CPSC 2150 Project Report

Yash Patel

Requirements Analysis

Functional Requirements:

- 1. As a player, I need to place a marker so that I can indicate my turn
- 2. As a player, I want to restart the game after drawing so I can continue playing.
- 3. As a player, I need to alternate players so that I can have each of them have a turn playing.
- 4. As a programmer, I need to have a computer to play with, so I can view and test my game.
- 5. As a player, I can enter the number of rows and columns between 3 20 so I can customize the board
- 6. As a player, I need to be able to place my marker anywhere on a [num rows] x [num columns] grid, so then I can use my turn.
- 7. As a gamescreen, I need to know when all the spaces are filled with no winning side so that it can display the draw message.
- 8. As a player, I want to be able to quit the game so that I can stop playing
- 9. As a player, I need to be player 'X' if I am first so I can start game
- 10. As a gameboard, I need to place either 'X' or 'O', so I know the game pieces
- 11. As a gameboard, I need to check to see if the row/col is valid so that I can place a marker there
- 12. As a gameboard, I need to know if 5 of the same markers are placed in the same column to count as a vertical win
- 13. As a gameboard, I need to know if 5 of the same markers are placed in the same row to count as a horizontal win
- 14. As a gameboard, I need to know if 5 of the same markers are placed in a diagonal line to count as a diagonal win
- 15. As a player, I should be able to change the number of players in a game from 2-10 so I can play with more people

Non-Functional Requirements

- 1. The program is written in java
- 2. The program is played using a GUI

- 3. The program must have at least two players and at max 10
- 4. The program must be developed in IntelliJ
- 5. The player must know the rules of tic-tac-toe
- 6. the program must have a [num rows] x [num columns] board, meaning the user can choose the size from 3 to 20
- 7. The 'X' Player must go first

Deployment Instructions

Details in Projects 2-5.

System Design

Class 1: GameScreen

Class diagram

GamesScreen
-Column: int [1] -Row: int[1]
+ main(String[]): static void

Class 2: BoardPosition

Class diagram

BoardPosition - Row: int[1] - Column: int[1] + boardPosition(int, int): void + getRow():int + getCol():int + equals(BoardPosition): boolean +toString(): void

Class 3: GameBoard

Class diagram

GameBoard

- PlayerChar: char[*] {'X', 'O'}
- Board: char[*][*]
- + GameBoard(): void
- + checkSpace(BoardPosition): boolean
- + placeMarker(BoardPosition, char): void
- + checkForWinner(BoardPosition): boolean
- + checkForDraw(): boolean
- + checkHorizontalWin(BoardPosition, char): boolean
- + checkVerticalWin(BoardPosition, char): boolean
- + checkDiagonalWin(BoardPosition, char): boolean
- + whatsAtPos(BoardPosition): char
- + isPlayerAtPos(BoardPosition, char): boolean
- + toString():string

Class 4: IGameBoard

Class diagram

IGameBoard

- -rowNum: int[1]
- -colNum: int[1]
- -winCount: int[1]
- + getNumRows(): int <default>
- + getNumColumns(): int <default>
- + getNumToWin(): int <default>
- + checkSpace(BoardPosition): boolean <default>
- + placeMarker(BoardPosition, char): void
- + checkForWinner(BoardPosition): boolean <default>
- + checkVerticalWin(BoardPosition): boolean <default>
- + checkHorizontalWin(BoardPosition): boolean <default>
- + checkDiagonalWin(BoardPosition): boolean <default>
- + checkForDraw(): boolean <default>
- + whatsAtPos(boardPosition): char
- + isPlayerAtPos(boardPosition, char): boolean <default>

Class 5: AbsGameBoard

Class diagram

AbsGameBoard

+ toString(): string

Class 6: GameBoardMem

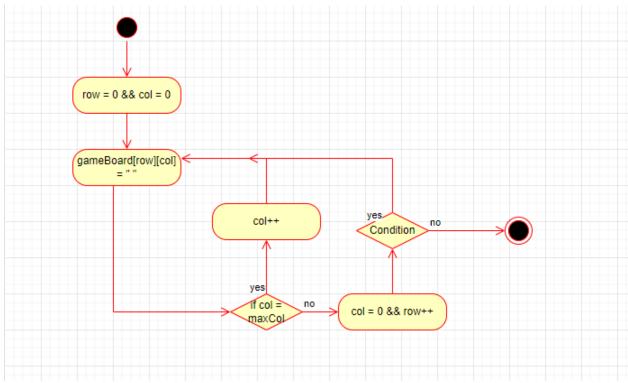
Class diagram

GameBoardMem

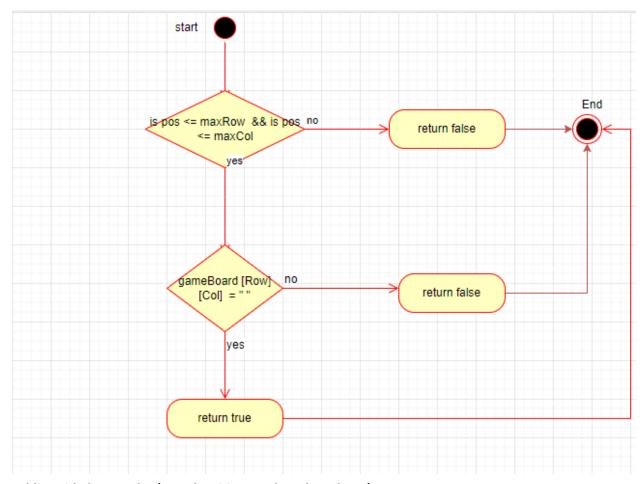
- Map<Character, List<BoardPosition>> board: board
- playerChar: char
- winScore: int
- row: int
- col: int
- + GameBoardMem(int,int,int)
- + placeMarker(BoardPosition,char): void
- + whatsAtPos(BoardPosition): char
- + isPlayerAtPos(BoardPosition, char): boolean
- + getNumRows(): int
- + getNumColumns(): int
- +getNumToWIn(): int

Activity diagrams

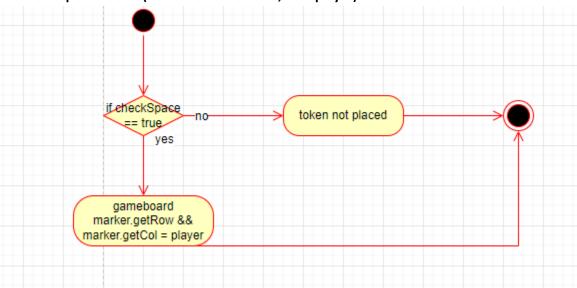
Gameboard(Constructor):



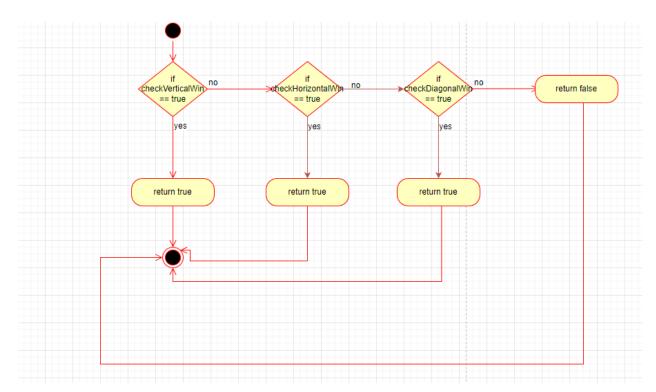
public boolean checkSpace(BoardPosition pos):



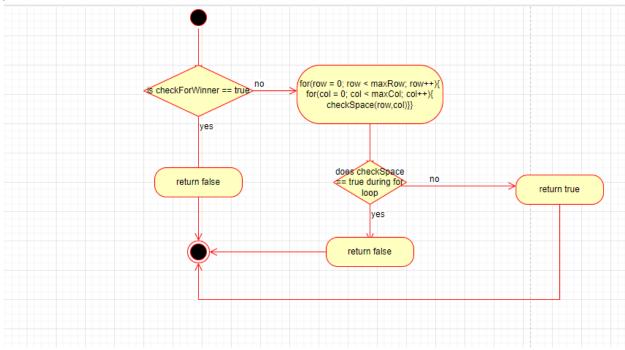
public void placeMarker(BoardPosition marker, char player):



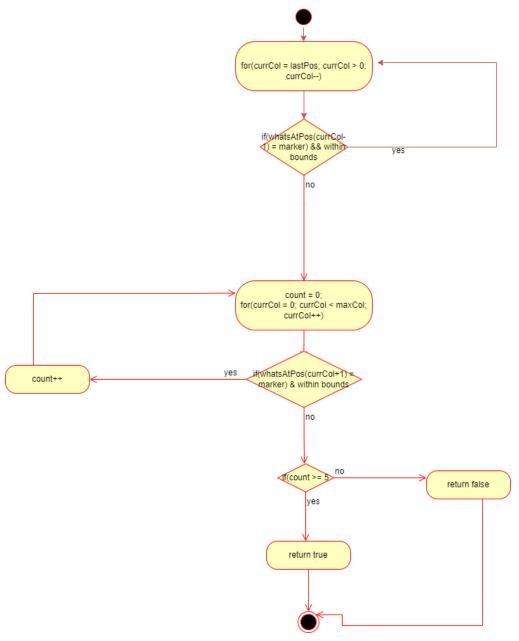
public boolean checkForWinner(BoardPosition lastPos):



