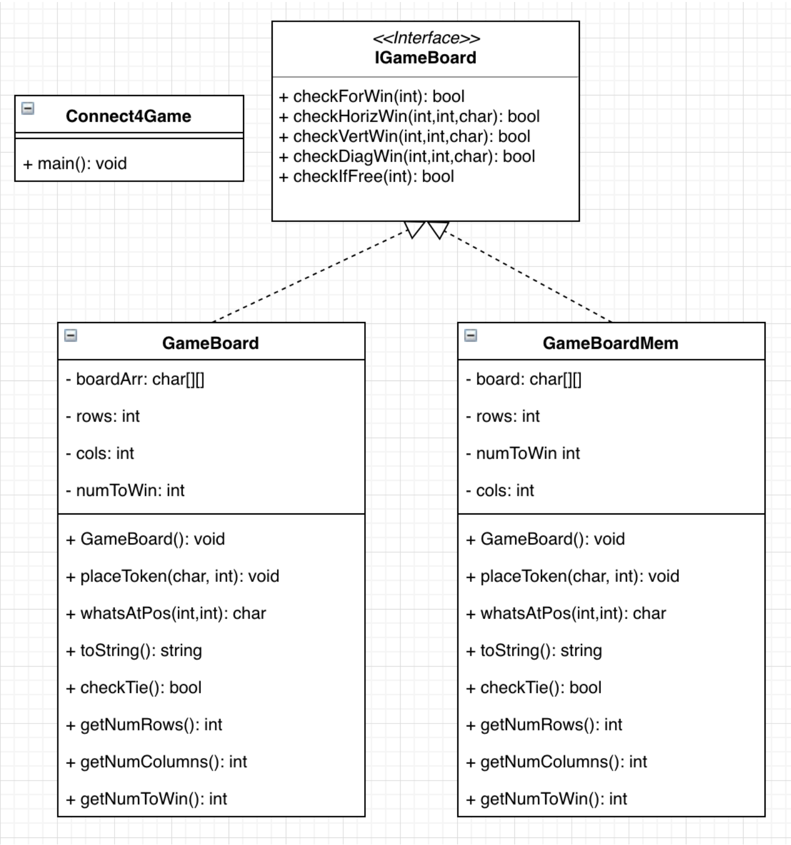
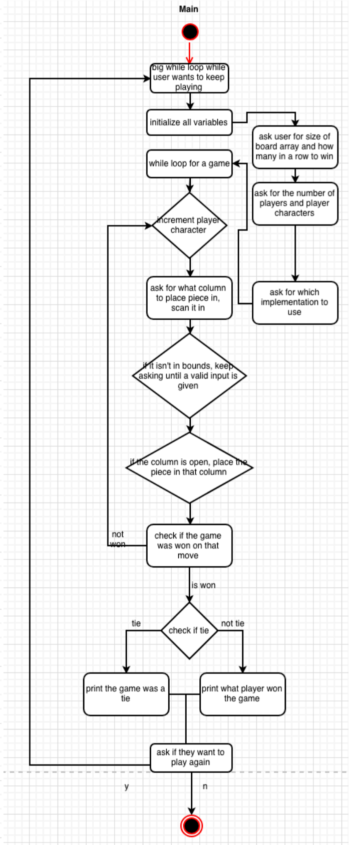
Homework 4 – Ben Joye
CPSC 2150

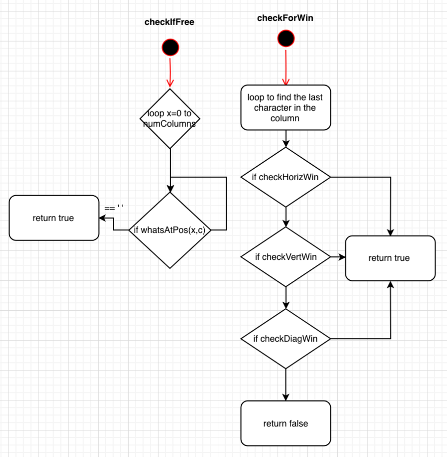
Requirements Analysis:

