CPSC 2150 Project Report

Abigail Barrett

Requirements Analysis

Functional Requirements:

- 1. As a player I can view the current game board on my turn, so that I know which spaces I can put my token in.
- 2. As a player I can view which player's turn it is, so that I know which player is adding the token.
- 3. As a player I can select the row to add my token to so that I can add my token to that position.
- 4. As a player I can select the column to add my token to so that I can add my token to that position.
- 5. As a player I can select a position with a token already in it, so I can see if I can place my token in that position.
- 6. As a player I can select a position not on the grid, so I can see if I can place my token in that position.
- 7. As a player I can place number_to_win of my markers in a row horizontally, so I can win the game.
- 8. As a player I can place number to win of my markers in a row vertically, so I can win the game.
- 9. As a player I can place number_to_win of my markers in a row diagonally, so I can win the game.
- 10. As a player I can select an empty position on the grid, so I can place my token in that position
- 11. As a player I can view that I made an error so that I know not to select that position.
- 12. As a player I can select another position after I made an error so that I can play a token in a valid position on the board.
- 13. As a player I can view when the game is over, so that I know when the game has finished.
- 14. As a player I can view which player won the game, so that I know who won the game.
- 15. As a player I can view the game board when the game is over, so I know what positions my opponent and I put our tokens in.
- 16. As a player I can view when the game is a draw, so I know no one won the game.
- 17. As a player I can choose to play the game again so that I can play another game.
- 18. As a player I can exit the program so that I cannot play another game.
- 19. As a player I can specify the number_of_rows, so that I can choose the size of my board.
- 20. As a player I can specify the number_of_columns, so that I can choose the size of my board.
- 21. As a player I can specify the number_to_win, so that I can choose how many tokens it takes to win the game.
- 22. As a player I can choose how many players will play the game, so that I can play with more players.
- 23. As a player I can choose the token of each player playing the game, so I can customize my token.
- 24. As a player I can view when I entered an invalid number_to_win, so I can choose a number_to_win that will make the game work correctly.
- 25. As a player I can view when I entered an invalid number_of_rows, so I can choose a number_of_rows that will make the game work correctly.

26. As a player I can view when I entered an invalid number_of_columns, so I can choose a number_of_columns that will make the game work correctly.

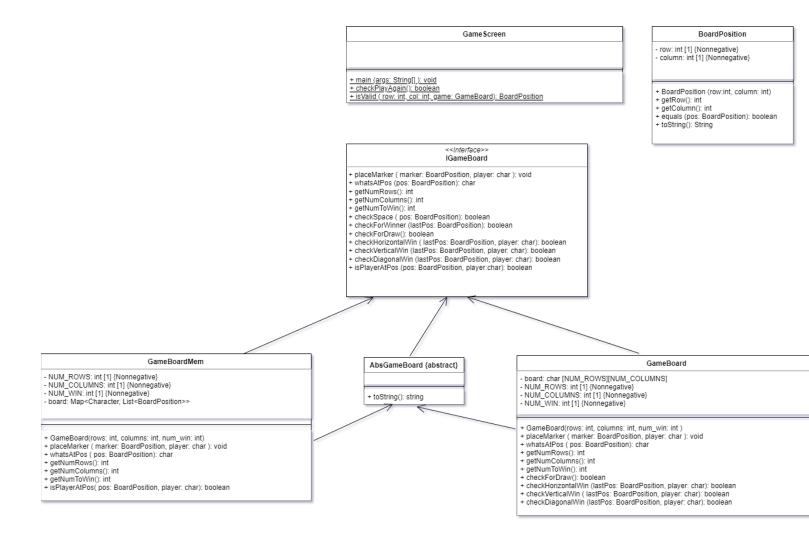
Non-Functional Requirements

- 1. The system must be written in Java.
- 2. The system should run on unix.
- 3. The system should run without the user noticing any delays.
- 4. The system should display an easily readable GUI to the user.
- 5. The system should use memory implementation based on the GUI requirements.
- 6. The system should not crash when run
- 7. The system can run with either a fast implementation or a memory efficient implementation
- 8. If fast implementation, the board is of size user_entered_number_of_rows x user_entered_number_of_columns
- 9. If memory efficient implementation, the board has no allocated memory until user enters value
- 10. Player order of play goes in order of player tokens (provided in Controller)
- 11. Player order rotates in the same order of play

Deployment Instructions

Details in Projects 2-5.