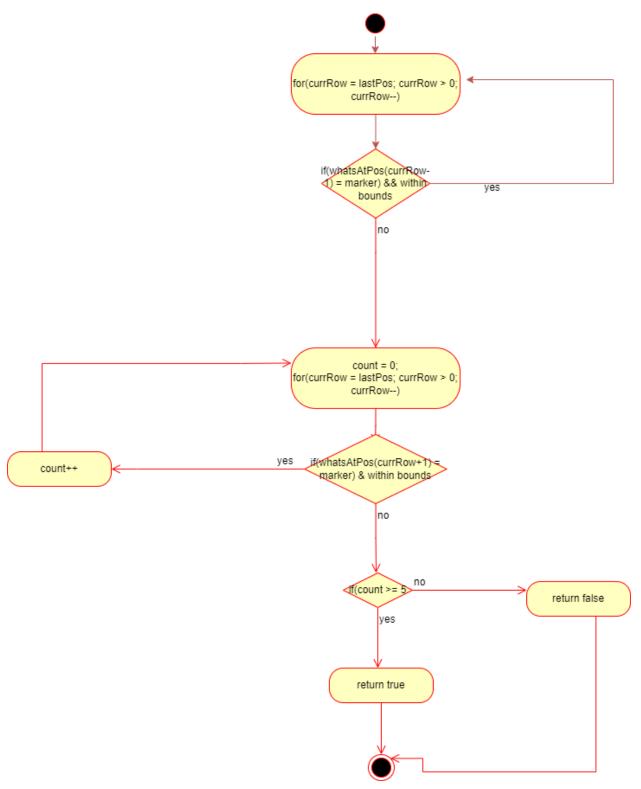
public boolean checkForDraw():



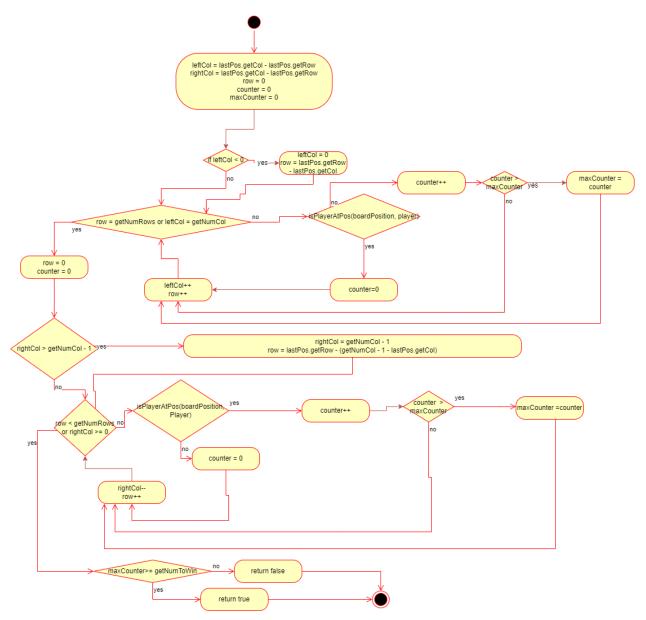
public boolean checkHorizontalWin(BoardPosition lastPos, char player):



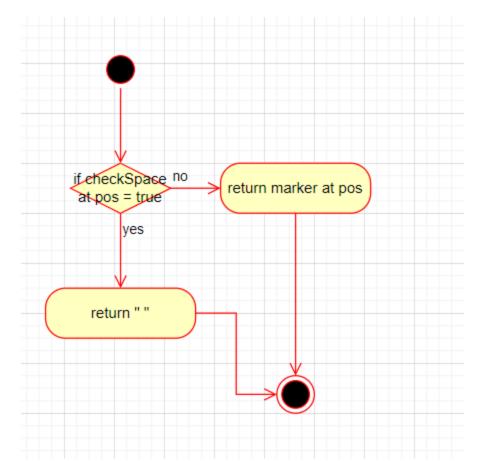
public boolean checkVerticalWin(BoardPosition lastPos, char player):



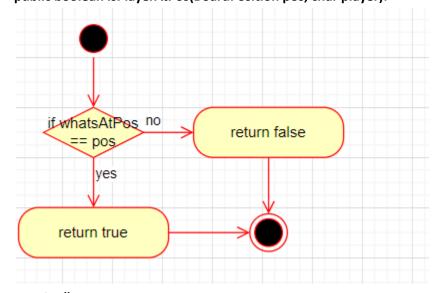
public boolean checkDiagonalWin(BoardPosition lastPos, char player):



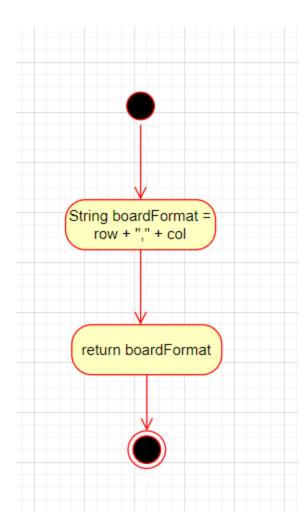
public char whatsAtPos(BoardPosition pos):



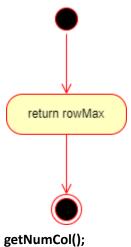
public boolean isPlayerAtPos(BoardPosition pos, char player):

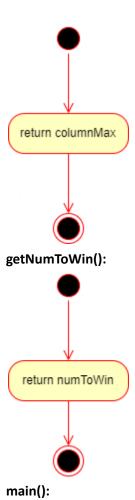


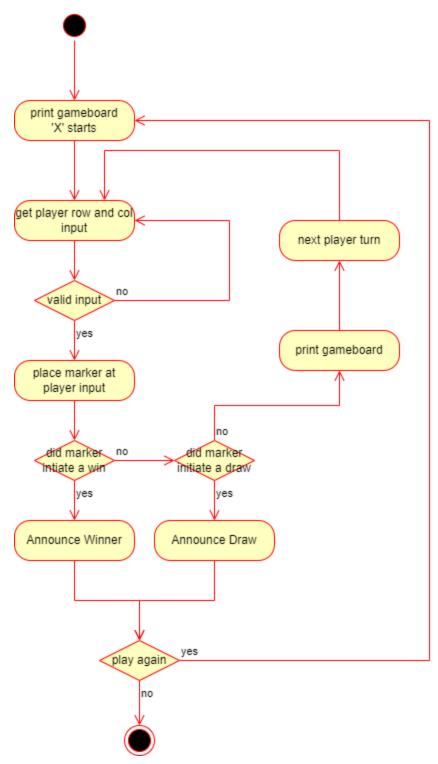
toString():



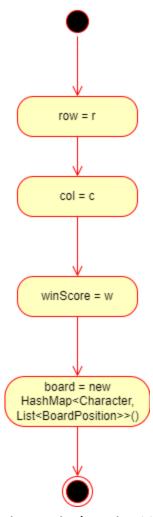
getNumRows():



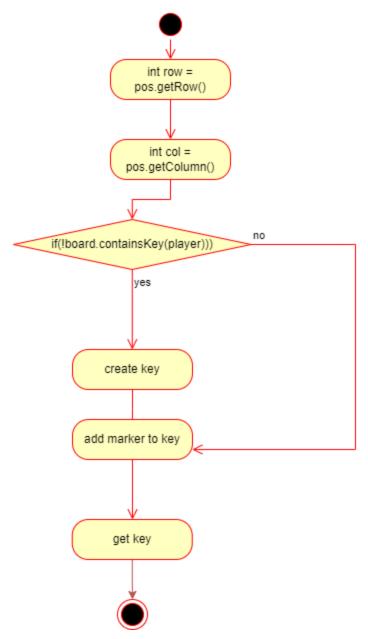




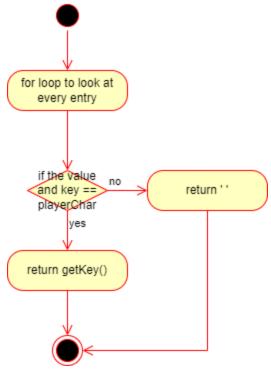
GameBoardMem(int r, int c, int w):



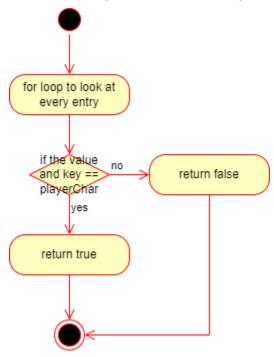
placeMarker(BoardPosition pos, char Player):



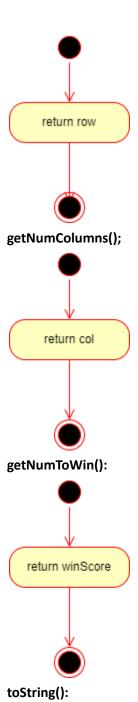
whatsAtPos(BoardPosition pos):

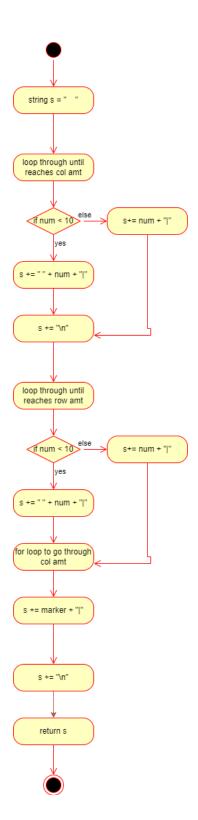


@Override isPlayerPos(BoardPosition pos, char playerChar):



getNumRows():





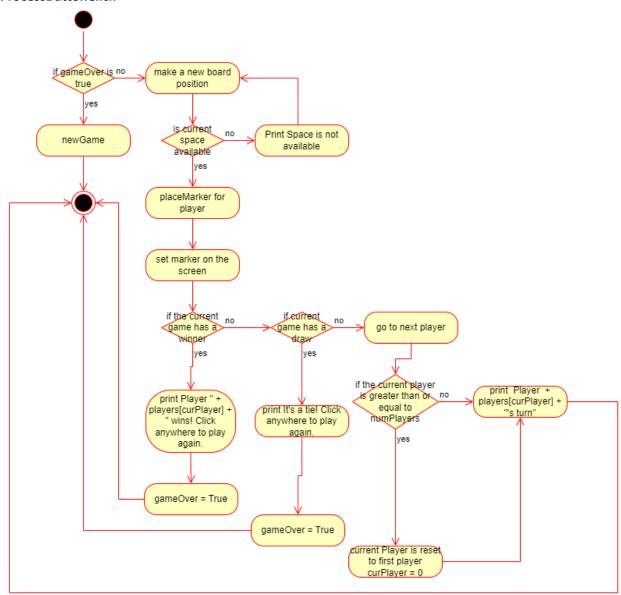
Class 7: TicTacToeController Class Diagram

TicTacToeController

- curGame: IGameBoard[1]
- screen: TicTacToeView[1]
- + MAX_Players; Static final int[1]
- numPlayers: int [1]
- players: char[10]
- curPlayer: int [1]
- gameOver: boolean[1]
- + TicTacToeController(IGameBoard, TicTacToeView, int)
- + processButtonClick(int, int) : void
- newGame(): void

Activity Diagram:

ProcessButtonClick



How To Run:

make: compiles the program make run: runs the entire program make clean: deletes all the .class files

make test: complies the program with the test files

make testGB: runs and uses the test file only for the Gamboard class

make testGBmem: runs and uses the test file only for the GameboardMem class.

Test Cases

Details in Project 4.

Constructor:

Input: row = 3	Outp State:							Reason: This test is unique and distinct
col = 3 numToWin = 3		0)	1		2	- 1	ecause it tests the minimum mount of board requirements
	0						n	needed for a table
	1							estGameBoard_min_input
	2							
	getNu getNu getNu	ımCo	lumn	s = 3				
input: row = 5	Outp	ut:						Reason: This test is unique and distinct
col = 5 numToWin = 5		0	1	2	3	4	b	pecause it tests bounds of valid ralues thats not the min or max
	0						F	stGameBoard_valid
	1						"	estGameBoard_valid
	2							
	3							
	4							
	getNu getNu getNu	ımCo	lumn	s = 5				

input: Output: Reason: row = 100 This test is unique and distinct col = 100 because it tests the maximum 0 1 2 99 numToWin = 25 amount of board requirements needed for a table 0 **Function name:** testGameBoard_max_input 1 2 99 getNumRows = 100 getNumColumns = 100 getNumToWin = 25

checkSpace:

0 1 2 pos.ge	3 to win 0 tRow = 0 tCol = 2	1 0	2	Output: checkspace = true state of the board is unchanged	Reason: This test is unique and distinct because it is checking a space on an unpopulated board Function name: testCheckSpace_empty
Input: State:	3 to win	l		Output: checkSpace = false	Reason: This space is unique because it
	0	1	2	State: unchanged	is in a spot where a marker is out of bounds
0	х		0		Function: testCheckSpace_bounds
1	х	Х	0		testerieckspace_bourius
2	0				
	tRow = 5		,		

Input: State: 3	3 to win			Output: checkSpace = false	Reason: This space is unique because the
	0	1	2	State: unchanged	board is already full Function :
0	х	0	х		testCheckSpace_full
1	О	Х	0		
2	Х	Х	0		
	tRow = 2 tCol = 1		•		

checkHorizontalWin:

Input State	t: : 4 to	win				Output: checkHorizontalWin = true	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the last X was placed in the middle of the
0							string of 4 consecutive X's as opposed to on the end, so
1							the function needs to count X's on the right and left
2	х	Х	х	х			
3	0	0	0	Х	0		Function: testCheckHorizontalWin_
4							_Middle
	getRov getCol = 'X'		,				

Inpu State	t: e: 4 to	win				Output: checkHorizontalWin = true	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the last X was placed at the end of the string
0							of 4 consecutive X's, so the function needs to count X's on
1							the left
2	Х	Х	Х	Х			Function:
3	0	0	0	Х	0		testCheckHorizontalWin_End
4							
Inpu State	e: 4 to		1_	T_		Output: checkHorizontalWin = False state is unchanged	Reason: This test case is unique and distinct because the last X was
State	0	win 1	2	3	4	state is unchanged	distinct because the last X was
0							placed at a spot that would not give it a win, thus resulting in a
1							false value Function:
2	Х	Х	Х				testCheckHorizontalWin_No_W n
3	0	0	0	Х			
4							
pos.	getRo	w = 3	;		1		

Inpu State	t: e: 4 to	win				Output: checkHorizontalWin = true	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the last X was placed at the start of the string
0							of 4 consecutive X's, so the function needs to count X's on
1							the right
2	Х	Х	Х	х			Function:
3	0	0	0	х	О		testCheckHorizontalWin_start
4							
	getRo getCo = 'X'		;	-			

checkVerticalWin:

Inpu ^s State	t: :: 4 to	win				Output: checkVerticalWin = true	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the last marker was at the end of the string of 4
0	Х	0					consecutive X's, so the function needs to count X's from the
1	Х	0					bottom to the top.
2	Х	0					testCheckVerticalWin_end
3	Х						
4							
1 '	getRov getCol = 'X'		;				

State: 4 to win	Output: checkVerticalWin = true	Reason: This test case is unique and
0 1 2 3 4	state is unchanged	distinct because the last marker was at the start of the string of 4
0 X O		consecutive X's, so the function needs to count X's from the top
1 X O		to the bottom.
2 X O		testCheckVerticalWin_start
3 X		
4		
pos.getCol = 0; pos = 'X' Input: State: 4 to win	Output: checkVerticalWin = true	Reason: This test case is unique and
0 1 2 3 4	state is unchanged	distinct because the last marker was at the middle of the string
0 X O		of 4 consecutive X's, so the function needs to count X's from
1 X O		both the top to the bottom and
2 V 0		bottom to top. Function:
2 X O		I toctChock\/ortical\\/in mid
3 X		testCheckVerticalWin_mid

pos = 'X'

Inpu State	t: e: 4 to	win				Output: checkVerticalWin = false
	0	1	2	3	4	state is unchanged
0	Х	0				
1	Х	0				
2		0				
3	Х					
4	Х					
-	getCo	w = 4; l = 0;		•		

Reason:This test case is uniq

This test case is unique and distinct because the last marker was at the end of the string but also does not out put 4 consecutive markers for the string so it would result in a false

Function:

testCheckVerticalWin_No_Win

CheckDiagonalWin:

Inpu State	t: e: 4 to	win				Output: checkDiagonalWin = true	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the last X was placed at the start of the string
0							of 4 diagonal X's, so the function needs to count from
1		Х					the last placed marker, then
2			Х				going down and to the right
3	0	0	0	х			Function: testCheckDiagonalWin_start
4					х		
	getRov getCol = 'X'						

I <mark>npu</mark> State	t: e: 4 to	win				Output: checkDiagonalWin = true	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the last X was placed at the end of the string
0							of 4 diagonal X's, so the function needs to count from
1		Х					the last placed marker, then going up and to the left
2			Х				
3	0	О	0	Х			Function: testCheckDiagonalWin_end
4			0		x		
	s.getCol = 4; s = 'X' out: te: 4 to win						
Inpu	t:					Output:	Reason:
Inpu	t: e: 4 to],]2	T ₄]	Output: checkDiagonalWin = true state is unchanged	Reason: This test case is unique and distinct because the last X was
I npu State	t:	win 1	2	3	4	checkDiagonalWin = true	This test case is unique and distinct because the last X was placed at the middle of the
Inpu	t: e: 4 to		2	3	4	checkDiagonalWin = true	This test case is unique and distinct because the last X was placed at the middle of the string of 4 diagonal X's, so the function needs to count from
Inpu State	t: e: 4 to		2	3	4	checkDiagonalWin = true	This test case is unique and distinct because the last X was placed at the middle of the string of 4 diagonal X's, so the function needs to count from the last placed marker, then
Inpu State	t: e: 4 to	1	2 X	3	4	checkDiagonalWin = true	This test case is unique and distinct because the last X was placed at the middle of the string of 4 diagonal X's, so the function needs to count from
Inpu State	t: e: 4 to	1		3 X	4	checkDiagonalWin = true	This test case is unique and distinct because the last X was placed at the middle of the string of 4 diagonal X's, so the function needs to count from the last placed marker, then both going down and to the
O 1	t: 0	1 X	x		4 X	checkDiagonalWin = true	This test case is unique and distinct because the last X was placed at the middle of the string of 4 diagonal X's, so the function needs to count from the last placed marker, then both going down and to the right and up and to the left

