CPSC 2150 Project Report

Mark Sheldon

Requirements Analysis

Functional Requirements:

As a player, I <what/need/can> <goal> so that <reason>

- 1. As a player, I can know what character I am so that I know which markers are mine
- 2. As a player, I can view the board so that I know where markers have been played
- 3. As a player, I can view the number of rows on the board so that I know what indices are valid
- 4. As a player, I can view the number of columns on the board so that I know what indices are valid
- 5. As a player, I need to know if my position choice is invalid so that I can choose a new one
- 6. As a player, I need to know who wins so that I know who the winner is
- 7. As a player, I need to know when I get enough markers in a row so that I know I won the game
- 8. As a player, I need to know when my opponent gets enough markers in a row so that I know I lost
- 9. As a player, I need to know if someone wins so that I know the game is over
- 10. As a player, I need to know if there are no more spaces available so I know the game is a draw
- 11. As a player, I need to know if the game is a draw so I know the game is over
- 12. As a player, I need to be able to play again once the game is over so that I can play another game
- 13. As a player, I need to be asked if I want to stop playing so that I can choose to not player another game
- 14. As a player, I can see the board after my opponent plays so I can plan my next move
- 15. As a player I can see the board after I play so that I can make sure I played in the right spot
- 16. As a player, I need to know whose turn it is so that I can know if it is my turn
- 17. As a player, I need to be asked how many rows the board should have so I can choose the board size
- 18. As a player, I need to be asked how many columns the board should have so I can choose the board size
- 19. As a player, I need to be asked how many markers in a row are needed to win so I can decide how long it will take to win
- 20. As a player, I need to be asked how many players are playing so I can have everyone who wants to play, play

Non-Functional Requirements

- 1. The game must be coded in java
- 2. The board can be structured as a 2D array of characters
- 3. The board can be structured as a map of character-BoardPosition list pairs
- 4. The game must have a GUI

- 5. The GUI must be coded in Java Swing
- 6. The board must have a flexible number of rows and columns
- 7. The board's indexing must start in the top left corner
- 8. The program must have a fast version and a memory efficient version
- 9. The program must allow for between 2 and 10 players
- 10. The program must check for a win after each turn
- 11. The program must check for a draw after each turn
- 12. The program must cycle until the player closes the window
- 13. The program must allow a user to change settings between games
- 14. There should only be one main method
- 15. The main method should be inside of the GameScreen class
- 16. Input for a marker's indices must be read row first, then column
- 17. Input must only be read through the GameScreen class
- 18. Output must only be sent through the GameScreen class

System Design

Class 1: TicTacToeController

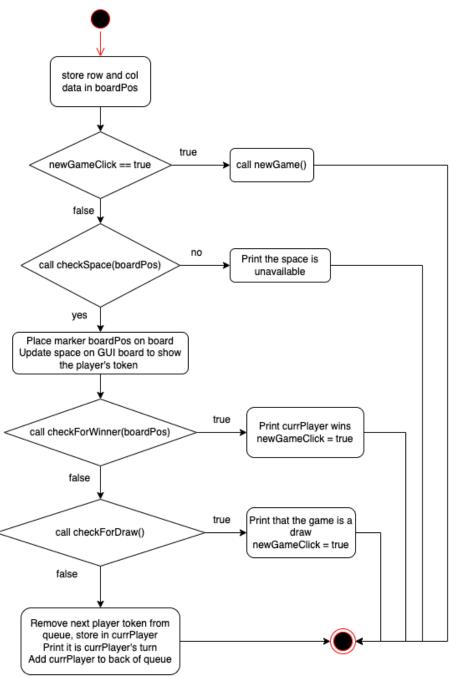
Class Diagram:

TicTacToeController

- + MAX_PLAYERS: int[1] = 10 {readOnly} + POSSIBLE_MARKERS: int[2-10] {readOnly}
- curGame: IGameBoard[1]
- screen: TicTacToeView[1]
- numPlayers: int[1]
- playerMarkers: Queue<Character>[1]
- currPlayer: Character[1]
- newGameClick: boolean[1]
- + TicTacToeController(IGameBoard, TicTacToeView, int): void
- + processButtonClick(int, int): void
- newGame(void): void

Activity Diagrams:

processButtonClick



Class 2: BoardPosition.java

Class diagram

BoardPosition

- row: int[1] - col: int[1]

+ getRow(void): int + getColumn(void): int + equals(Object): boolean + toString(void): String

+ BoardPosition(int, int): void

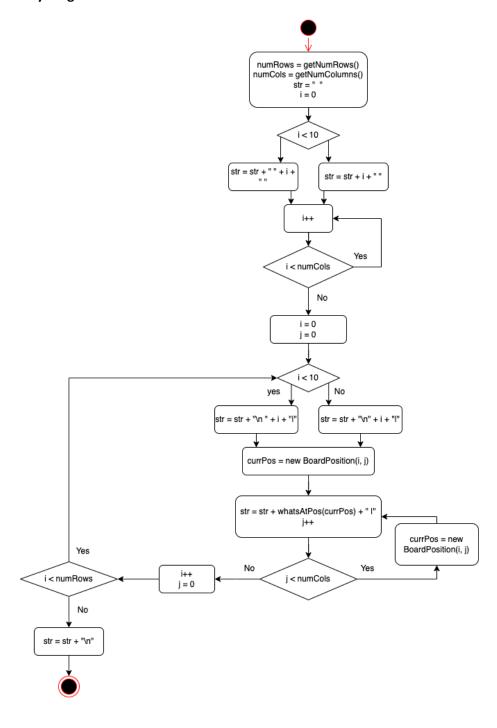
Class 3: AbsGameBoard.java

Class Diagram:

AbsGameBoard

+ toString(void): (String)

Activity Diagram:



Class 4: GameBoard.java

Class diagram

GameBoard

- board: char[3-100][3-100]

- numToWin: int[1]

- markersPlayed: int[1]

+ getNumRows(void): int

+ getNumColumns(void):int

+ getNumToWin(void): int

+ placeMarker(BoardPosition, char): void

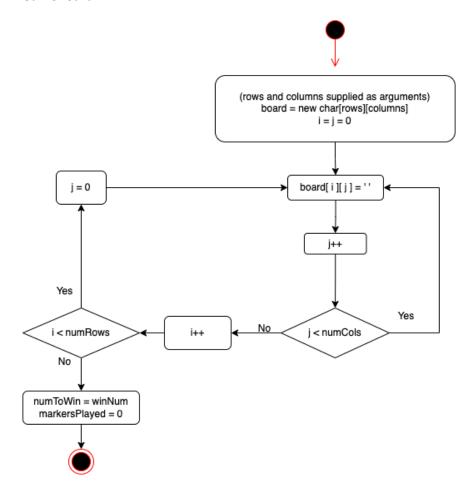
+ whatsAtPos(BoardPosition): char

+ GameBoard(int, int, int): void

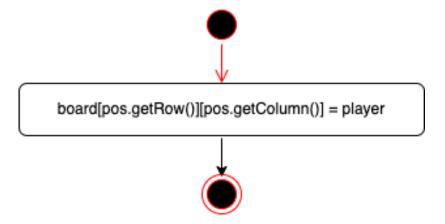
+ checkForDraw(void): boolean

Activity diagrams:

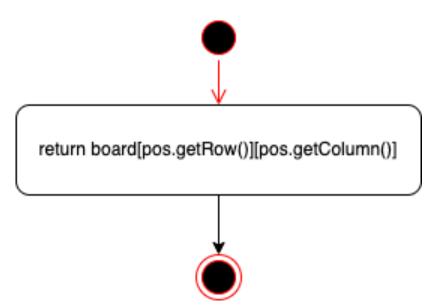
GameBoard:



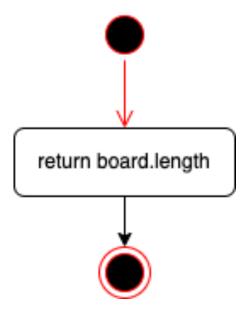
placeMarker:



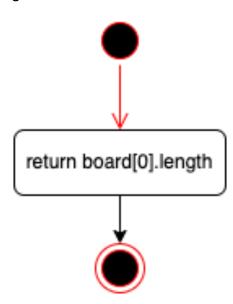
whatsAtPos:



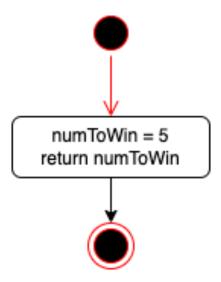
getNumRows:



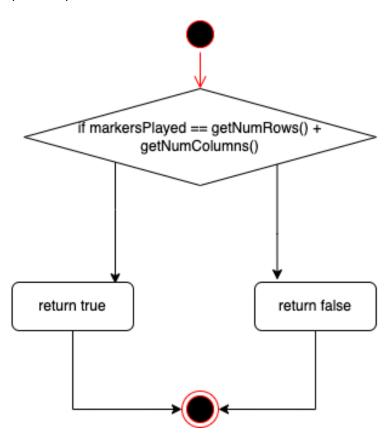
getNumColumns:



getNumToWin:



(override) checkForDraw:



Class 5: GameBoardMem

Class Diagram

GameBoardMem

rows: int[1]columns: int[1]numToWin: int[1]markersPlayed: int[1]

- board: Map<Character, List<BoardPosition>>

+ GameBoardMem(int, int, int): void

+ getNumRows(void): int + getNumColumns(void): int + getNumToWin(void): int

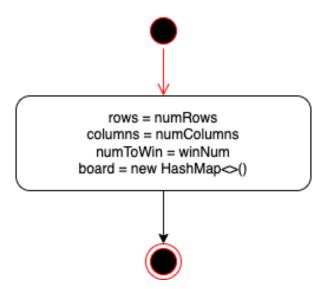
+ placeMarker(BoardPosition, char): void

+ whatsAtPos(BoardPosition): char

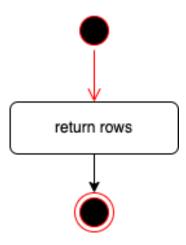
+ isPlayerAtPos(BoardPosition): boolean

Activity Diagrams

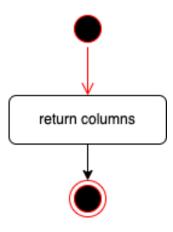
GameBoardMem:



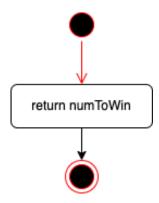
getNumRows:



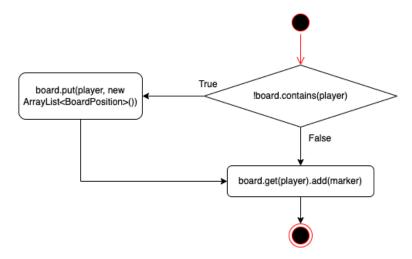
getNumColumns:



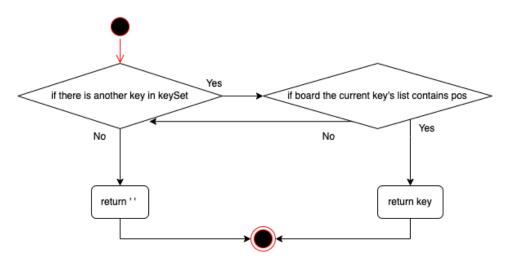
getNumToWin:



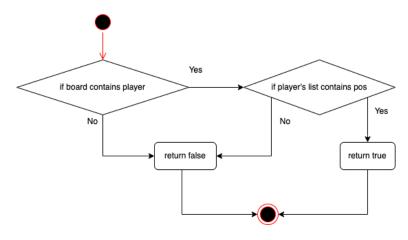
placeMarker:



whatsAtPos:



isPlayerAtPos:



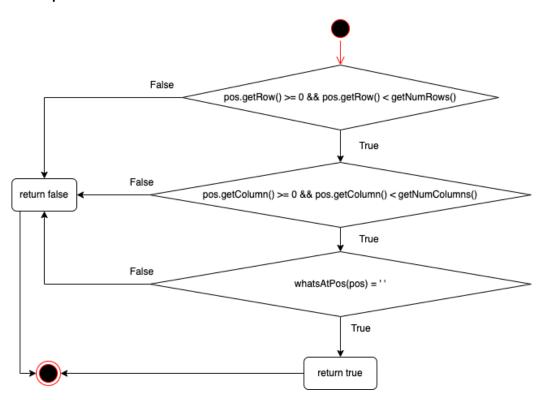
Class 6: IGameBoard

Class Diagram:

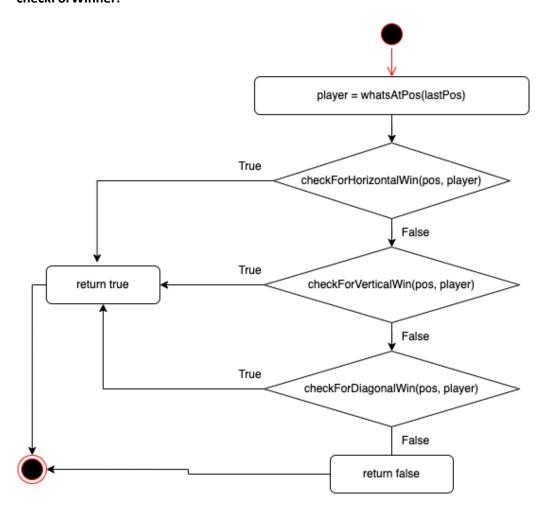
IGameBoard + MAX_ROW: int[1] = 100 {readOnly} + MAX_COL: int[1] = 100 {readOnly} + MAX_WIN: int[1] = 25 {readOnly} + getNumRows(void): int + getNumColumns(void): int + getNumToWin(void): int + checkSpace(BoardPosition): boolean + placeMarker(BoardPosition, char): void + checkForWinner(BoardPosition): boolean + checkForDraw(void): boolean + checkHorizontalWin(BoardPosition, char): book + checkVerticalWin(BoardPosition, char): boolear + checkDiagonalWin(BoardPosition, char): boolea + whatsAtPos(BoardPosition): char + isPlayerAtPos(BoardPosition, char): boolean

Activity Diagram:

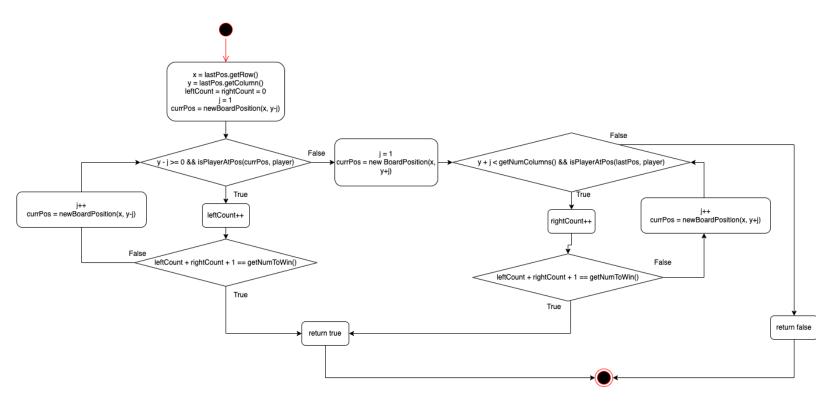
checkSpace:



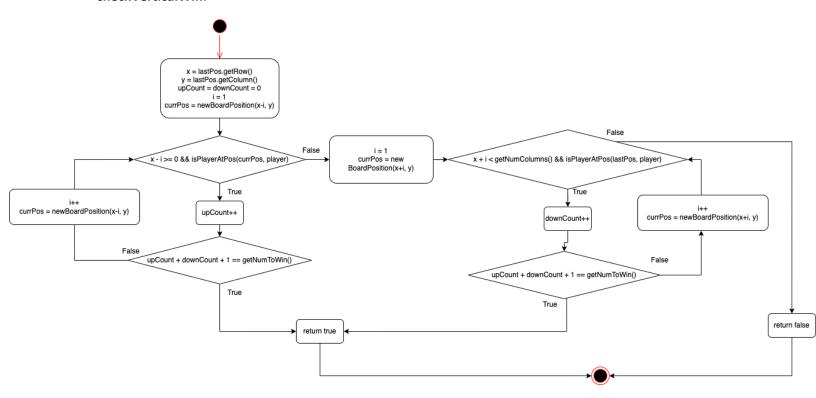
checkForWinner:



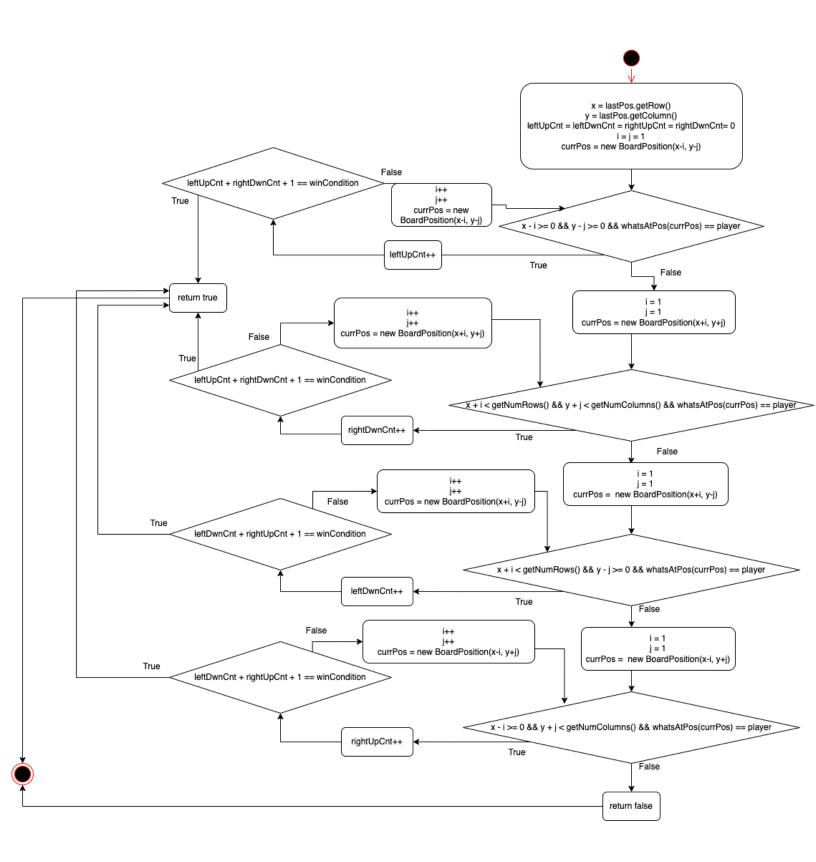
checkHorizontalWin:



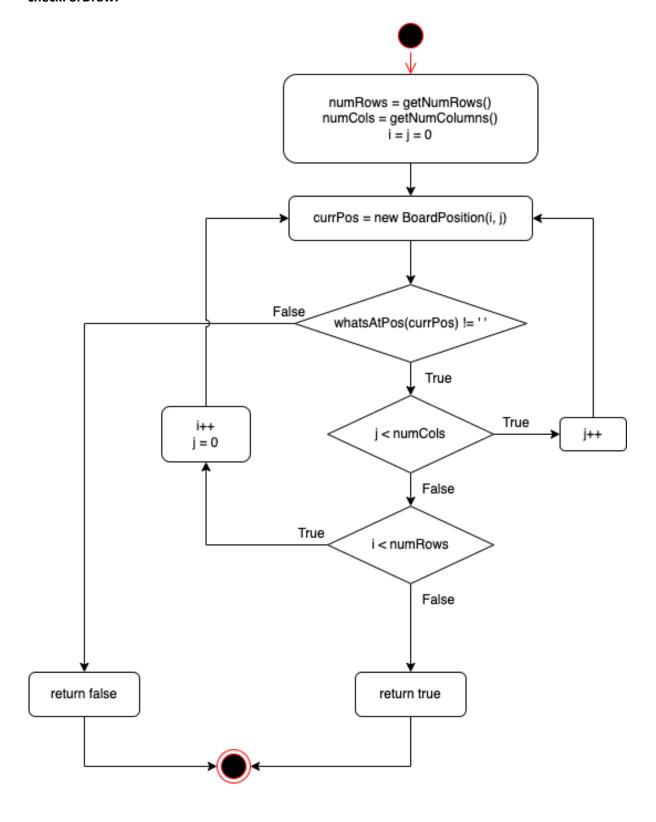
checkVerticalWin:



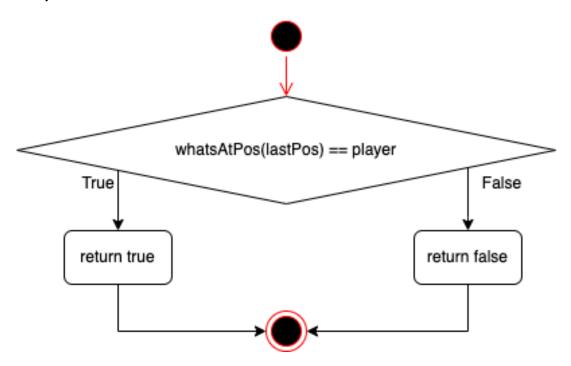
checkDiagonalWin:



checkForDraw:



isPlayerAtPos:



Test Cases

Details in Project 4.

