Griffin Smith CPSC 2150 March 24, 2021

Requirements Analysis System Requirements and Operation

- As a player, I have a device with java 11 installed so that the program will be able to run
- As a player, I can not place the token outside of the board so that my turn will not result in an invalid token.
- As a player, I can not place the token in a full column.
- As a player, I can run the program on this device by following the instructions under deployment As a player, I can not input a desired number of rows outside of 3 and 100
- As a player, I can not input a desired number of Columns outside of 3 and 100
- As a player, I can not input a desired number in a row to win outside of 3 and 25
- As a player, I can not input a desired number of players outside of 2 and 10
- Program runs on Linux
- 0,0 is the bottom left of the gameboard
- GameBoard is a 100x100 or smaller
- GamerBoard is a 3x3 or larger

Game Start

- The user will input the desired rules (players, tokens, columns, rows, number to win)
- Players will take turns placing their respected tokens in a column of their choice
 - o Repeats until the game board is full (tie) or there is a winner.
- There will be a congratulating message for the winner
- The user will be asked if they want to play again
- The user must input 'Y' for yes or 'N' for no.

Deployment

- User will open a terminal
- User will navigate to the correct location of the program in the terminal.
- User will compile program by inputting <make>.
- User will run program by inputting <make run>.
- User will delete .class files by inputting <make clean>
- User will compile test cases by inputting <make test>
- User will run test cases by inputting <make testGB> or make<testGBmem>

Class Diagrams

GameScreen.java

- + field: public
- +main(String[] args): void

BoardPosition.java

- + row: int
- + column: int
- + BoardPosition(int row, int column): Board Position + getRow(void): int +getColumn(void): int

- + equals(BoardPosition first, BoardPosition last): boolean + toString(String board): String

GameBoard.java

- + board: char[][] NUMROWS: final int NUMCOLUMNS: final int
- NUMTOWIN: final int

- + GameBoard(constructor) + placeToken(char p, int c): void + whatsAtPos(BoardPosition pos): char +getNumRows(): int
- +getNumColumns(): int +getNumToWin(): int

IGameBoard.java

- NUM_ROWS: final int
- NUM_COLUMNS: final int NUM_TOWIN: final int
- + checklfFree(int c): boolean
- + checkForWin(int c): boolean + checkTie(void): boolean + placeToken(char p, int c): void
- + place foken(char p, int c). Void + checkHorizWin(BoardPosition pos, char p): boolean + checkDertWin(BoardPosition pos, char p): boolean + checkDiagWin(BoardPosition pos, char p): boolean + whatsAtPos(BoardPosition pos): char + isPlayerAtPos(BoardPosition pos, char player): boolean

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

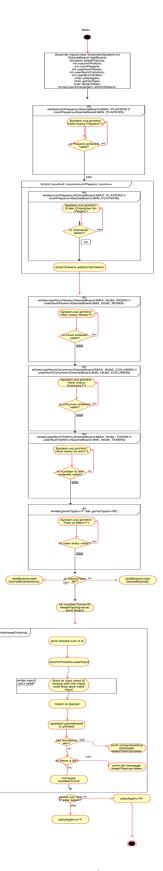
AbsGameBoard.java

+toString(): String

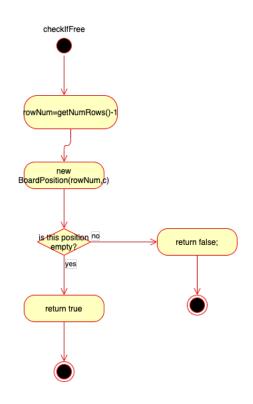
GameBoardMem.java

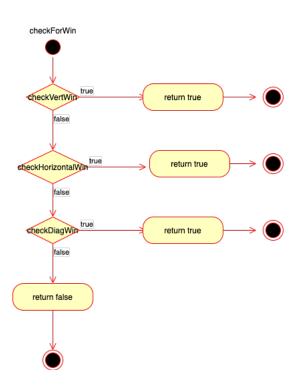
- NUMROWS: final int NUMCOLUMNS: final int
- NUMTOWIN: final int
- + GameBoard(constructor) + placeToken(char p, int c): void + whatsAtPos(BoardPosition pos): char
- +getNumRows(): int +getNumColumns(): int +getNumToWin(): int

