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td></tr><tr><td>X</td><td>X</td><td></td></tr></table>
pos.getRow = 2
pos.getCol = 2
p = ‘X’</td>
<td>Output:

isPlayerAtPos = false

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are making sure the function will return false if we check for a token at position and it is not there

Function name:
test_isPlayerAtPos_wrongChar</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char p)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>X</td><td>X</td><td>X</td></tr><tr><td>O</td><td>O</td><td>O</td></tr><tr><td>X</td><td>X</td><td>X</td></tr></table>
pos.getRow = 1
pos.getCol = 1
p = ‘O’</td>
<td>Output:

isPlayerAtPos = true

state of the board does not change</td>
<td>Reason:
This is a distinct test case because we are checking one of the boundary states where the board is completely full and making sure the tokens are stacked on top of each other when they are placed

Function name:
test_isPlayerAtPos_fullBoard</td>
</tr>
</table>

void placeToken(char p, int c)

<table>
<tr>
<td>Input:
State: numToWin = 3
<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>
p = ‘X’
c = 1</td>
<td>Output:

<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td></tr></table></td>
<td>Reason:
This is a distinct test case because I am placing in a completely empty column

Function name:
test_placeToken_empty</td>
</tr>
</table>

void placeToken(char p, int c)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td>X</td><td>X</td><td></td></tr><tr><td>O</td><td>O</td><td>X</td></tr><tr><td>X</td><td>X</td><td>O</td></tr></table>
p = ‘O’
c = 2</td>
<td>Output:

<table><tr><td>X</td><td>X</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td></tr><tr><td>X</td><td>X</td><td>O</td></tr></table></td>
<td>Reason:
This is a distinct test case because I am placing in a column that will fill up the entire board

Function name:
test_placeToken_almostFull</td>
</tr>
</table>

void placeToken(char p, int c)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>
p = ‘X’
c = 0</td>
<td>Output:
<table><tr><td>X</td><td></td><td></td></tr><tr><td>O</td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table></td>
<td>Reason:
This is a distinct test case because I am placing in a column that is very close to being full

Function name:
test_placeToken_fillCol</td>
</tr>
</table>

void placeToken(char p, int c)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>X</td><td>O</td><td></td></tr></table>
p = ‘X’
c = 2</td>
<td>Output:
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>X</td><td>O</td><td>X</td></tr></table></td>
<td>Reason:
This is a distinct test case because I am placing in a row that is very close to being full

Function name:
test_placeToken_fillRow</td>
</tr>
</table>

void placeToken(char p, int c)

<table>
<tr>
<td>Input:
State: numToWin = 3
<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table>
p = ‘O’
c = 0</td>
<td>Output:
<table><tr><td></td><td></td><td></td></tr><tr><td>O</td><td></td><td></td></tr><tr><td>X</td><td></td><td></td></tr></table></td>
<td>Reason:
This is a distinct test case because I am placing in a column that is not empty but it is not close to being full

Function name:
test_placeToken_partialCol</td>
</tr>
</table>

Deployment:

1. Make a run configuration in IntelliJ with the main class as ConnectXApp. Run the main configuration then you can play the game