GameBoard(int rows, int columns, int winNum)

				0 ± 0		, III WINNUM)
Input:	Output:					Reason:
						This test case is unique because it is creating a board of
State:	State:				_	the minimum size for all 3 parameters
	0		1	2		
rows = 3	0					Function:
columns = 3	1					testGameBoard_min_rows_cols_winNum
winNum = 3	2					
	numTo\	Vin =	: 3			
	markers			0;		
Input:	Output:					Reason:
						This test case is unique because it creates a board with
State:	State:					the maximum number of rows and columns using the
	(empty	100x	100	tabl	e)	maximum number to win
rows = 100					•	
columns = 100	numTo\	Vin =	= 25			Function:
winNum = 25	markers	Play	ed =	0;		testGameBoard max rows cols winNum
Input:	Output:					Reason:
						This test case is unique because it creates a routine
State:	State:					board with no minimums or maximums for any
	0	1	2	3	4	parameters
rows = 5	0					
columns = 5	1					Function:
winNum = 4	2					testGameBoard no max no min
	3					
	4					
	[4					
	numTo\	Nin -	- 1			
	markers			۸٠		
	markers	ridy	eu –	υ,		

checkSpace(BoardPosition pos)

Counting Counting	checkSpace(BoardPosition pos)									
State: O 1 2 3 4	Input:						Output:	Reason:		
State: O 1 2 3 4							checkSpace = true	This case is unique because it is a routine		
State: O 1 2 3 4	State	2:					·			
O		0	1	2	3	4	state of the board is	teet of an open space		
This case is unique because it is a routine test of an occupied space	0			_				Function:		
Dos.getRow = 0 Dos.getRow = 0 State: O							difficulties			
State: Output: checkSpace = false State of the board is unchanged State: Output: checkSpace = false State of the board is unchanged State: Output: checkSpace = false State of the board is unchanged State: Output: checkSpace = false State of the board is unchanged State: Output: checkSpace = false State of the board is unchanged State of the board is unchanged State: Output: checkSpace = false State of the board is unchanged State: Output: checkSpace = false State of the board is unchanged State of the board is unchan	l							cestemenspace_open_space		
Dos.getRow = 0	_									
pos.getRow = 0 pos.getColumn = 0 Input: State: O 1 2 3 4	3									
Dote	4									
Dote										
Input: CheckSpace = false State: State of the board is unchanged State: State: State: Output: CheckSpace = false State: Stat	-									
CheckSpace = false CheckSpace = false	pos.g	getCo	lumr	า = 0						
CheckSpace = false CheckSpace = false										
State: O 1 2 3 4	Inpu	t:					Output:	Reason:		
state of the board is unchanged Function: testCheckSpace_taken_space Function: This case is unique because it tests that the function returns false when a position not on the board is passed in Function: testCheckSpace_invalid_space Function: testCheckSpace_invalid_space							checkSpace = false	This case is unique because it is a routine		
O	State	2:						test of an occupied space		
testCheckSpace_taken_space testCheckSpace_taken_space testCheckSpace_taken_space testCheckSpace_taken_space testCheckSpace_taken_space testCheckSpace_taken_space testCheckSpace_taken_space testCheckSpace_taken_space Reason: This case is unique because it tests that the function returns false when a position not on the board is passed in testCheckSpace_invalid_space Function: testCheckSpace_invalid_space		0	1	2	3	4	state of the board is			
pos.getRow = 0 pos.getColumn = 0 Input: State: O 1 2 3 4	0	Χ					unchanged	Function:		
pos.getRow = 0 pos.getColumn = 0 Input: State: Output: checkSpace = false State: State: Output: checkSpace = false state of the board is unchanged Function: testCheckSpace_invalid_space	1							testCheckSpace taken space		
pos.getRow = 0 pos.getColumn = 0 Input: State: O 1 2 3 4 0 0 1 0 2 0 3 0 4 0 5 1 1 6 1 7 1 8 1 8 1 9 1 1 1 1 1 1 1 1 1	l									
pos.getRow = 0 pos.getColumn = 0 Input: State: Output: checkSpace = false State of the board is unchanged Function: testCheckSpace_invalid_space pos.getRow = 0	_									
pos.getRow = 0 pos.getColumn = 0 Input: CheckSpace = false State: Output: CheckSpace = false state of the board is unchanged Function: testCheckSpace_invalid_space pos.getRow = 0										
Dos.getColumn = 0 Coulomb	4									
Dos.getColumn = 0 Coutput: CheckSpace = false State:				_						
Input: State: Output: checkSpace = false State of the board is unchanged Function: testCheckSpace_invalid_space pos.getRow = 0										
State: O 1 2 3 4	pos.g	pos.getColumn = 0								
State: O 1 2 3 4										
State: O 1 2 3 4	Inpu	t:					Output:	Reason:		
0 1 2 3 4 0 1 1 2 1 1 3 1 1 4 1 1 pos.getRow = 0 state of the board is unchanged Function: testCheckSpace_invalid_space							checkSpace = false	This case is unique because it tests that		
0 unchanged 1 runction: 2 testCheckSpace_invalid_space pos.getRow = 0	State:							the function returns false when a		
Function: testCheckSpace_invalid_space pos.getRow = 0		0	1	2	3	4	state of the board is	position not on the board is passed in		
Function: testCheckSpace_invalid_space pos.getRow = 0	0						unchanged			
<pre>2</pre>							_	Function:		
9								testCheckSpace invalid space		
pos.getRow = 0										
pos.getRow = 0										
	4									
				_						
pos.getColumn = 5	-									
	pos.g	getCo	lumr	า = 5						

checkHorizontalWin(BoardPosition lastPos, char player)