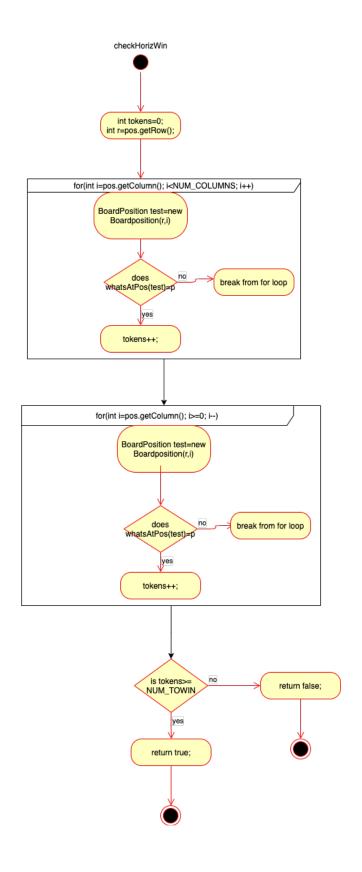
GameScreen.java

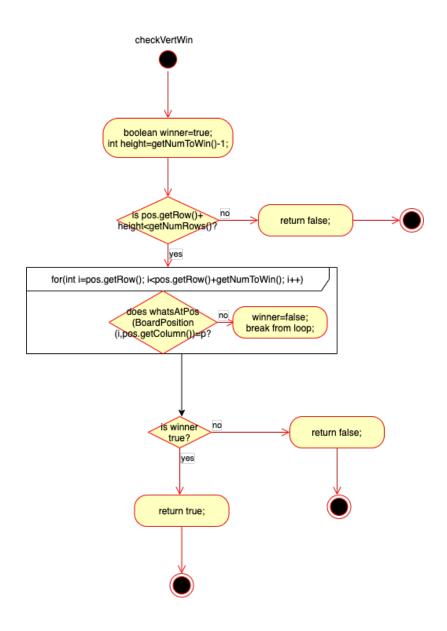


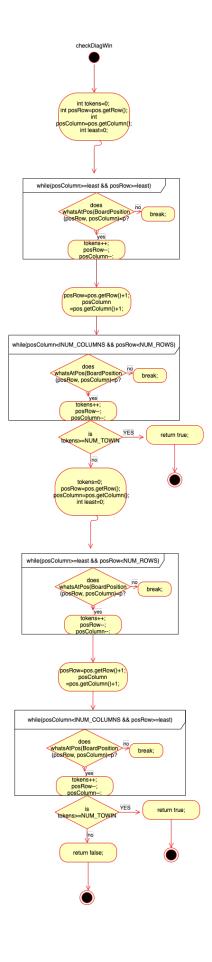
IGameBoard.java

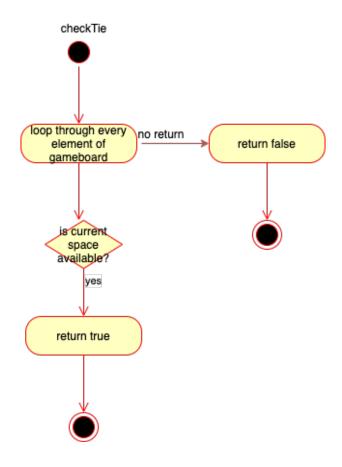


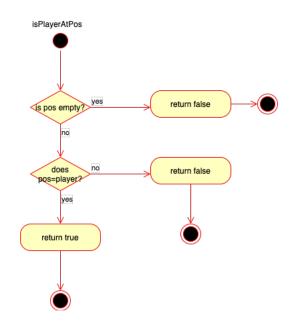




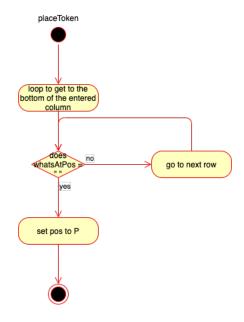


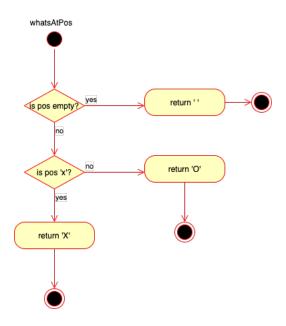


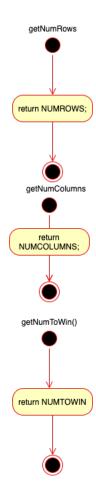




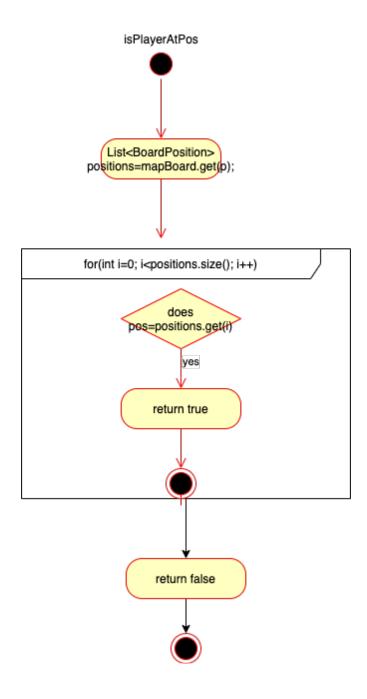
GameBoard.java

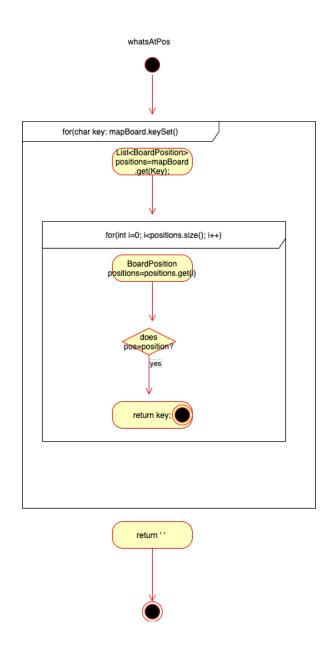


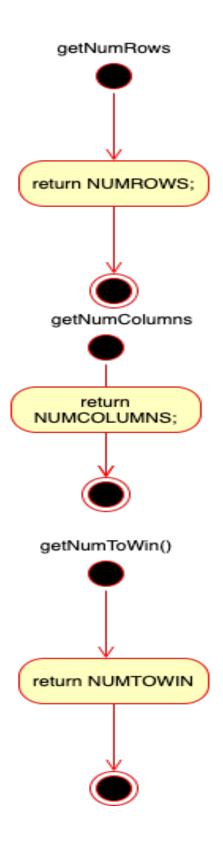




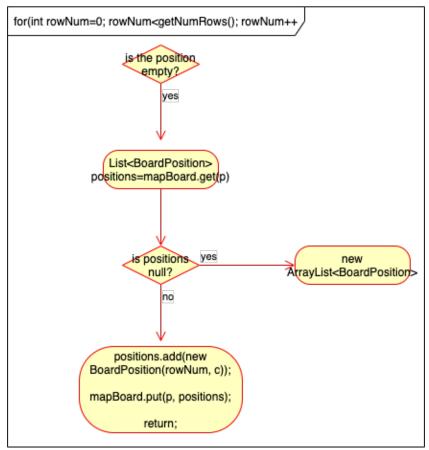
GameBoardMem.java







placeToken



Test public GameBoard(int userNumRows, int unserNumColumns, int userNumToWin)

Input:	Outpu		Reason:
State:	The output is a		This test case is unique and distinct
userNumRows=100	gameBoa		because it tests for the constructor
userNumColumns=100	(Not Able to	insert	to construct the maximum size of the
userNumToWin=25	100x100 emp	ty table)	game board allowed(100x100)
	The table is er	npty and	Function Name:
	every position	on is ' '	MakeGameBoard_Max_SizeBoard()
Input:	Outpu	t:	Reason:
State:			This test case is unique and distinct
userNumRows=3			because it tests for the constructor
userNumColumns=3			to construct the minimum size of the
userNumToWin=3	The 3x3 table	is empty	game board allows(3x3)
	and every pos	ition is ''	Function Name:
			MakeGameBoard_Min_SizeBoard()
Input:	Outpu	t:	Reason:
State:	X Y X	ΥX	This test case is unique and distinct
userNumRows=5	Y X Y	Ху	because it tests for the constructor
userNumColumns=5	XYX	Yx	to construct the game board and
userNumToWin=4	YXY	XY	tests if tokens can be placed
	XYX	YX	Function Name:
	The 3x3 table	is empty	MakeGameBoard_WholeBoard_XY()
	and every pos	ition is ' '	

Boolean checkIfFree(int c)

Input:	Output:	Reason:
State:	checkIfFree=false	This test case is unique and
C=0		distinct because it fills the left
X	state of the board is	most column on the board
X	unchanged	with tokens and tests
X		checkIfFree on the full column
X		while the other columns are
X		empty
		Function Name:
		test_checkIfFree_LeftColumn()
Input:	Output:	Reason:
State:	checkIfFree=false	This test case is unique and
C is looped from 0 to 4		distinct because it fills the
X X X X X	state of the board is	whole board with Xs and tests
X X X X X	unchanged	checkIfFree on every column
X X X X X		on the gameboard to make
X X X X X		sure every column is false

хх	ХХ	х			Function Name:
					test_checkIfFree_LeftColumn()
ı	nput:			Output:	Reason:
9	State:			checkIfFree=true	This test case is unique and
C is loope	ed from	0 to 4	_		distinct because it keeps the
				state of the board is unchanged	board empty and tests every column to make sure checkIfFree is true for every column on a new gameboard Function Name: test checkIfFree LeftColumn()