Inpu State	t: e: 4 to	win				Output: checkDiagonalWin = true	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the last X was placed at the start of the string
0							of 4 diagonal X's, so the function needs to count from
1				Х			the last placed marker, then going down and to the left
2			х				
3		Х		0	О		Function: testCheckDiagonalWin_start_op
4	Х			0	0		posite
pos : Inpu State	t:	win				Output: checkDiagonalWin = true	Reason: This test case is unique and
•	e: 4 to	win	ı			checkDiagonalWin = true state is unchanged	This test case is unique and distinct because the last X was
	0	1	2	3	4	- Same in annual Bear	placed at the end of the string
0							of 4 diagonal X's, so the function needs to count from
1				х			the last placed marker, then going up and to the right
2			х				
3		Х		0	О		Function: testCheckDiagonalWin_end_op
4	х			0	0		posite
						1	I

pos.getCol = 0; pos = 'X'

Innu	<u> </u>					Outmut	Pagani
Inpu	:: : 4 to	win				Output: checkDiagonalWin = true	Reason: This test case is unique and
State	0	1	2	3	4	state is unchanged	distinct because the last X was placed at the middle of the
0							string of 4 diagonal X's, so the function needs to count from
1				Х			the last placed marker, then both going down and to the left
2			Х				and up and to the right
3		Х		0	0		Function:
4	х			0	О		testCheckDiagonalWin_middle_ opposite
	getRov getCol : 'X'		,				
Inpu State	t: : 4 to	win				Output: checkDiagonalWin = false	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the last marker was at the end of the string but
0					Х		also does not out put 4 consecutive markers for the
1				х			string so it would result in a false
2							Function:
3		Х		0	0		testCheckDiagonalWin_No_Win
4	Х			0	0		
	etRov getCol		;				

checkForDraw:

<mark>Input</mark> State	t: : 4 to	win				Output: checkForDraw = false	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the board is empty, so the function only
0							would result in a false value Function name:
1							testCheckForDraw_empty
2							
3							
4							
Input State	t: : 4 to	win				Output: checkForDraw = true	Reason: This test case is unique and
	0	1	2	3	4	state is unchanged	distinct because the board is fu and there is no win condition
0	Х	О	х	О	х		through either marker, so the function only would result in a
1	0	х	О	х	0		true value Function name:
2	0	Х	Х	0	Х		testCheckForDraw_full
3	Х	0	Х	0	х		
4	0	х	0	х	0		
	•			•	•		
Input						Output:	Reason:
State	: 4 to	win 1	2	3	4	checkForDraw = false state is unchanged	This test case is unique and distinct because the board's fire
0		0	X	0	X		position is empty, so the function would result in a false
				x	0		value. Function name:
2	0	X	O X	0	Х		testCheckForDraw_first_empty
3			X		X		
3	X	О		0			
4	0	ı X	0	X	0		

	nput tate	t: : 4 to	win				Output: checkForDraw = false	Reason: This test case is unique and
		0	1	2	3	4	state is unchanged	distinct because the boards full but has player X won, so the
	0	Х	0	Х	0	Х		function would result in a false value.
	1	0	х	0	х	0		Function name:
	2	0	Х	Х	0	х		testCheckForDraw_Full_Win
	3	Х	Х	0	0	Х		
	4	0	х	0	х	Х		
p	_	etCol	w = 3; = 1;	;				

whatsAtPos:

Input State	t: : 4 to	win				Output: wharsAtPos = ' '	Reason: This test case is unique
	0	1	2	3	4	state is unchanged	and distinct because the board is empty, which will
0							make the function return a space.
1							Function:
2							testWhatsAtPos_empty
3							
4							
1 '	getRov getCol						

Inpu State	i t: e: 4 to	win				Output: wharsAtPos = ' '	Reason: This test case is unique			
	0	1	2	3	4	state is unchanged	and distinct because the space is next to a filled column, which will make the			
0	Х						function return a space.			
1	0	-		-			Function: testWhatsAtPos_adjacent_spa			
2	X	_		_			e			
3	0									
4	X									
	getRo getCo									
nput: State: 4 to win						Output: wharsAtPos = ' '	Reason: This test case is unique			
	0	1	2	3	4 state is unchanged	and distinct because the board is filled except one,				
0		0	Х	0	Х		which will make the function return a space. Function: testWhatsAtPos_single_space			
1	0	Х	0	Х	О					
2	0	Х	Х	0	Х		testwilatsAtros_sliigle_space			
3	Х	0	Х	0	Х					
4	0	х	0	х	Х					
	getRo getCo		;							
I npu State	i t: e: 4 to	win				Output: wharsAtPos = 'X '	Reason: This test case is unique			
	0	1	2	3	4	state is unchanged	and distinct because the board is empty except for			
0							one, which will make the function return the marker.			
1		Х					Function:			
2							testWhatsAtPos_Marker			
3										
4										
4 pos.	getRo getCo									