<u>checkHorizontalWin</u>	n (BoardPosition	lastPos, char player)		
Input:	Output:	Reason:		
	checkHorizontalWi	This case is unique because it is a win resulting from		
State: (numToWin = 4)	n = true	a play on the right side of the win		
0 1 2 3 4	state of the board	Function		
0 0 0 0 0		Function: testCheckHorizontalWin play on end		
1	is unchanged	cestenceknorizontarwin_pray_on_ena		
2 X				
3 X A A A A A A A A A A A A A A A A A A				
4 X				
lastPos.getRow = 0				
lastPos.getColumn = 3				
player = 'O'				
Input:	Output: checkHorizontalWi	Reason: This case is unique because it is a win resulting from		
State: (numToWin = 4)	n = true	a play in the middle of the winning string (meaning		
0 1 2 3 4	ii – ti de	the function must check both left and right of the last		
0 0 0 0 0	state of the board	position played)		
1	is unchanged			
2 X		Function:		
3 X		testCheckHorizontalWin_play_in_middl		
4 X		е		
lastPos.getRow = 0				
lastPos.getColumn = 2				
player = 'O'				
Input:	Output:	Reason:		
	checkHorizontalWi	This case is unique because it tests that the function		
State: (numToWin = 4)	n = true	can recognize a win with a full board		
0 1 2 3 4				
0 0 0 0 X	state of the board	Function:		
1 X X X O X	is unchanged	testCheckHorizontalWin_full		
2 O O O X O				
3 X X X O X				
4 0 0 0 X 0				
lostDec zetDevic O				
lastPos.getRow = 0				
lastPos.getColumn = 3 player = 'O'				
player - U				

Inp	ut:					Output:	Reason:
						checkHorizontalWi	This case is unique because it is a win resulting from
Sta	te: (r	num	ToW	/in =	4)	n = true	a play in the corner of the board that results in win.
	0	1	2	3	4		(It should not try and read out of bounds when
0	0	0	0	0		state of the board	reading left)
1						is unchanged	
2	2 X						Function:
3	3 X						testCheckHorizontalWin_corner_win
4	Χ						
	•						
last	Pos.	getF	Row	= 0			
last	Pos.	get(Colur	nn =	= 0		
plav	yer =	· 'O'					
	•						

Input:						Output:	Reason:			
						checkVerticalWin	This case is unique because it is a win resulting from a			
State: (numToWin = 4)						= true	play on the end of the winning string			
l 🛌	0	1	2	3	4					
0						state of the board	Function:			
1	0	X	0	0	0	is unchanged	testCheckVerticalWin_play_on_end			
2		X								
3		X								
4		Х								
last	Pos.	σetF	SOW	= 1						
	Pos.	_			= 1					
	/er =	_			_					
l	,									
Inp	ut:					Output:	Reason:			
						checkVerticalWin	This case is unique because it is a win resulting from a			
Stat	:e: (r			1	4)	= true	play in the middle of the winning string, meaning the			
	0	1	2	3	4		function must check both above and below the last			
0						state of the board	played marker			
1	0	Х	0	0	0	is unchanged	Function:			
2		X					testCheckVerticalWin play in middle			
3		X					cestemeekverereaiwin_piay_in_miaare			
4		Χ								
lact	Pos.	σ _Φ +F	2014	- 2						
	Pos.	_			= 1					
	/er =	_			_					
	'									
Inp	ut:					Output:	Reason:			
						checkVerticalWin	This case is unique because it tests that the function			
Stat	:e: (r					= true	can recognize a win with a full board			
l 🛌	0	1	2	3	4					
0	0	0	X	X	0	state of the board	Function:			
1	X	X	0	X	X	is unchanged	testCheckVerticalWin_full			
2	0	0	X	X	0					
3	X O	X O	O X	X O	X O					
4	U	0	_ ^	U	U					
last	Pos	get F	Row	= 1						
lastPos.getRow = 1 lastPos.getColumn = 3										
	/er =		•••	•	-					

Input:						Output: checkVerticalWin	Reason: This case is unique because it is a win where the last
Stat	te: (ı	num	ToV	اا	= 4)	= true	played token is in the corner of the board. The
	0	1	2	3	4		function must not access out of bounds when
0	Χ					state of the board	attempting to check for markers above the last played
1	Χ		0			is unchanged	marker
2	Χ			0			
3	Χ				0		Function:
4							testCheckVerticalWin_corner_win
lastPos.getRow = 0 lastPos.getColumn = 0 player = 'X'							

checkDiagonalWin(BoardPosition lastPos, char player)