:-\A/:--/Da dDocitio

				Воо	lean checkHorizWin(BoardF	Position pos, char p)			
	ı	nput	:		Output:	Reason:			
	:	State	:		checkHorizWin=true	This test case is unique and distinct			
ļ ķ	os.g	etRo	w()=4	1		because it fills the whole board with Xs			
рс	s.get	tColu	mn()	=4	state of the board is	and tests checkHorizWin on the top			
		p='X'	1		unchanged	right position of the gameboard for a			
X	X	X	X	X		horizontal win.			
X	Х	Х	Х	Х		Function Name:			
Х	Х	Х	Х	Х		test_checkHorizWin_FullBoard			
X	Х	Х	Х	Х					
Х	Х	Х	Х	х					
		nput	:		Output:	Reason:			
		State	:		checkHorizWin=false	This test case is unique and distinct			
(ro	w an	d col	umn	are		because it fills the board with 1 less			
loc	ped	to te	st eve	ery	state of the board is	than the number to win in a row. This			
pos	sition	0-4 ו	mear	ıs O	unchanged case tests every position on the boar				
thro	ough	4 thr	ough	out		to make sure there is no horizontal win			
eve	ery lo	op it	erati	on)		in any position.			
p	os.ge	tRow	v()=0-	-4		Function Name:			
pos	.get(Colun	nn()=	0-4		test_checkHorizWin_OneShort_ToWin			
	num	ToW	in=4						
		p='X'							
X	Х	X							
X	Х	Х							
X	Х	X							
X	Х	X							
X	Х	X							
	ı	nput	:		Output:	Reason:			
	:	State	:		checkHorizWin=true	This test case is unique and distinct			
ļ ķ	oos.g	etRo	w()=0)		because it fills the bottom row of the			
pos	_	Colun	• • • • • • • • • • • • • • • • • • • •	0-4	state of the board is	board with Xs in a row. This tests every			
	num	ToW	in=4		unchanged	single position on the bottom row to			

		p='X'				assert true that there is a horizontal win in every position
						Function Name:
						test_checkHorizWin_BottomRow_Win
X	Х	X	X	Х		
	ı	nput	:		Output:	Reason:
	9	State	:		checkHorizWin=false	This test case is unique and distinct
р	os.ge	tRow	/()=O-	-4		because it tests an empty gameboard
pos	.get(Colun	nn()=	0-4	state of the board is	for a horizontal win. This test case
	num	ToW	in=4		unchanged	tests every single position on the game
		p='X'				board to determine that they are false
						since there are no tokens on the board
						Function Name:
						test_checkHorizWin_Empty()

Boolean checkVertWin(BoardPosition pos, char p)

				D	Dolean checkvertwintboar	arosition pos, char pj			
	I	nput	:		Output:	Reason:			
	9	State	:		checkVertWin=true	This test case is unique and distinct			
po	os.ge	tRow	<mark>/()=</mark> 0	-4		because it fills the whole board with Xs			
pos	.get(Colun	nn()=	0-4	state of the board is	and tests checkVertWin on every single			
		p='X'	1		unchanged	position on the board to determine that			
X	Х	X	X	X		checkVertWin works on every position.			
Х	X	X	X	X		Function Name:			
Х	Х	Х	X	Х		test_checkVertWin_FullBoard			
Х	Х	Х	X	Х					
Х	Х	Х	X	х					
	I	nput	:		Output:	Reason:			
	9	State	:		checkVertWin=true	This test case is unique and distinct			
ро	os.ge	tRow	<mark>/()=</mark> 0	-4		because it fills the left most column with			
ро	s.get	Colu	mn()	=0	state of the board is the same token. It tests each position i				
	num	ToW	in=4		unchanged	the left most column to determine that			
		p='X'				checkVertWin works on each of these			
X						positions.			
X						Function Name:			
Х						test_checkVertWin_FirstColumn_Win			
Х									
Х									
	ı	nput	:		Output:	Reason:			
	9	State	:		checkVertWin=false	This test case is unique and distinct			
þ	os.g	etRo	w()=	0		because it fills the board with 1 less than			
pos	.get(Colun	nn()=	-0-4		the number to win in a row vertically.			

	num	ToW	in=4		state of the board is	This case tests every position on the
		p='X'	,		unchanged	board to make sure there is no vertical
						win in any position.
						Function Name:
Χ	Х	Х	х	Х		test_checkVertWin_OneLess_NumToWin
Χ	Х	Х	х	Х		
X	Χ	X	Х	Х		
	I	nput	::		Output:	Reason:
	9	State	:		checkVertWin=false	This test case is unique and distinct
рс	s.ge	tRov	v()=0	-4		because it tests an empty gameboard for
pos	.get(Colur	nn()=	-0-4	state of the board is	a vertical win. This test case tests every
	num	ToW	/in=4		unchanged	single position on the game board to
		p='X'	,			determine that they are false since there
						are no tokens on the board
						Function Name:
						test checkVertWin Empty()
						, ,,
	X X x	X X X X X X X X X X X X X X X X X X X	x x x X X X x X X Input State pos.getRov pos.getColur numToW	p='X' X x x x X X X X X X X X Input: State: pos.getRow()=0 pos.getColumn()=	X X X X X X X X X X X X X X X X X X X	p='X' unchanged X x x X X X X X X X x X X X X x X X X X

Boolean checkDiagWin(BoardPosition pos, char p)