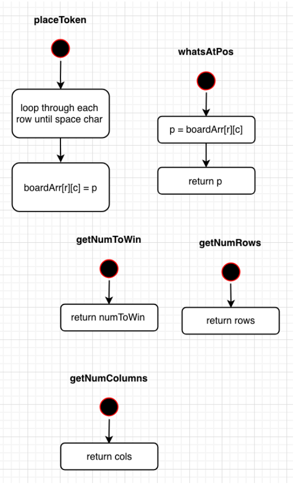
- **Functional**
  - As a user, I can input what column to place my piece into so that it is stored in the board array
  - As a user, I can see the board array after every turn.
  - As a user, both players will alternate turns.
  - As a user, I can choose to play again so that the game will keep running.
  - As a user, I can input numbers to decide the size of the game board.
  - As a user, I can input numbers to decide the number of pieces in a row you need to win.
  - As a user, I can input an integer to set how many players can play.
  - As a user, I can input characters to set the symbols for each player.
  - As a user, I can input characters to decide which implementation to use.
- **Non-Functional**
  - The system must be able to detect when a player has won.
  - The system must display which players turn it is.
  - The system must keep track of every move and display the board after every turn.
  - The system must handle a board size of up to 100 rows and 100 columns.
  - The system only lets dimensions from 3 to 100 for the board array.

Design:


Connect4Game
+ main(): void
<<Interface>>
IGameBoard
+ checkForWin(int): bool
+ checkHorizWin(int,int,char): bool
+ checkVertWin(int,int,char): bool
+ checkDiagWin(int,int,char): bool
+ checkIfFree(int): bool
GameBoard
- boardArr: char[][]
- rows: int
- cols: int
- numToWin: int
+ GameBoard(): void
+ placeToken(char, int): void
+ whatsAtPos(int,int): char
+ toString(): string
+ checkTie(): bool
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int
GameBoardMem
- board: char[][]
- rows: int
- numToWin int
- cols: int
+ GameBoard(): void
+ placeToken(char, int): void
+ whatsAtPos(int,int): char
+ toString(): string
+ checkTie(): bool
+ getNumRows(): int
+ getNumColumns(): int
+ getNumToWin(): int


Main
big while loop while user wants to keep playing
initialize all variables
ask user for size of board array and how many in a row to win
while loop for a game
ask for the number of players and player characters
increment player character
ask for what column to place piece in, scan it in
ask for which implementation to use
if it isn't in bounds, keep asking until a valid input is given
if the column is open, place the piece in that column
not won
check if the game was won on that move
is won
tie
check if tie
not tie
print the game was a tie
print what player won the game
ask if they want to play again
y
n


checkIfFree
loop x=0 to numColumns
return true
== ' '
if whatsAtPos(x,c)
checkForWin
loop to find the last character in the column
if checkHorizWin
if checkVertWin
return true
if checkDiagWin
return false


placeToken
loop through each row until space char
boardArr[r][c] = p
whatsAtPos
p = boardArr[r][c]
return p
getNumToWin
return numToWin
getNumRows
return rows
getNumColumns
return cols

Testing:

GameBoard()

void testIGameBoard_min()

<table>
<tr>
<td>Input State: null</td>
<td>Output State:<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>Empty board</td>
<td>Reason: tests the minimum values for rows and columns</td>
</tr>
</table>

void testIGameBoard_max()

Input State: null	Output State: 100x100 board	Reason: tests the maximum values for rows and columns

void testIGameBoard_minandmax()

Input State: null	Output State: 3x100 board	Reason: tests the minimum and maximum values for rows and columns together

Char WhatsAtPos(int r, int c)

void testWhatsAtPos_origin()

<table>
<tr>
<td>Input State:<table><tr><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></tr><tr><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></tr></table>R=0, C=0</td>
<td>Output State:

Return ‘X’

State of the board is unchanged</td>
<td>Reason:

Testing the first spot in the array, could be problematic</td>
</tr>
</table>

void testWhatsAtPos_2ndrow()

<table>
<tr>
<td>Input State:<table><tr><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></tr><tr><td>O</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></tr></table>R=1, C=0</td>
<td>Output State:

Return ‘O’

State of the board is unchanged</td>
<td>Reason:

Testing if there are multiple characters in a column</td>
</tr>
</table>

void testWhatsAtPos_top()

<table>
<tr>
<td>Input State:<table><tr><td>X</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></tr><tr><td>X</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></tr></table>R=3, C=0</td>
<td>Output State:

Return ‘X’

State of the board is unchanged</td>
<td>Reason:

Testing if there are multiple characters in a column</td>
</tr>
</table>

void testWhatsAtPos_3rdrow()

<table>
<tr>
<td>Input State:<table><tr><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></tr><tr><td>X</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></tr></table>R=2, C=0</td>
<td>Output State:

Return ‘X’

State of the board is unchanged</td>
<td>Reason:

Testing if there are more than 2 characters in a column</td>
</tr>
</table>

void testWhatsAtPos_middle()

<table>
<tr>
<td>Input State:<table><tr><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></tr><tr><td></td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>X</td><td></td><td></td></tr></table>R=2, C=3</td>
<td>Output State:

Return ‘X’

State of the board is unchanged</td>
<td>Reason:

Testing other columns than the first one</td>
</tr>
</table>

void testWhatsAtPos_topmiddle()

<table>
<tr>
<td>Input State:<table><tr><td></td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>X</td><td></td><td></td></tr></table>R=3, C=3</td>
<td>Output State:

Return ‘X’

State of the board is unchanged</td>
<td>Reason:

Testing the top of other columns than the first one</td>
</tr>
</table>

void testWhatsAtPos_topmiddle()

<table>
<tr>
<td>Input State:<table><tr><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>X</td></tr><tr><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>X</td></tr></table>R=3, C=5</td>
<td>Output State:

Return ‘X’

State of the board is unchanged</td>
<td>Reason:

Testing the far top corner of the board</td>
</tr>
</table>

Void PlaceToken(char p, int c)

void testPlaceToken_origin()

<table>
<tr>
<td>Input State:<table><tr><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></tr><tr><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></tr></table>P=X, C=0</td>
<td>Output State:<table><tr><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></tr><tr><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></tr></table></td>
<td>Reason:

Tests that the user can place in the first spot of the board</td>
</tr>
</table>

void testPlaceToken_opencolumn()

<table>
<tr>
<td>Input State:
<table>
<tr><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></tr>
<tr><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></tr>
</table>P=X, C=2</td>
<td>Output State:
<table>
<tr><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></tr>
<tr><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></tr>
</table></td>
<td>Reason:

Tests that the user can place a token in a different open column</td>
</tr>
</table>

void testPlaceToken_difchars()

<table>
<tr>
<td>Input State:
<table>
<tr><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></tr>
<tr><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></tr>
</table>P=O, C=2</td>
<td>Output State:
<table>
<tr><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></tr>
<tr><td></td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
</table></td>
<td>Reason:

Tests that the user can place different chars in the same column</td>
</tr>
</table>

void testPlaceToken_toprow()

<table>
<tr>
<td>Input State:
<table>
<tr><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></tr>
<tr><td></td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
</table>P=O, C=2</td>
<td>Output State:
<table>
<tr><td></td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr>
</table></td>
<td>Reason:

Tests that the user can place a token in the top row</td>
</tr>
</table>

void testPlaceToken_topcorner()

<table>
<tr>
<td>Input State:
<table>
<tr><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></tr>
<tr><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>X</td></tr>
</table>P=X, C=5</td>
<td>Output State:
<table>
<tr><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>X</td></tr>
<tr><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>X</td></tr>
</table></td>
<td>Reason:

Tests that the user can place a token in the far top corner of the board</td>
</tr>
</table>

Boolean CheckIfFree(int c)

void testCheckIfFree_emptyboard()

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

Return true;

State of the board is unchanged</td>
<td>Reason:

Empty board could cause problems</td>
</tr>
</table>

void testCheckIfFree_fullcolumn()

<table>
<tr>
<td>Input State:<table><tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr></table>C=2</td>
<td>Output State:
Return false;
State of the board is unchanged</td>
<td>Reason:
Makes sure one column fills up</td>
</tr>
</table>

void testCheckIfFree_halffull()

<table>
<tr>
<td>Input State:<table><tr><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></tr><tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td><td></td></tr></table>C=2</td>
<td>Output State:
Return true;
State of the board is unchanged</td>
<td>Reason:
Makes sure you can still place if its half full</td>
</tr>
</table>

Boolean CheckHorizWin(int r, int c, char p)

void testCheckHorizWin_bottomrow()

<table>
<tr>
<td>Input State:<table><tr><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></tr><tr><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></tr></table>R=0, C=2, P=X</td>
<td>Output State:
Return true;</td>
<td>Reason:
Basic 3 in a row</td>
</tr>
</table>

void testCheckHorizWin_bottomrowdifchars()

<table>
<tr>
<td>Input State:<table><tr><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></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>X</td><td>O</td><td>X</td><td></td><td></td><td></td></tr></table>R=0, C=2, P=X</td>
<td>Output State:
Return false;</td>
<td>Reason:
3 in row but different chars</td>
</tr>
</table>

void testCheckHorizWin_toprow()

<table>
<tr>
<td>Input State:<table><tr><td>X</td><td>X</td><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td>O</td><td>O</td><td></td><td></td><td></td></tr><tr><td>O</td><td>O</td><td>O</td><td></td><td></td><td></td></tr><tr><td>O</td><td>O</td><td>O</td><td></td><td></td><td></td></tr></table>R=3, C=2, P=X</td>
<td>Output State:
Return true;</td>
<td>Reason:
Makes sure 3 in a row works on the top row</td>
</tr>
</table>

void testCheckHorizWin_middle()