Input State	t: : 4 to	win				Output: wharsAtPos = 'O '	Reason: This test case is unique
	0	1	2	3	4	state is unchanged	and distinct because the space on the board is filled,
0	0	0	Х	0	х		which will make the function return the marker.
1	0	Х	0	Х	0		Function: testWhatsAtPos full
2	0	Х	Х	0	х		testwildtsAtPos_luli
3	Х	0	Х	0	х		
4	0	Х	0	Х	х		
1	getRov getCol						

isPlayerAtPos:

Inpu State	t: e: 4 to	win				Output: isPlayerAtPos = false	Reason: This test case is unique
	0	1	2	3	4	state is unchanged	and distinct because the board is empty, which will
0							test to see if a single marker is in an empty board.
1							Function:
2							testIsPlayerAtPos_empty
3							
4							
	getRo getCo = 'X'		•	•			

I <mark>npu</mark> State	i t: e: 4 to	win				Output: isPlayerAtPos = true	Reason: This test case is unique
	0	1	2	3	4	state is unchanged	and distinct because the board will test to see if a
0							single marker is in position Function:
1		Х					testIsPlayerAtPos_Marker
2							
3							
4							
	= 'X'						
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,							
Inpu		win				Output: isPlayerAtPos = true	Reason: This test case is unique
Inpu	t:	win 1	2	3	4	-	
Inpu	t: e: 4 to		2	3	4	isPlayerAtPos = true	This test case is unique and distinct because the board will test to see if the function can differentiate
Inpu State	t: e: 4 to		2	3	4	isPlayerAtPos = true	This test case is unique and distinct because the board will test to see if the function can differentiate between two different markers Function :
Inpu State	t: e: 4 to	1	2	3	4	isPlayerAtPos = true	This test case is unique and distinct because the board will test to see if the function can differentiate between two different markers
Inpu State	t: e: 4 to	1	2		4	isPlayerAtPos = true	This test case is unique and distinct because the board will test to see if the function can differentiate between two different markers Function :
O 1	t: e: 4 to	1	2		4	isPlayerAtPos = true	This test case is unique and distinct because the board will test to see if the function can differentiate between two different markers Function :

							T
Input						Output:	Reason:
State	: 4 to	win				isPlayerAtPos = false state is unchanged	This test case is unique and distinct because
	0	1	2	3	4	state is unchanged	the pos is different and marker
0							is already taken by a different marker
1		Х					Function: testIsPlayerAtPos_Two_Markers
2				О			_False
3							
4							
	getRov getCol : 'X'						
Input State	t: : 5 to	win				Output: isPlayerAtPos = true	Reason: This test case is unique
	0	1	2	3	4	state is unchanged	and distinct because the pos is filled and the board is
0	А	S	D	F	G		filled Function:
1	S	D	F	G	А		testIsPlayerAtPos_filled
2	D	F	G	А	S		
3	F	G	Α	S	D		
4	Α	S	D	F	G		
1	etRov getCol : 'S'		•	•			

placeMarker:

Input State:	: : 4 to v	vin			Outp State		_			Reason: This test case is unique and
	0	1	2	3		0	1	2	3	distinct because I am placing a marker representing a player
0					0					who has not been placed on thi board before.
1					1			А		Function:
2		Х			2		Х			testPlaceMarker_col_not_emp
3	О				3	0				У
	etRow etCol = ,									
Input State:	: : 4 to v	vin			Outp State				_	Reason: This test case is unique and
	0	1	2	3		0	1	2	3	distinct because I am placing a marker representing a player
0					0					when the board is empty.
1					1		Х			Function: testPlaceMarker_empty
2					2					testriaceiviarkei_empty
3					3					
-	etRow etCol = .'									
Input State:	: : 4 to v	vin			Outp State					Reason: This test case is unique and
	0	1	2	3		0	1	2	3	distinct because I am placing a marker representing a player
0	х	0	Х	0	0	х	0	Х	0	when the board is filled except one space.
1	О	Х		Х	1	0	Х	0	Х	Function:
2	х	0	Х	0	2	Х	0	Х	0	testPlaceMarker_single_space
3	0	Х	Х	О	3	0	Х	Х	0	
-	etRow etCol =)'		<u> </u>	· · ·				<u> </u>	ļ	

Input State	t: : 4 to w	vin				put: te is unc	hanged	d		Reason: This test case is unique and
	0	1	2	3			J		distinct because I am placing a marker in the corner	
0	Х									Function:
1										testPlaceMarker_Corner
2										
3]					
p = 'X Input	t:					:put:			Reason:	
State	: 4 to v	1	2	3		State: 0 1 2 3				This test case is unique and distinct because I am placing a
0					0					marker representing a player who has been placed on this
1					1			х		board before.
2		х			2		Х			Function: testPlaceMarker_same_marke
3	0				3	0				
	getRow getCol = ('		•	•		•	•		•	'