		In	put:			Output:	Reason:
		St	tate:			checkDiagWin=t	This test case is unique and distinct
	р	os.ge	tRow	()=0		rue	because it fills the whole board with Xs
	pos	s.get(Colum	n()=0)		and tests checkDiagWin in the bottom
		numT	oWir	า=4		state of the	left to top right direction. It does this
		р	='X'		•	board is	by calling checkDiagWin on the
X	X	X	X	X		unchanged	0,0(bottom left) position.
X	X	X	X	X			Function Name:
X	Х	Х	Х	Х			test_checkDiagWin_BottomLeft_UpRi
Х	Х	Х	Х	Х			ght
Х	Х	Х	Х	х			
		In	put:			Output:	Reason:
		St	tate:			checkDiagWin=t	This test case is unique and distinct
ŗ	os.ge	etRow	v()=4(top r	ow)	rue	because it fills the whole board with Xs
	pos	s.get(Colum	n()=0)		and tests checkDiagWin in the top left
	(pos i	s top	left p	ositio	on)	state of the	to bottom right direction. It does this
		р	='X'			board is	by calling checkDiagWin on the 4,0(top
		numT	oWir	า=4	•	unchanged	left) position.
X	X	Х	Х	Х			Function Name:
X	X	X	X	X			test_checkDiagWin_TopLeft_BottomR
X	Х	X	Х	Х			ight
Х	Х	Х	Х	Х			
X	Х	Х	Х	Х			
		In	put:			Output:	Reason:

Г			C.	tatai			shockDiagMin_t	This test ease is unique and distinct
	_	200 00		tate: -/_4/	ton r	ow)	checkDiagWin=t	This test case is unique and distinct because it fills the whole board with Xs
	1	os.ge		., .	•	•	rue	
	,	pos is	_	Colum	• • •		state of the	and tests checkDiagWin in the top
	(•	•	ngnı ToWir	•	ion)	state of the board is	right to bottom bottom direction. It
					1=4			does this by calling checkDiagWin on
1 6	· · ·	V	•	='X'	V		unchanged	the 4,4(top right) position.
-	X	X	X	X	X			Function Name:
-	X	X	X	X	X			test_checkDiagWin_TopRight_Bottom
-	X	X	X	X	X			Leit
-	X	X	X	X	X			
	Х	X	Х	Х	X		_	_
				put:			Output:	Reason:
				tate:			checkDiagWin=t	This test case is unique and distinct
	k	os.ge		., .	•	•	rue	because it fills the whole board with Xs
		•	_	Colum	.,			and tests checkDiagWin in the bottom
	(pc			_	•	sition)	state of the	right to top left direction. It does this
		İ	-	oWir	1=4		board is	by calling checkDiagWin on the
l	p='X'						unchanged	0,4(bottom right) position.
-	Х	X	Х	Х	Х			Function Name:
-	Х	Х	Х	Х	Х			test_checkDiagWin_BottomRight_Top
	Х	Х	Х	Х	Х			Left
-	X	X	X	X	Х			
۲	Х	Х	Χ.	X	Х			_
	^	_ ^	In	put:	Х		Output:	Reason:
<u> </u>	^_		In St	put: tate:			checkDiagWin=t	This test case is unique and distinct
	^	р	In Stos.ge	put: tate: tRow	()=2		<u> </u>	This test case is unique and distinct because it fills the whole board with Xs
		p pos	In Stos.ge os.get(i put: tate: tRow Colum	()=2 nn()=2		checkDiagWin=t rue	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle
		p pos (pos i	In Stos.ge os.get(is mid	iput: tate: tRow Colum	()=2 nn()=2 ositio		checkDiagWin=t rue state of the	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that
		p pos (pos i	In Si os.ge s.get(is mid num]	tate: tRow Colum ddle p	()=2 nn()=2 ositio		checkDiagWin=t rue state of the board is	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even
		p pos (pos	In Sfos.ge os.get(is mid num]	tate: tRow Colum ddle p oWir	()=2 nn()=2 nosition=4		checkDiagWin=t rue state of the	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways
	X	pos (pos	In Si os.ge s.get(is mid num] p	tate: tRow Colum ddle p oWir ='X' X	()=2 nn()=2 nosition=4 X		checkDiagWin=t rue state of the board is	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position.
	X X	p pos (pos	In Si os.get(is mid num1 p X	tate: tate: tRow Colum ddle p oWir ='X' X	()=2 nn()=2 nosition=4 X		checkDiagWin=t rue state of the board is	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name:
	X X X	p pos (pos i	In Si os.get s.get is mid num p X X	tate: tRow Colum ddle p ToWir ='X' X X	()=2 nn()=2 nosition=4 X X		checkDiagWin=t rue state of the board is	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name: test_checkDiagWin_FullBoard_TestMi
	X X X	p pos (pos	In Si os.get s.get is mid num p X X X	tate: tate: tate: ddle p oWir ='X' X X	()=2 nn()=2 nosition=4 X X X		checkDiagWin=t rue state of the board is	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name:
	X X X	p pos (pos i	In Si os.ge s.get(is mid num1 p X X X X	aput: tate: tRow Colum ddle p ToWir ='X' X X X	()=2 nn()=2 nosition=4 X X		checkDiagWin=t rue state of the board is unchanged	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name: test_checkDiagWin_FullBoard_TestMidlePos
	X X X	p pos (pos	In Si os.get s.get(is mid num1 p X X X X	tate: tate: tate: ddle p oWir ='X' X X X X put:	()=2 nn()=2 nosition=4 X X X		checkDiagWin=t rue state of the board is unchanged Output:	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name: test_checkDiagWin_FullBoard_TestMidlePos
	X X X	p pos (pos i	In Si os.ge s.get(is mid num1 p X X X X In Si	tate: tate: tRow Colum ddle p ToWir ='X' X X X X x put: tate:	()=2 nn()=2 nosition=4 X X X X		checkDiagWin=t rue state of the board is unchanged Output: checkDiagWin=t	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name: test_checkDiagWin_FullBoard_TestMidlePos Reason: This test case is unique and distinct
	X X X	p pos (pos i	In Si os.ge s.get(is mid numT p X X X X In Si os.ge	tate: tRow Colum ddle p oWir ='X' X X X X put: tate:	()=2 nn()=2 nosition=4 X X X X	on)	checkDiagWin=t rue state of the board is unchanged Output:	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name: test_checkDiagWin_FullBoard_TestMidlePos Reason: This test case is unique and distinct because it fills the whole board with Xs
	X X X X	p pos (pos i	In Si os.ge s.get(is mid num1 p X X X In Si os.ge	tate: tate: tate: tate: ddle p oWir ='X' X X X X put: tate: tRow Colum	()=2 nn()=2 nosition=4 X X X X X x	on)	checkDiagWin=t rue state of the board is unchanged Output: checkDiagWin=t rue	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name: test_checkDiagWin_FullBoard_TestMidlePos Reason: This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin when the
	X X X X	p pos (pos i	In Si os.ge s.get(is mid num1 p X X X X In Si os.get(is) sotto	tate: tRow Colum ddle p ToWir ='X' X X X x put: tate: tRow Colum m lef	()=2 nn()=2 nosition=4 X X X X X x	on)	checkDiagWin=t rue state of the board is unchanged Output: checkDiagWin=t rue state of the	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name: test_checkDiagWin_FullBoard_TestMidlePos Reason: This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin when the number to win is equal to the number
	X X X X	p pos (pos i	In Si os.get s.get(is mid num] X X X X In Si os.get botto	tate: tate: tate: ddle p oWir ='X' X X X X put: tate: tRow Colum m lef	()=2 nn()=2 n=4 X X X X X x ()=0 nn()=0	on)	checkDiagWin=t rue state of the board is unchanged Output: checkDiagWin=t rue state of the board is	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name: test_checkDiagWin_FullBoard_TestMidlePos Reason: This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin when the number to win is equal to the number of rows and number of columns in the
	х х х х	p pos (pos i	In Sicos.ge s.get(sis mice num1 p X X X X In Sicos.get(sis get(sis get	tate: tRow Colum ddle p ToWir ='X' X X X X put: tate: tRow Colum m lef ='X' oWin	()=2 nn()=2 n=4 X X X X x ()=0 nn()=0	on)	checkDiagWin=t rue state of the board is unchanged Output: checkDiagWin=t rue state of the	This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin in the middle of the gameboard. This shows that checkDiagWin works correctly even when there are multiple winning ways diagonally on a given position. Function Name: test_checkDiagWin_FullBoard_TestMidlePos Reason: This test case is unique and distinct because it fills the whole board with Xs and tests checkDiagWin when the number to win is equal to the number

Х	Х	Х	Х	Х	Х	Х	Х	Х	Х		Function Name:
Х	Х	Х	Х	X	Х	Х	X	X	Х		test_checkDiagWin_WinNum_
Х	Х	Х	X	X	X	Х	X	X	Х		EqualBoardSize_FullWin
X	Х	Х	Х	Х	Х	Х	Х	X	Х		
X	Х	X	X	Х	X	X	X	X	X		
X	X	X	X	X	X	X	X	X	X		
X	X	X	X	Χ	X	X	X	X	X		
X	X	X	X	X	X	X	X	X	X		
X	X	X	X	Χ	Χ	X	X	X	X		
				Inp	ut:					Output:	Reason:
				Sta	te:					checkDiagWin=	This test case is unique and distinct
		рс	s.g	etR	ow(()=0	-4			false	because it fills the whole board with Xs
		pos	.get	Col	um	n()=	-0-4	ŀ			and Ys in a checker board pattern. This
			nun	nTo	Wi	n=4				state of the	case tests every position on the game
				p=	'X'					board is	board to determine that no positions
	2	Χ	0	١	1	Q	٧	V		unchanged	result in a diagonal win.
	X O Y Q W						٧	٧			Function Name:
		X	0	١	1	Q	٧	٧			test_checkDiagWin_CheckPattern_No
		Χ	0	١	1	Q	٧	V			Win
)	X	0	\	/	Q	٧	٧			

Boolean checkTie()

					Doorcan check	
		In	put:		Output:	Reason:
		St	tate:		checkTie=true	This test case is unique and distinct
						because it fills the whole board with Xs
X	Х	X	Х	X	state of the	in order to make the board full and
X	X	X	Х	X	board is	then it calls checkTie to confirm that
X	Х	Х	Х	Х	unchanged	the board is full, therefore there is a
X	Х	Х	Х	Х		tie.
Х	Х	Х	Х	Х		Function Name:
			ı			test_checkTie_Full_Board
		In	put:		Output:	Reason:
		St	tate:		checkTie=false	This test case is unique and distinct
						because it fills the whole board with Xs
X	X	X	X		state of the	except for one position. Then it tests
X	X	X	X	X	board is	checkTie in order to confirm that there
X	Х	X	Х	X	unchanged	is not yet a tie since a token can still be
X	Х	Х	Х	X		placed.
Х	Х	Х	Х	Х		Function Name:
	ı		I			test_checkTie_Full_Board
		In	put:		Output:	Reason:
		St	tate:		checkTie=false	This test case is unique and distinct
						because it fills the whole board with Xs
X	Х	X	X			except for one column. Then it tests

	1	1	—			1		1 1 1
X	Х	Х		X			state of the	checkTie in order to confirm that there
X	Х	X		X			board is	is not yet a tie since a token can still be
X	Х	Х		Χ			unchanged	placed in the open column. Confirms
Х	Х	Х		Х				that checkTie still works when a full
					ı			column is open.
								Function Name:
								test_checkTie_Missing_RightColumn
			Inp	ut			checkTie=true	Reason:
	us	erNu	mR	low	s=10	0		This test case is unique and distinct
	use	rNum	Со	lum	ns=1	00	state of the	because it fills the whole board with Xs
	us	erNu	mΤ	оW	/in=2	5	board is	in order to make the board full and
100)x100	game	ebo	ard	is fill	ed with	unchanged	then it calls checkTie to confirm that
	X and	Ys in	a c	hec	kerb	oard		the board is full, therefore there is a
		р	atte	ern.				tie. This test case is distinct because it
	(Not A	ble t	o in	ser	t 100	x100		fills up the maximum size
		empty table)						gameboard(100x100) confirming that
								no matter how large the size, the
								checkTie function will still work.
								Function Name:
								test_checkTie_Max_SizeBoard_Full

Char whatsAtPos(BoardPosition pos)

Chai WhatsAtros(Boardrosition pos)								
Input:	Output:	Reason:						
State:	whatsAtPos='O'	This test case is unique and distinct						
Pos.row=4		because it fills the whole board with Xs						
Pos.col=4	state of the except for one position. Then it							
(top right position)	board is	whatsAtPos to determine that the						
X X X X O	unchanged	function works for the position that is						
X X X X X		not an X in the top right position on						
X X X X X		the Gameboard.						
X X X X X		Function Name:						
X X X X X		test_whatsAtPos_Top_Right						
Input:	whatsAtPos=' '	Reason:						
State:		This test case is unique and distinct						
pos.getRow()=0-4	state of the	because it calls an empty gameboard.						
pos.getColumn()=0-4	board is	It uses whatsAtPos to determine that						
	unchanged	nchanged every position in the gameboard is						
		initialized to ''						
		Function Name:						
		test_whatsAtPos_EmptyBoard						
Input	whatsAtPos=X '	Reason:						
userNumRows=100								

		400		-t		
	IumColumn		state of the	This test case is unique and distinct		
	^r NumToWir	_	board is	because it uses a gameboard that is		
100x100 ga	ameboard is	s filled with	unchanged	the maximum size (100x100) and fills it		
	Xs.			with Xs. whatsAtPos is tested on every		
(Not Abl	le to insert	100x100		position to determine that each		
(empty table	e)		position = 'X'		
Pos	.getRow()=(0-99		Function Name:		
	etColumn()			test_whatsAtPos_Max_Size_FullBoard		
	.,					
	Input:		whatsAtPos=X '	Reason:		
	State:			This test case is unique and distinct		
pos	s.getRow()=	:0-4	state of the	because it uses a gameboard that is		
pos.g	getColumn()=0-4	board is the minimum size (3x3) and fills it			
Х	Х	Х	unchanged	Xs. whatsAtPos is tested on every		
Х	Χ	Х		position to determine that each		
Х				position = 'X'		
	l			Function Name:		
				test_whatsAtPos_Min_Size_FullBoard		
	Input:		whatsAtPos=X '	Reason:		
	State:			This test case is unique and distinct		
Pos.getRow	v()=0		state of the	because it does not fill the gameboard		
Pos.getColu	umn()=0		board is	all the way, but calls the function in		
хх			unchanged	the bottom left corner to determine		
хх	x x x x		_	that the function works when the		
хх	хх			gameboard is not full.		
хх	ХХ			Function Name:		
хх	хх			test_whatsAtPos_BottomLeftX		

Boolean isPlayerAtPos(BoardPosition pos, char p)

Input:						Output:	Reason:		
State:						isPlayerAtPos=true			
Pos.row=4							because it fills the whole board with Xs		
Pos.c	ol=4					state of the board	except for one position. Then it tests		
P='0	,					is unchanged	isPlayerAtPos to determine that the		
(top	(top right position)				_		function returns true for the position		
Х	Х	Х	Х	0			that is not an X in the top right position		
Х	X X X X X			on the Gameboard.					
Х	x x x x x			Function Name:					
Х	x x x x x				test_isPlayerAtPos_Top_Right				
X X X X X									
Input:						isPlayerAtPos=true	Reason:		
State:							This test case is unique and distinct		
Pos.getRow()=0				state of the board	because it does not fill the gameboard				
Pos.g	etCo	lumn	()=0			is unchanged	all the way, but calls the function in the		

P='X'							bottom left corner to determine that the			
x	Х	Х	Х		1		function works when the gameboard is			
$\frac{\lambda}{X}$	X	X	X		-		not full and that the function correctly			
-					1		identifies the correct player token.			
X	X	X	X		-		Function Name:			
X	X	Х	X		4					
X	X	Х	X				test_isPlayerAtPos_BottomLeftX			
		Inpu				isPlayerAtPos=false	Reason:			
		Stat					This test case is unique and distinct			
	pos.g	getRo	w()=	0-4		state of the board	because it calls an empty gameboard. It			
р	os.ge	tColu	ımn()=0-4		is unchanged	uses isPlayerAtPos to determine that			
		p='	X'		_		every position in the gameboard is			
							initialized to '' and not to a character			
							token			
							Function Name:			
					1		test_isPlayerAtPos_EmptyBoard			
		Inp	ut	1		isPlayerAtPos=true	Reason:			
l	userN	-		=100		,	This test case is unique and distinct			
us	userNumColumns=100					state of the board because it uses a gameboard that is t				
	userNumToWin=25					is unchanged maximum size (100x100) and fills it				
	100x100 gameboard is filled						Xs. isPlayerAtPos is tested on every			
	with Xs.						position to determine that each position			
(Not	(Not Able to insert 100x100						returns true.			
(npty 1					Function Name:			
	Pos.g			•			test_isPlayerAtPos_Max_Size_FullBoard			
	_		• • •	=0-99)		test_isi layerAti os_iviax_size_i aliboara			
'	J3.gct	.Coiui ('=2	.,	-0-93	,					
		F – 7	^							
		Inpu	ıt:			isPlayerAtPos=true	Reason:			
		Stat				,	This test case is unique and distinct			
	pos.getRow()=0-4				state of the board	because it uses a gameboard that is the				
pos.getColumn()=0-4				is unchanged	minimum size (3x3) and fills it with Xs.					
p='X'					isPlayerAtPos is tested on every position					
Х		<u>ү</u> Х	·	Х			to determine that each position returns			
				X			true.			
				X			Function Name:			
^	X X X					test_isPlayerAtPos_Min_Size_FullBoard				
						Void placeToken(ch	nar p, int c)			

Input:	Output:					Reason:		
State:						This test case is unique and distinct		
P='X'	X X X X X				Х	because I placed the X marker in every		
C=0-4	X X X X X				Х	position on the gameboard.		

	v		· ·		\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	Function Names	
	X	X	X	X	X	Function Name: test_placeToken_Full_Board	
	X					test_placeToken_ruii_board	
	X	Х	X	X	X		
Immust.						Dogger,	
Input: State:		U	utpu	ıt.		Reason: This test case is unique and distinct	
P='X'						because I placed the X marker in a	
C=1						column that is empty. This is also	
						unique because it is the first token	
						placed on the newly initialized board	
		Х				Function Name:	
						test_placeToken_newBoard	
Input:		0	utpı	ıt:		Reason:	
State:		_				This test case is unique and distinct	
P='X','Y','Z','W','Q'	Х	Υ	Z	W	Q	because I place different markers. This	
C=0-4	Х	Υ	Z	W	Q	shows that the function works when	
	Х	Υ	Z	W	Q	placing different markers.	
	Х	Υ	Z	W	Q	Function Name:	
	Х	Υ	Z	W	Q	test_placeToken_Full_BoardXYZWQ	
Input		0	utp	ut		Reason:	
userNumRows=100	Output is a 100x100					This test case is unique and distinct	
userNumColumns=100	gameboard					because I use a maximum sized board.	
userNumToWin=25	completely filled with					This shows that the function continues	
100x100 gameboard is filled	Xs					working throughout the length of the	
with Xs.						program, no matter how big a board	
(Not Able to insert 100x100						the user wants.	
empty table)						Function Name:	
C=0-99						test_placeToken_MaxBoard	
P='X'							
Input:)ııt»	ıt		Reason:	
State:	Output					This test case is unique and distinct	
C=0-2	X X X					because I use a minimum sized board.	
p='X'	X X X X X					This shows that the function continues	
					^	working throughout the length of the	
						program, no matter how small a board	
						the user wants.	
						Function Name:	
						test_placeToken_MinBoard	