<table>
<tr><td>Input State:
<table><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td>X</td><td>X</td><td></td></tr><tr><td></td><td></td><td>O</td><td>O</td><td>O</td><td></td></tr><tr><td></td><td></td><td>O</td><td>O</td><td>O</td><td></td></tr></table>R=0, C=2, P=X</td><td>Output State:

Return true;</td><td>Reason:

Makes sure it works in the middle of the board</td></tr>
</table>

void testCheckHorizWin_mixed()

<table>
<tr><td>Input State:
<table><tr><td></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></tr><tr><td></td><td>X</td><td>X</td><td>X</td><td></td><td></td></tr><tr><td></td><td>X</td><td>O</td><td>O</td><td></td><td></td></tr></table>R=1, C=3, P=X</td><td>Output State:

Return true;</td><td>Reason:

Pieces are all mixed up</td></tr>
</table>

Boolean CheckVertWin(int r, int c, char p)

void testCheckVertWin_left()

<table>
<tr><td>Input State:
<table><tr><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></tr><tr><td>O</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></tr></table>R=2, C=0, P=O</td><td>Output State:

Return true;</td><td>Reason:

Easy 3 in a row</td></tr>
</table>

void testCheckVertWin_nowin()

<table>
<tr><td>Input State:
<table><tr><td></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></tr><tr><td></td><td>O</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>X</td><td></td><td></td><td></td><td></td></tr></table>R=2, C=1, P=O</td><td>Output State:

Return false;</td><td>Reason:

3 in a row but different chars</td></tr>
</table>

void testCheckVertWin_right()

<table>
<tr><td>Input State:
<table><tr><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>O</td></tr><tr><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>X</td></tr></table>R=3, C=5, P=O</td><td>Output State:

Return true;</td><td>Reason:

Makes sure 3 in a row works up in the top</td></tr>
</table>

void testCheckVertWin_middle()

<table>
<tr><td>Input State:
<table>
<tr><td></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></tr>
<tr><td></td><td>O</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>O</td><td></td><td></td><td></td><td></td></tr>
</table>
R=2, C=1, P=O</td><td>Output State:

Return true;</td><td>Reason:

3 in a row works in the middle of the board</td></tr>
</table>

void testCheckVertWin_horiz()

<table>
<tr><td>Input State:
<table>
<tr><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></tr>
<tr><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></tr>
</table>
R=0, C=2, P=X</td><td>Output State:

Return false;</td><td>Reason:

Makes sure a horizontal win doesn’t trip it</td></tr>
</table>

Boolean CheckDiagWin(int r, int c, char p)

void testCheckDiagWin_lefttoright()

<table>
<tr><td>Input State:
<table>
<tr><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></tr>
<tr><td></td><td>X</td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>X</td><td>O</td><td>O</td><td></td><td></td><td></td></tr>
</table>
R=2, C=2, P=X</td><td>Output State:

Return true;</td><td>Reason:

Basic 3 in a row</td></tr>
</table>

void testCheckDiagWin_righttoleft()

<table>
<tr><td>Input State:
<table>
<tr><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></tr>
<tr><td>X</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>O</td><td>O</td><td>X</td><td></td><td></td><td></td></tr>
</table>
R=2, C=0, P=X</td><td>Output State:

Return true;</td><td>Reason:

3 in a row in the other direction</td></tr>
</table>

void testCheckDiagWin_horiz()

<table>
<tr><td>Input State:
<table>
<tr><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></tr>
<tr><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></tr>
</table>
R=0, C=2, P=X</td><td>Output State:

Return false;</td><td>Reason:

Makes sure horizontal win doesn’t trip it</td></tr>
</table>

void testCheckDiagWin_vert()

<table>
<tr><td>Input State:<table><tr><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></tr><tr><td>X</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></tr></table>R=0, C=2, P=X</td><td>Output State:

Return false;</td><td>Reason:

Makes sure vertical win doesn’t trip it</td></tr>
</table>

Boolean CheckTie()

void testCheckTie_notie()

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

Return false;</td><td>Reason:

Random board situation</td></tr>
</table>

void testCheckTie_fullboard()

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

Return true;</td><td>Reason:

Board completely full</td></tr>
</table>

void testCheckTie_onecolfull()

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

Return false;</td><td>Reason:

Makes sure one column being full doesn’t trip it</td></tr>
</table>

void testCheckTie_onerowfull()

<table>
<tr><td>Input State:<table><tr><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></tr><tr><td></td><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>X</td><td>X</td></tr></table></td><td>Output State:

Return false;</td><td>Reason:

Makes sure a whole row being full doesn’t trip it</td></tr>
</table>

Deployment:

- Type make to compile the program
- Type make run to run the program