Input:	Output: checkDiagonal	Reason: This case is unique because it is a win resulting from a play
State: (numToWin =	Win = true	on the upper right end of the winning string
4) 0 1 2 3 4 0 X X 1 X O 2 X O 3 X O 4 O	state of the board is unchanged	Function: testCheckDiagonalWin_upper_right_win
lastPos.getRow = 0 lastPos.getColumn = 3 player = 'X'		
Input:	Output:	Reason:
State: (numToWin = 4)	checkDiagonal Win = true	This case is unique because it is a win resulting from a play on the lower left end of the winning string
0 1 2 3 4 0 X X 1 X O 3 X O 4 O	state of the board is unchanged	Function: testCheckDiagonalWin_lower_left_win
lastPos.getRow = 3 lastPos.getColumn = 0 player = 'X'		
Input:	Output:	Reason:
State: (numToWin = 4)	checkDiagonal Win = true	This case is unique because it is a win resulting from a play on the upper left end of the winning string
1 X O O O O O O O O O O O O O O O O O O	state of the board is unchanged	Function: testCheckDiagonalWin_upper_left_win
lastPos.getRow = 1		

lastPos.getColumn =		
0 player = 'X'		
Input: State: (numToWin = 4) 0 1 2 3 4 0 1 X 0 2 X 0 3 X 0 4 X X lastPos.getRow = 4 lastPos.getColumn = 3 player = 'X'	Output: checkDiagonal Win = true state of the board is unchanged	Reason: This case is unique because it is a win resulting from a play on the lower right end of the winning string Function: testCheckDiagonalWin_lower_right_win
Input: State: (numToWin = 4)	Output: checkDiagonal Win = true state of the board is unchanged	Reason: This case is unique because it is a win resulting from a play in the middle of the winning string where both the upper left and lower right diagonals must be checked Function: testCheckDiagonalWin_upper_left_lower_right_win
Input:	Output: checkDiagonal Win = true	Reason:

State: (numToWin = 4) 0 1 2 3 4 0	state of the board is unchanged	This case is unique because it is a win resulting from a play in the middle of the winning string where both the lower left and upper right diagonals must be checked Function: testCheckDiagonalWin_lower_left_upper_ri		
2 X O 3 X O 4 X O 4 X O IastPos.getRow = 2 IastPos.getColumn = 2 player = 'X'		ght_win		
p. 17 2.				
Input: State: (numToWin = 4)	Output: checkDiagonal Win = false	Reason: This case is unique because it is a routine check on the last played marker where there is not enough markers in a row.		
0 1 2 3 4 0	state of the board is unchanged	Function: testCheckDiagonalWin_no_win		

Input:						Output:	Reason:
State:						checkForDraw =	This case is unique because it is a routine scenario
Star	te: 0	1	2	3	4	false	where there is no draw
0	U	1		3	4	state of the board	Function:
1	0	Х	0	0	0	is unchanged	testCheckForDraw no draw
2		Х					
3		Х					
4		Х					
mai	rkers	sPlay	/ed :	= 7			
Inp	ut:					Output:	Reason:
						checkForDraw =	This case is unique because it is a routine check that
Stat	1	_	_	_		true	the number of markers played is equal to the spots
	0	1	2	3	4	state of the board	available on the board
0	X	0	X	0	X	is unchanged	Function:
2	X	0	X	0	Х	13 differialiged	testCheckForDraw draw
3	0	Х	0	Х	0		
4	0	X	0	X	0		
-	U	^	0				
mai	rkers	Play	/ed =	= 25			
Inp	ut:					Output:	Reason:
						checkForDraw =	This case is unique because it is a check of the board
Stat	1			1		false	where all spaces are filled except one. The function
	0	1	2	3	4		should recognize that there is still an available space
0	X	0		0	X	state of the board	therefore there is no draw
1	X	0	X	0	X	is unchanged	Function:
2	X	0	X O	0	Х		testCheckForDraw near draw
3	0	X		X	0		cosconconforbran_near_aran
4	0	۸	U	۸	U		
markersPlayed = 24							
Inn	ut:					Output:	Reason:
IIIP						checkForDraw =	This case is unique because it is a check for draw on
Stat	te:					falsa	an empty board. It should recognize there is not a
_	te: 0	1	2	3	4	false	
_	1	1	2	3	4		draw
Stat 0 1	1	1	2	3	4	state of the board	draw
0 1 2	1	1	2	3	4		draw Function:
Stat 0 1 2 3	1	1	2	3	4	state of the board	draw
0 1 2	1	1	2	3	4	state of the board	draw Function:

Input:	Output:	Reason:
	whatsAtPos = ' '	This case is unique because it is a routine check of a
State: (numToWin = 4)		blank space
0 1 2 3 4	state of the board	
0	is unchanged	Function:
1		testWhatsAtPos_blank_space
2		
3		
4		
pos.getRow = 1		
pos.getColumn = 1		
Input:	Output:	Reason:
	whatsAtPos = 'X'	This case is unique because it is a routine check of a
State: (numToWin = 4)		space that has been played on
0 1 2 3 4	state of the board	
0	is unchanged	Function:
1 X		testWhatsAtPos_non_edge_taken_space
2		
3		
4		
pos.getRow = 1		
pos.getColumn = 1		
Input:	Output:	Reason:
input:	whatsAtPos = 'X'	
State: (numToWin = 4)	WildtsAtros - X	This case is unique because it is a boundary check on the corner of the board
0 1 2 3 4	state of the board	the corner of the board
0 X 3 4	is unchanged	Function:
	is anchanged	testWhatsAtPos taken corner space
2		+
3		
4		
pos.getRow = 0		
pos.getColumn = 0		
F-23-02-23-141111		