System Design

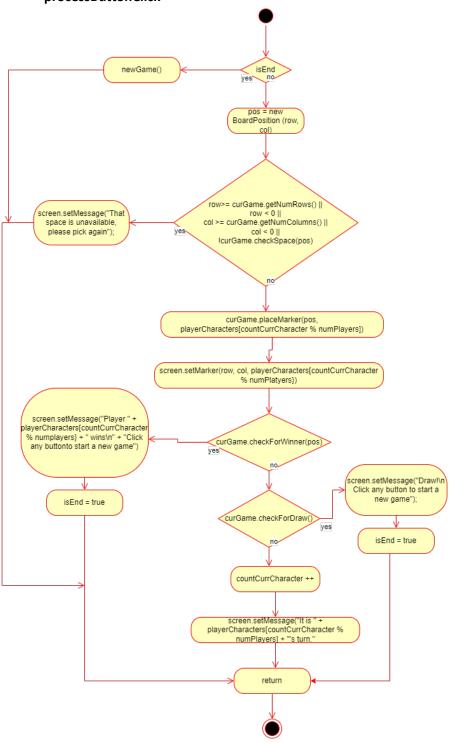


TicTacToeController

- POSSIBLE_PLAYER_CHARS: char[10]
- playerCharacters: char[]
- countCurrCharacter: int [1] {Nonnegative}
- pos: BoardPosition
 isEnd: boolean
- curGame: IGameBoard
 screen: TicTacToeView
 MAX_PLAYERS: int
- + TicTacToeController(model: IGameBoard, view: TicTacToeView, np: int)
- + processButtonClick(row: int, col: int): void
- newGame(): void

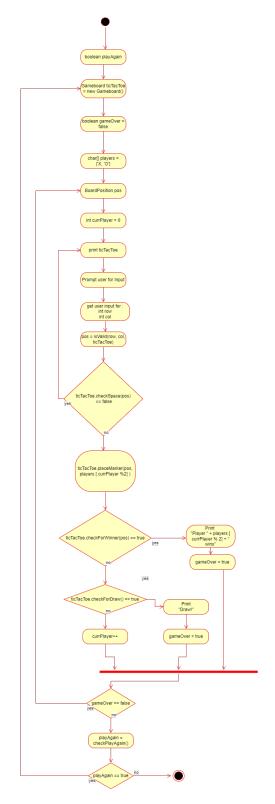
TicTacToeController:

process Button Click

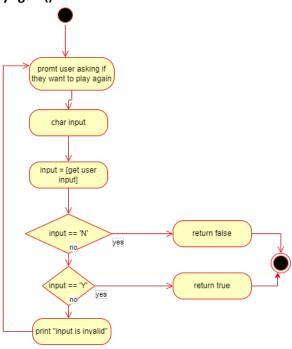


GameScreen:

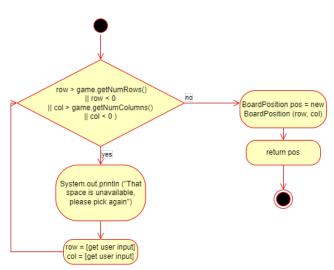




checkPlayAgain()

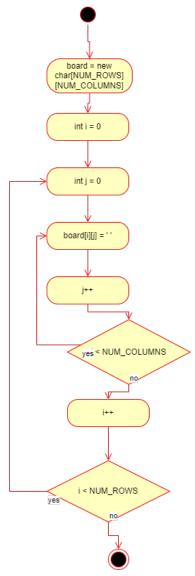


isValid

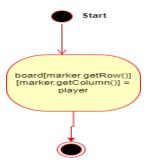


GameBoard:

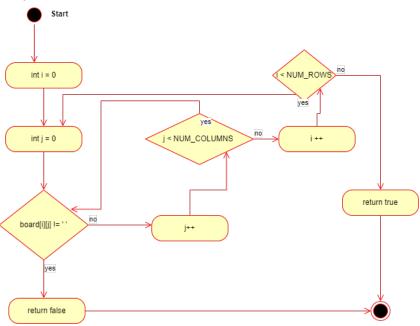
Constructor



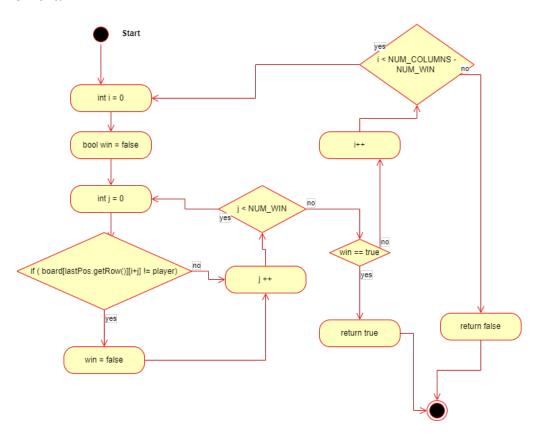
Place Marker



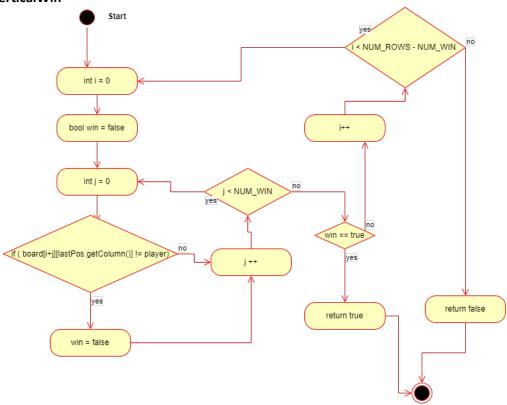
CheckForDraw



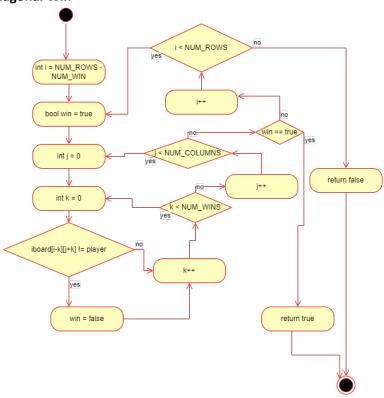
CheckHorizontalWin



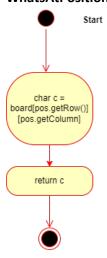
checkVerticalWin



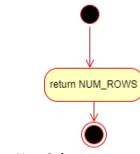
checkDiagonal Win



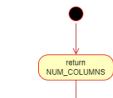
Whats At Position



${\sf getNumRows}$



getNumColumns

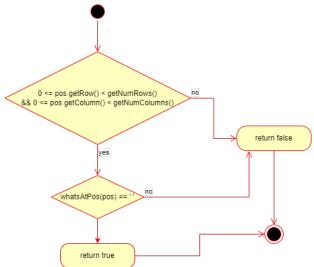


${\sf getNumToWin}$

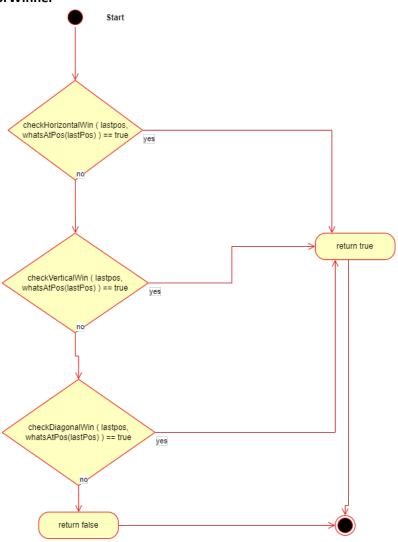


IGameBoard:

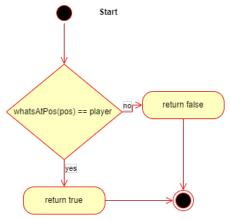
checkSpace



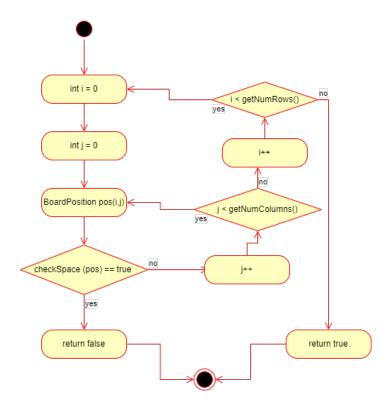
check For Winner



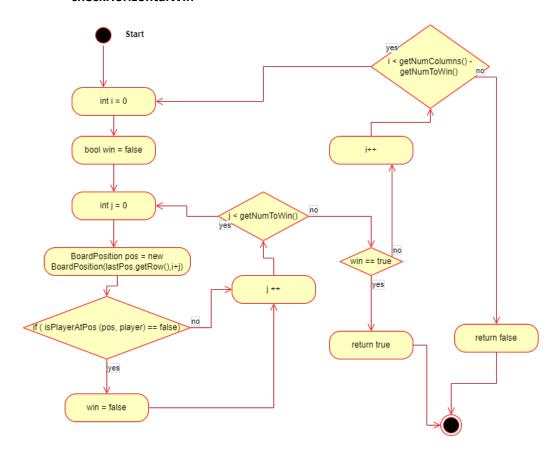
isPlayerAtPos



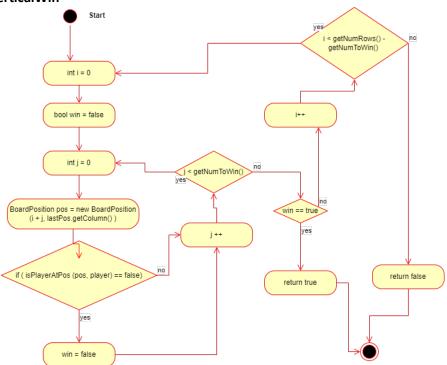
checkForDraw



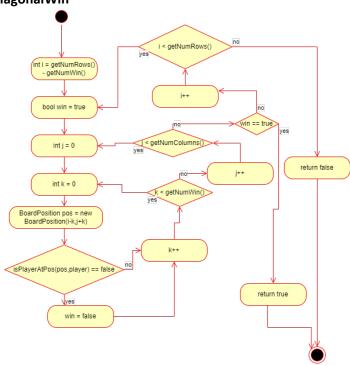
checkHorizontalWin



checkVerticalWin

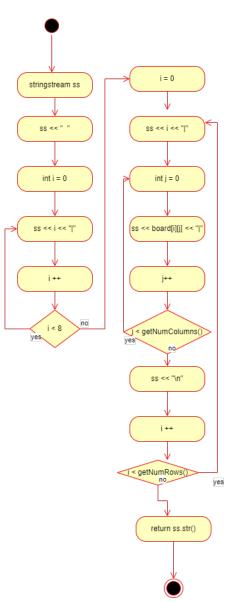


checkDiagonalWin



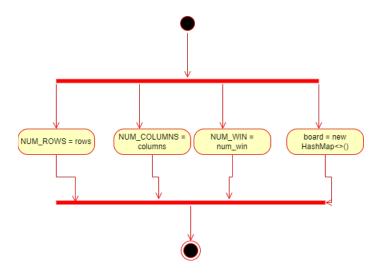
AbsGameBoard:

toString

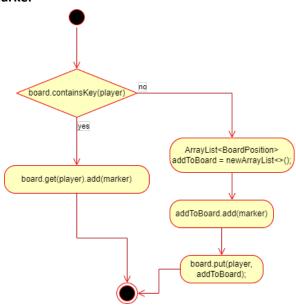


GameBoardMem:

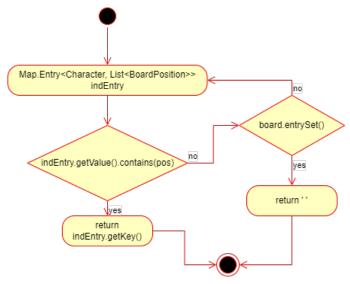
Constructor



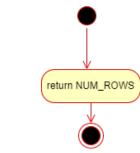
PlaceMarker



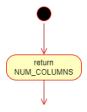
WhatsAtPos



${\sf getNumRows}$



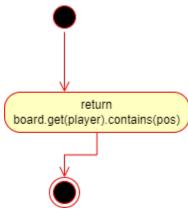
getNumColumns



${\sf getNumToWin}$



isPlayerAtPos



Test Cases

JUnit test function.
Test the following methods:

Constructor (final int rows, final int columns, final int num win)

Constructor (linal int rows, linal in								
Input:	Outp	ut:						Reason:
								This test case is unique and distinct because I
Rows = 3	State:							am creating an empty GameBoard of the
Columns = 3		0		1		2		minimum size.
Num_win = 3	0							
	1							Function Name:
	2							testGameBoardConstructor_min_num_row_col
Input:	Outp	ut:						Reason:
								This test case is unique and distinct because I
Rows = 100	State:							am creating an empty GameBoard of the
Columns = 100	100x1	00	em	pty	tab	le		maximum size.
Num_win = 25								
								Function Name:
								testGameBoardConstructor_max_num_row_co
								1
Input:	Outp	ut:						Reason:
								This test case is unique and distinct because I
Rows = 7	State:							am creating an empty GameBoard of different
Columns = 5	0	1	2	3	4	5	6	numbers of rows and columns.
Num_win = 4	0							
	1							Function Name:
	2							testGameBoardConstructor_diff_num_row_col
	3							
	4							

Input:				Output:	Reason:
					This test case is unique and
State:				checkSpace = true	distinct because the space is
					empty.
	0	1	2	State of the board is unchanged	
0	Χ				Function Name:
1					testCheckSpace_empty_space
2	0				
pos.getf	Row() = :	1			
pos.get(Column() = O			
Input:				Output:	Reason:
					This test case is unique and
State:				checkSpace = false	distinct because the space is full
					and we are checking a unique
	0	1	2	State of the board is unchanged	character.
0	Х				
1					Function Name:
2	0				testCheckSpace_not_space
			<u> </u>		
pos.getf	Row() = (0			
pos.get(Column() = O			
Input:				Output:	Reason:
					This test case is unique and
State:				checkSpace = false	distinct because every space is
					full and we are checking a
	0	1	2	State of the board is unchanged	unique character.
0	Х	Х	Х		
1	Х	0	Х		Function Name:
2	0	0	0		testCheckSpace_full_space
pos.getf	Row() = :	1			
pos.get(
	•	-		ı	ı

boolean checkHorizontalWin (BoardPosition lastPos, char player)

State: Num_win 0 0 X 1 2 C		1 X	2 X	Output: checkHorizontalWin = true	Reason: This test case is unique and distinct because there is a horizontal win and our last placed character is in the
Num_win		X			because there is a horizontal win and
Num_win		X			
0 0 X 1		X			our last placed character is in the
0 X		X			1 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 -
1			Х	State of the board is	middle.
l) (Ω		unchanged	
l)	0			Function Name:
	I				testHorizontalWin_is
		<u>- </u>	<u> </u>		_Win_Add_Middle
lastPos.ge	tRow() = 0			
lastPos.ge		-	1		
10301 03.80	colui	()	-		
player = X	•				
Input:				Output:	Reason:
					This test case is unique and distinct
State:				checkHorizontalWin = false	because there is not a number in a
Num win	= 3			1	row that equals the number to win.
0		1	2	State of the board is	The state of all of the frame to will
0 X		<u>+</u> X		unchanged	Function Name:
		^		anenangea	testHorizontalWin_no_Win
2 C		0			testriorizoritarviii_no_vviii
2 10	,	U			
		١			
lastPos.ge		-			
lastPos.ge	etColun	nn() =	1		
	,				
player = X	<u> </u>				B
Input:				Output:	Reason:
CLA				de al la compagnition de la comp	This test case is unique and distinct
State:	2			checkHorizontalWin = true	because there is a horizontal win and
Num_win				Chata afalba ba a 13	our last placed token is the rightmost
0		1	2	State of the board is	on the win.
0 X		X	Х	unchanged	
1					Function Name:
2 C)	0			testHorizontalWin_is_Win_Add_Right
lastPos.getRow() = 0					
lastPos.ge	tColun	nn() =	2		
player = X	·				
Input:				Output:	Reason:
-					This test case is unique and distinct
State:				checkHorizontalWin =	because there is a horizontal win but
player = X				Output:	

Num_	_win = 3				our last placed token is the leftme
	0	1	2	State of the board is	on the win.
0	Х	Х	Х	unchanged	
1					Function Name:
2	0	0		71	testHorizontalWin_is_Win_Add_L
	os.getRo os.getCo				
olaye	r = X				

	in (BoardPosition lastE	
Input:	Output:	Reason:
		This test case is unique and distinct
State:	checkVerticalWin = true	because there is a vertical win and
Num_win = 3		our last placed character is in the
0 1 2	State of the board is	middle.
0 X	unchanged	
1 X O		Function Name:
2 X O		testVerticalWin_is
		_Win_Add_Middle
lastPos.getRow() = 1		
lastPos.getColumn() = 0		
player = X		
Input:	Output:	Reason:
		This test case is unique and distinct
State:	checkVerticalWin = false	because there is not a number in a
Num_win = 3		vertical that equals the number to
0 1 2	State of the board is	win.
0 X	unchanged	
1 X		Function Name:
2 0 0		testVerticalWin_no_Win
lastPos.getRow() = 1		
lastPos.getColumn() = 0		
lasti os.geteoiaiiii() – o		
player = X		
Input:	Output:	Reason:
input.	Output.	
State:	checkVerticalWin = true	This test case is unique and distinct because there is a vertical win and
Num_win = 3	CHECKVELUCATVVIII – UIUE	our last placed token is the
	State of the board is	uppermost on the win.
		appermost on the will.
0 X	unchanged	Function Name:
1 X 0		testVerticalWin_is_Win_Add_Upper
2 X O		testverticalwill_is_will_Add_Opper
lastPos.getRow() = 0		
		İ
lastPos.getColumn() = 0		
lastPos.getColumn() = 0		
lastPos.getColumn() = 0 player = X		
lastPos.getColumn() = 0	Output:	Reason:
lastPos.getColumn() = 0 player = X	Output: checkVerticalWin = true	Reason: This test case is unique and distinct because there is a vertical win but

Num_	win = 3				our last placed token is the
	0	1	2	State of the board is	lowermost on the win.
0	Х			unchanged	
1	Х	0			Function Name:
2	Х	0			testVerticalWin_is_Win_Add_Lower
	_	w() = 2 lumn() =	= 0		

Boolean checkDiagonalWin (BoardPosition lastPos, char player)

Input:	all Clic	ECKD	rage	JIIaIWII	Output:	Reason:
iliput.					Output.	This test case is unique and
State:					checkDiagonalWin = true	distinct because it tests the left
Num v	win = 3				checkbiagonalwin - true	diagonal and the last placed
I Train_v	0	1		2	State of the board is unchanged	character is in the middle.
0	X	+ -			State of the board is unchanged	character is in the imagic.
1		Х				Function Name:
2	0	0		Χ		testDiagonallWin_is
	10	0		^		_Win_Add_Middle_Left_D
lactDoc	s.getRov	۸ <i>،</i> (۱ –	1			
	getCol	• • • • • • • • • • • • • • • • • • • •				
iastros	s.getCoi	umm	.) — <u> </u>			
player	= X					
Input:	- A				Output:	Reason:
iliput.					Output.	This test case is unique and
State:					checkDiagonalWin = true	distinct because it tests the left
Num v	win = २				ancondiagonary in - true	diagonal and the last placed
I Nam_v	0	1	2	3	State of the board is unchanged	character is the upper/leftmost
0		X		-	State of the board is unchanged	and it is not on the main
1		^	Х			diagonal.
2	0	0	^			diagonal.
3	0	U		X		Function Name:
3			1			testDiagonallWin_is
						_Win_Add_Up/Left_Left_D
lastDes	antDay	() _	0			
	s.getRo	• • • • • • • • • • • • • • • • • • • •				
IdStPOS	.getCol	umm) = 1			
playor	_ V					
player	= X				Outrout	Reason:
Input:					Output:	This test case is unique and
State:					checkDiagonalWin = true	distinct because it tests the left
Num v	win = 3				Checkbiagonalyvin – ti de	diagonal and the last placed
Nam_v	0	1		2	State of the board is unchanged	character is the
0	X	+			State of the board is unchanged	lowest/rightmost.
1	^	Х				is it esty right most.
2	0	0		X		Function Name:
	10	IU		^		testDiagonallWin_is
						_Win_Add_Low/Right_Left_D
lastPos	getRo	w() =	2			
lastPos	.getCol	umn() = 2			
player	= X					
Input:					Output:	Reason:

State: Num_v	win = 3	}			This test case is unique and distinct because it tests the right diagonal and the last placed character is in the	
	0	1	2	3	State of the board is unchanged	middle and it is not on the main
0			Х			diagonal.
1		Χ				
2	Χ	0	0			Function Name:
3						testDiagonallWin_is _Win_Add_Middle_Right_D
lastPos lastPos	-					
player	= X					
Input:					Output:	Reason:
State: Num_v	win = 3	ł.			checkDiagonalWin = true	This test case is unique and distinct because it tests the right diagonal and the last
TVairi_V	0	1		2	State of the board is unchanged	placed character is the
0	1			X	and the same and t	upper/rightmost.
1		Х				
2	Х	0		0		Function Name: testDiagonallWin_is
lastPos lastPos player	s.getCo					_Win_Add_Up/Right_Right_D
Input:					Output:	Reason:
. .						This test case is unique and
State:					checkDiagonalWin = true	distinct because it tests the
Num_v				2	State of the board is unchanged	right diagonal and the last placed character is the
0	0	1		2 X	State of the board is unchanged	lowest/leftmost.
1		X	-	۸		lowesy leranost.
2	Х	0		0		Function Name:
						testDiagonallWin_is _Win_Add_Low/Left_Right_D
lastPos.getRow() = 2 lastPos.getColumn() = 0						
player	= X					
Input:					Output:	Reason: This test case is unique and
input:						
State:					checkDiagonalWin = false	distinct because it tests the left

	0	1	2	State of the board is unchanged	diagonal and there is no
0	Х				diagonal win.
1		Х			Function Name:
2	0	0	0		testDiagonallWin_no
					_Win
	s.getRow s.getColu		2		
player	= O				

- Create 4 distinct test cases

Input:	. 31361116			Output:	Reason:
					This test case is unique and
State:				checkForDraw = true	distinct because it tests a full
3					board on the minimum board
	0	1	2	State of the board is unchanged	size.
0	0	Х	0		
1	X	X	0		Function Name:
2	0	0	Х		testCheckForDraw_is
					_Draw
Input:				Output:	Reason:
					This test case is unique and
State:				checkForDraw = false	distinct because it tests an
					empty board.
	0	1	2	State of the board is unchanged	
0					Function Name:
1					testCheckForDraw_is
2					_Empty
Input:				Output:	Reason:
					This test case is unique and
State:				checkForDraw = false	distinct because it tests a
	1	1			partially full board.
	0	1	2	State of the board is unchanged	
0	0	Х	0		Function Name:
1	Х				testCheckForDraw_is
2	0	0	X		_Partially_Full
Input:				Output:	Reason:
					This test case is unique and
State:				checkForDraw = true	distinct because it tests a full
					board on the maximum board
100x100) board	that is f	ull	State of the board is unchanged	size.
					Function Name:
					testCheckForDraw_is
					_Full_Max_Board

- Create 5 distinct test cases

Input:				Output:	Reason:
Pos.getl	Row() =	0:		checkWhatsAtPos = ' '	This test case is unique and distinct because it tests a
Pos.get	.,	-			character that is empty.
J	·	.,		State of the board is unchanged	. ,
State:					Function Name:
		1	1		testWhatsAtPos_is
	0	1	2		_Empty
0					
1					
2				Out and	Barana
Input:				Output:	Reason: This test case is unique and
Pos.get	Row() =	٥٠		checkWhatsAtPos = X	distinct because it tests a
Pos.get				checkwhats/tti os = X	unique character.
. 55.650	•	., •		State of the board is unchanged	
State:					Function Name:
					testWhatsAtPos_is
	0	1	2		_X_Unique_Char
0	Χ				
1					
2					
Input:				Output:	Reason:
•	D - ()	0		-	This test case is unique and
Pos.get				Output: checkWhatsAtPos = O	This test case is unique and distinct because it tests a
•				checkWhatsAtPos = O	This test case is unique and
Pos.geti				-	This test case is unique and distinct because it tests a unique character.
Pos.get				checkWhatsAtPos = O	This test case is unique and distinct because it tests a unique character. Function Name:
Pos.geti			2	checkWhatsAtPos = O	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is
Pos.geti	Column(() = 0	2	checkWhatsAtPos = O	This test case is unique and distinct because it tests a unique character. Function Name:
Pos.geti Pos.geti State:	Column(() = 0	2	checkWhatsAtPos = O	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is
Pos.geti Pos.geti State:	Column(() = 0	2	checkWhatsAtPos = O	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is
Pos.gettl Pos.gettl State:	Column(() = 0	2	checkWhatsAtPos = O	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is
Pos.gettl Pos.gettl State: 0 1 2 Input:	Column(1	2	checkWhatsAtPos = O State of the board is unchanged Output:	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is _O_Dist_Char Reason: This test case is unique and
Pos.gett Pos.gett State: 0 1 2 Input: Pos.gett	0 0 0	2) = 0	2	checkWhatsAtPos = O State of the board is unchanged	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is _O_Dist_Char Reason: This test case is unique and distinct because it tests a
Pos.gettl Pos.gettl State: 0 1 2 Input:	0 0 0	2) = 0	2	checkWhatsAtPos = O State of the board is unchanged Output: checkWhatsAtPos = X	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is _O_Dist_Char Reason: This test case is unique and distinct because it tests a unique character at the bottom
Pos.gettl Pos.gettl State: 0 1 2 Input: Pos.gettl	0 0 0	2) = 0	2	checkWhatsAtPos = O State of the board is unchanged Output:	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is _O_Dist_Char Reason: This test case is unique and distinct because it tests a
Pos.gett Pos.gett State: 0 1 2 Input: Pos.gett	0 0 0	2) = 0	2	checkWhatsAtPos = O State of the board is unchanged Output: checkWhatsAtPos = X	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is _O_Dist_Char Reason: This test case is unique and distinct because it tests a unique character at the bottom left corner.
Pos.gettl Pos.gettl State: 0 1 2 Input: Pos.gettl	O O O Column((i) = 0 1 2 (i) = 2		checkWhatsAtPos = O State of the board is unchanged Output: checkWhatsAtPos = X	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is _O_Dist_Char Reason: This test case is unique and distinct because it tests a unique character at the bottom left corner. Function Name:
Pos.gettl Pos.gettl State: 0 1 2 Input: Pos.gettl Pos.gettl	0 0 0	2) = 0	2	checkWhatsAtPos = O State of the board is unchanged Output: checkWhatsAtPos = X	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is _O_Dist_Char Reason: This test case is unique and distinct because it tests a unique character at the bottom left corner. Function Name: testWhatsAtPos_is
Pos.geti Pos.geti O 1 2 Input: Pos.geti Pos.geti O State:	O O O Column((i) = 0 1 2 (i) = 2		checkWhatsAtPos = O State of the board is unchanged Output: checkWhatsAtPos = X	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is _O_Dist_Char Reason: This test case is unique and distinct because it tests a unique character at the bottom left corner. Function Name:
Pos.gettl Pos.gettl State: 0 1 2 Input: Pos.gettl Pos.gettl	O O O Column((i) = 0 1 2 (i) = 2		checkWhatsAtPos = O State of the board is unchanged Output: checkWhatsAtPos = X	This test case is unique and distinct because it tests a unique character. Function Name: testWhatsAtPos_is _O_Dist_Char Reason: This test case is unique and distinct because it tests a unique character at the bottom left corner. Function Name: testWhatsAtPos_is

Input:				Output:	Reason:
	Row() = Column(checkWhatsAtPos = O	This test case is unique and distinct because it tests a unique character on a
State:				State of the board is unchanged	completely full board in the upper right corner.
	0	1	2		Function Name:
0	Х	0	0		testWhatsAtPos_is
1	Х	0	0		_Dist_Char_Full_Upper_Right
2	0	Х	Х		

- Create 5 distinct test cases

Input:				Output:	Reason:
Pos.getF	Row() = (O;		checklsPlayerAtPos = false	This test case is unique and distinct because it tests an
Pos.get(.,	-		,	empty board with a nonempty
				State of the board is unchanged	player character. It compares a
Player =	Χ				nonwhite space to a
a					whitespace.
State:					Function Name:
	0	1	2		testisPlayerAtPos_is
0	U	1			_empty_vs_Unique_player
1					
2					
Input:				Output:	Reason:
					This test case is unique and
Pos.getF				checkIsPlayerAtPos = true	distinct because it tests an
Pos.get(Column() = O			empty board with an empty
Dlavor -	, ,			State of the board is unchanged	player character.
Player =					Function Name:
State:					testisPlayerAtPos_is
					_empty_vs_empty_player
	0	1	2		
0					
1					
2					
Input:				Output:	Reason:
Pos.getF	20w() - (٦٠		checkIsPlayerAtPos = true	This test case is unique and distinct because it tests an non
Pos.get(checkist layerAti os – ti de	empty position on the board
	(, -		State of the board is unchanged	with the correct player
Player =	Χ				character.
State:					Function Name:
					testisPlayerAtPos_is
	0 X	1	2		_true
0	۸				
2					
Input:		<u>I</u>	1	Output:	Reason:
·					This test case is unique and
Pos.getRow() = 0;				checklsPlayerAtPos = false	distinct because it tests an non
Pos.getColumn() = 0					empty position on the board
				State of the board is unchanged	

Player = X					with the incorrect player character.
State:					Function Name:
	0	1	2		testisPlayerAtPos_is
	-	1			_false
0	0				_laise
1					
2					
Input:				Output:	Reason:
					This test case is unique and
Pos.getI	Row() =	1		checkIsPlayerAtPos = false	distinct because it compares a
Pos.get(Column(() = 1			new character that has not
				State of the board is unchanged	been placed on the board.
Player =	A				·
', -					Function Name:
State:					testisPlayerAtPos_is
State.					_new_Char
	0	1	2		_new_endi
0		-	-		
1	Х	0			
2					
	1	1			

- Create 5 distinct test cases

Input:	Output:	Reason: This test case is unique and distinct because it tests an empty board with a unique
marker.getRow() = 0; marker.getColumn() = 0	State:	
Player = X	0 1 2 0 X 1	character. Function Name:
State:	2	testPlaceMarker_Empty
0 1 2		
2		
Input:	Output:	Reason: This test case is unique and
marker.getRow() = 2 marker.getColumn() = 0	State:	distinct because it tests an almost full board with a unique
Player = O	0 1 2 0 0 X 0	character on the leftmost lower position.
State:	1 X O O 2 O X X	Function Name: testPlaceMarker_Almost_Full
0 1 2 0 0 X 0 1 X 0 0 2 X X		
Input:	Output:	Reason:
marker.getRow() = 2 marker.getColumn() = 0	State: 0 1 2	This test case is unique and distinct because I am placing a marker representing a player who has not been placed on this
Player = A	0 0 1 X	board before.
State:	2 A	Function Name: testPlaceMarker_New_Char
0 1 2 0 0 1 X		
2	Outroot	Page 1991
Input:	Output:	Reason: This test case is unique and
marker.getRow() = 0 marker.getColumn() = 2	State:	distinct because I am placing a

					0	1	2	unique marker in the upper		
Player = A				0	0		Α	right corner of the board.		
					Х					
State:	State:							Function Name:		
		_	, ,			•		testPlaceMarker_Upper_Right		
	0	1	2							
0	0									
1	Χ									
2										
Input:				Outpu	ıt:			Reason:		
								This test case is unique and		
marker.	getRow	() = 2		State:				distinct because I am placing a		
marker.	getColu	mn() = 2	•					unique marker in the lower		
					0	1	2	right corner of the board.		
Player =	: A			0	0					
				1	Х			Function Name: testPlaceMarker_Lower_Right		
State:				2			Α			
	0	1	2							
0	0									
1	Χ									
2										