State: (numToWin = 4)	Output: whatsAtPos = '' state of the board is unchanged	Reason: This case is unique because it checks that the function recognizes there is an open space left and that this spot is not taken Function: testWhatsAtPos_almost_full_board
Input: State: (numToWin = 4) 0 1 2 3 4 0 X X 0 0 X 1 0 X X 0 0 2 0 0 X X 0 3 X 0 X 0 X 4 X 0 0 X 0 Dos.getRow = 1 pos.getColumn = 3	Output: whatsAtPos = 'X' state of the board is unchanged	Reason: This case is unique because it checks that the function can return the correct character on a full board Function: testWhatsAtPos_full_board

isPlayerAtPos(Boar		
Input:	Output:	Reason:
	isPlayerAtPos	This case is unique because it is a routine check of a blank
State: (numToWin = 4)	= false	space
0 1 2 3 4		
0	state of the	Function:
1	board is	testIsPlayerAtPos blank space
2	unchanged	+
	unchangea	
3		
4		
pos.getRow = 1		
pos.getColumn = 1		
player = 'X'		
1 / -		
Input:	Output:	Reason:
	isPlayerAtPos	This case is unique because it is a routine check of a non-
State: (numToWin = 4)	= true	edge space that has been played on
	- true	euge space that has been played on
0 1 2 3 4		
0	state of the	Function:
1 X	board is	testIsPlayerAtPos_non_edge_taken_space
2	unchanged	
3		
4		
4		
pos.getRow = 1		
pos.getColumn = 1		
player = 'X'		
Input:	Output:	Reason:
	isPlayerAtPos	This case is unique because it is a boundary check on the
State: (numToWin = 4)	= true	corner of the board
0 1 2 3 4	0.0.0	
	state of the	Function:
1	board is	testIsPlayerAtPos_taken_corner_space
2	unchanged	
3		
4		
pos.getRow = 0		
pos.getColumn = 0		
. •		
player = 'X'		
	i .	

	1	
Input: State: (numToWin = 4) 0 1 2 3 4 0 X X 0 0 X 1 0 X X 0 0 2 0 0 X 0	Output: isPlayerAtPos = false state of the board is unchanged	Reason: This case is unique because it checks that the function recognizes there is an open space left and that this spot is not taken Function: testIsPlayerAtPos open almost full board
2 O O X O O X O X O X O X O X O X O X O X O O	unenangeu	
Input:	Output:	Reason:
	isPlayerAtPos	This case is unique because it checks that the function can
State: (numToWin = 4)	= true	return the correct character on a full board
0 1 2 3 4 0 0 X 0 X 0 1 X 0 X 0 X 2 0 X 0 X 0 3 X 0 X 0 X 4 0 X 0 X 0 pos.getRow = 2 pos.getColumn = 3	state of the board is unchanged	Function: testIsPlayerAtPos_full_board

Input:	Output:	Reason:
State: (numToWin = 4)	State: (numToWin = 4)	This case is unique because it places a marker on the board in the last available spot to play on (making the board full) Function: testPlaceMarker_fill_board
pos.getColumn = 3 player = X	pos.getColumn = 3 player = X	
State: (numToWin = 4)	Output: State: (numToWin = 4) 0 1 2 3 4 0 0 0 A 1 0 A A 2 X X A 3 0 0 A 4 0 0 A	Reason: This case is unique because it places a marker of a third player that has not yet been played Function: testPlaceMarker_adding_third_play er
State: (numToWin = 4)	Output: State: (numToWin = 4) 0 1 2 3 4 0 0 0 1	Reason: This case is unique because it is a routine placement on a corner space Function: testPlaceMarker_edge

In	out:					Out	put	:				Reason:
												This case is unique because it places a marker in
St	State: (numToWin = 4)						State: (numToWin = 4)				= 4)	the last spot available in a column
	0	1	2	3	4		0	1	2	3	4	
C						0		0				Function:
1		Х				1		Χ				testPlaceMarker_fill_column
2		0				2		0				
3		Х				3		Χ				
4		Х				4		Χ				
m	arke	r.getl	Row	= 0								
		r.get(= 1							
	ayer											
'	•											
In	out:					Out	put	:				Reason:
In	out:					Out	put	:				Reason: This case is unique because it places a marker in
		(num	ıToV	Vin =	= 4)				ıToV	Vin =	= 4)	
		(num	ToV 2	Vin =	4)				ToV	Vin =	= 4)	This case is unique because it places a marker in
	ate:						:e: (ı	num				This case is unique because it places a marker in
St	ote:	1				Stat	:e: (ı	num				This case is unique because it places a marker in the last spot available in a row
St	ote:	1	2	3		Stat 0	e: (ı	num 1	2	3	4	This case is unique because it places a marker in the last spot available in a row Function:
St C	ote:	1	2	3		Stat 0 1	e: (ı	num 1	2	3	4	This case is unique because it places a marker in the last spot available in a row Function:
St C	ote:	1	2	3		Stat 0 1 2	e: (ı	num 1	2	3	4	This case is unique because it places a marker in the last spot available in a row Function:
St 0	ote:	1	2	3		Stat 0 1 2 3	e: (ı	num 1	2	3	4	This case is unique because it places a marker in the last spot available in a row Function:
St	ate:	1 X	0	0		Stat 0 1 2 3	e: (ı	num 1	2	3	4	This case is unique because it places a marker in the last spot available in a row Function:
St	ate: 0 X	X X	2 O	3 O = 1	4	Stat 0 1 2 3	e: (ı	num 1	2	3	4	This case is unique because it places a marker in the last spot available in a row Function:
St C 1 2 3 4 m m	ate: 0 X	X X	2 O	3 O = 1	4	Stat 0 1 2 3	e: (ı	num 1	2	3	4	This case is unique because it places a marker in the last spot available in a row Function: