1. GameBoard(int rows, int columns, int numToWin)

Input: 3, 3, 3	<pre>Output: getNumRows() = 3 getNumColumns() = 3 getNumToWin() = 3</pre>	Reason: Testing the minimum boundary of values to make sure they are properly assigned
State:	State of the board is unchanged	Function Name: constructorTest AssignmentMin()

2. GameBoard(int rows, int columns, int numToWin)

Input: 100, 100, 25 State:	<pre>Output: getNumRows()=100 getNumColumns()=100 getNumToWin() = 25</pre>	Reason: Testing the maximum value to make sure they are properly assigned
	State of the board is unchanged	Function Name: constructorTest AssignmentMax()

3. GameBoard(int rows, int columns, int numToWin)

Input: 8, 8, 4 State:	<pre>Output: getNumRows() =8 getNumColumns() =8 getNumToWin() = 4</pre>	Reason: Testing a normal value (Normal defined as not a boundary) to make sure they are properly assigned
	State of the board is unchanged	
		Function Name: constructorTest AssignmentNormal()

Inp									Ou	tpu	t:							Reason: This test case is unique because we are testing on an empty board where no values exist. So this method tests to make sure the board was properly
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7	initialized
0									0									
1									1									Function Name:
2									2			Х						placeMarkerOnEmptyBoard ()
3									3									
4									4									
5									5									
6									6									
7									7									
Pos	s.g	r = '; etR etC	ow															

Inp	ut:								Ou	tpu	t:							Reason: This test case is unique because we are testing to make sure the board's (0,0) position is
Sta	te:																	the top left and make sure there are no out of bounds problems.
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7	
0									0	Х								
1									1									Function Name: placeMarkerTopLeft()
2			Х						2			X						proceduration in the second control of the s
3									3									
4									4									
5									5									
6									6									
7									7									
pla Pos Pos	s.ge	etRo	wc	= 0 : 0														

Inp	ut:								Ou	tpu	t:							Reason: This test case is unique because we are testing to make
Sta	ate:																	sure the board's (7,7) position is the bottom right and make sure there are no out of bounds
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7	problems.
0									0									
1									1									Function Name:
2			Х						2			Х						placeMarkerTopLeft()
3									3									
4									4									
5									5									
6									6									
7									7								Х	
				•			,	•		l	l	l	l	l	l	l		
Po	yer s.ge s.ge	etRo	ow															

Inp	ut:								Ou	tpu	ıt:							Reason: This test case is unique because we are testing to make
Sta	ite:																	sure the board's (7,0) position is the bottom left and make sure there are no out of bounds
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7	problems. On top of this we are also testing to make sure row and
0									0									column correspond to their proper location on the board (row,column)
1									1									
2			Х						2			Х						
3									3									Function Name: placeMarkerOnRow
4									4									Boundary()
5									5									
6									6									
7									7	Х								
										I	I					I		
Po	yer s.ge s.ge	etRo	ow															

Inp	ut:								Ou	tpu	ıt:							Reason: This test case is unique because we are testing to make
Sta	ıte:																	sure the board's (0,7) position is the top right and make sure there are no out of bounds problems.
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7	On top of this we are also testing to make sure row and column
0									0								Х	correspond to their proper location on the board (row,column)
1									1									,
2			Х						2			Х						
3									3									Function Name: placeMarkerOnColumn
4									4									Boundary()
5									5									
6									6									
7									7									
										I	I		I			I		
Po	s.ge	= '; etRo etCo	wc															

Inp Sta									Output: whatsAtPos = ' '	Reason: This test case is unique because we are testing to make sure whatsAtPos functions on an empty board.
	0	1	2	3	4	5	6	7		
0										Function Name:
1										whatsAtPosOnEmpty Board()
2										Board()
3										
4										
5										
6										
7										
Pos	s.ge	· = '; etRi etCi	wc	= 4 4						

Inp	ut:								Output:	Reason: This test case is unique because we are testing to make
Sta	ite:								whatsAtPos = 'X'	sure whatsAtPos is working with the board (0,0) being at the top left and that it's properly grabbing the
	0	1	2	3	4	5	6	7		character that was placed there
0	Х									
1										Function Name:
2										whatsAtPosOnMarkedTop Left()
3										Terc()
4										
5										
6										
7										
Pos	yer s.ge	etRo	wc							

Inp	ut:								Output:	Reason: This test case is unique because we are testing to make
Sta	ıte:								whatsAtPos = 'X'	sure whatsAtPos is working with the board (7,7) being bottom right and that it's properly grabbing the
	0	1	2	3	4	5	6	7		character that was placed there without going out of bounds. This
0										is the boundary for both row and column
1										Solution
2										
3										Function Name:
4										whatsAtPosOnMarked BottomRight()
5										
6										
7								Х		
	•	•	•	•	•	•	•			
pla Pos Pos	s.ge	etRo	wc							

Inp	ut:								Output:	Reason: This test case is unique because we are testing to make
Sta	ıte:								whatsAtPos = 'X'	sure whatsAtPos is working with the board (7,0) being bottom right and that it's properly grabbing the
	0	1	2	3	4	5	6	7		character that was placed there without going out of bounds. This
0										is the upper boundary for row and lower boundary for column
1										
2										
3										Function Name: whatsAtPosOnMarked
4										BottomLeft()
5										
6										
7	Х									
pla				_						
Pos										

Inp	ut:								Output:	Reason: This test case is unique
Sta	ite:								whatsAtPos = 'X'	because we are testing to make sure whatsAtPos is working with the board (0,7) being bottom right and that it's properly grabbing the
	0	1	2	3	4	5	6	7		character that was placed there without going out of bounds. This
0										is the lower boundary for row and upper boundary for column
1	1									apper soundary for column
2										
3										Function Name: whatsAtPosOnMarkedTopRi
4										ght()
5										
6										
7										
Pos	player = 'X' Pos.getRow = 0 Pos.getCol = 7									

1. checkSpace(BoardPosition pos)

Inp									Output: checkSpace = true	Reason: This test case is unique because we are testing on a board that is empty. Therefore checkSpace must return true every single time.			
	0	1	2	3	4	5	6	7	State of the board is unchanged				
0									anonangea				
1	1									Function Name: checkSpacePlacementEmpt			
2										yBoard()			
3													
4													
5													
6													
7													
Pos Pos													

2. checkSpace(BoardPosition pos)

Inp	ut:								Output: checkSpace = true	Reason: This test case is unique because we are testing on a board that is almost full with one				
Sta	ite:									remaining location. The method can only return true for this				
	0	1	2	3	4	5	6	7	State of the board is unchanged	particular value and if it does it's a good indication it's working				
0								X	unchanged	properly				
1	1 O X O X O X O X							X						
2	2 0 X 0 X 0 X 0 X							X		Function Name:				
3								X		checkSpacePlacementOneA				
4	3 O X O X O X O X							0		wayFromFullBoard()				
5	0	Χ	0	Χ	0	Χ	0	X						
6	0	Χ	0	Χ	0	Χ	0	X						
7	0	Χ	0	Χ	0	Χ	0	X						
Pos Pos	s.ge s.ge													

3. checkSpace(BoardPosition pos)

Inp	ut:								<pre>Output: checkSpace = false</pre>	Reason: This test case is unique because we are testing on a board that represents a game board that				
Sta	ite:									could happen during live play. We are checking a position already taken to make sure it returns false				
	0	1	2	3	4	5	6	7	State of the board is unchanged					
0	х	0		Х		0	Χ	Ο	differentiation					
1		0	Χ	0	Χ	Χ		X		Function Name:				
2	2 x X X O O X									<pre>checkSpacePlacementNorm alBoard()</pre>				
3								Ο		alboara ()				
4	Х	Х		0		Х		X						
5		0		0		Χ	Х	X						
6	х	Х	Х	0	Χ	Χ		X						
7	х	Χ	Χ	Χ		Χ		X						
Pos Pos														

Inp	ut:								Output: checkHorizontalWin =	Reason: This test case is unique because we are testing to make sure that if the last mark we				
Sta	ite:								true	placed connected into a horizontal line that the method can properly detect the new horizontal line that was found through the middle				
	0	1	2	3	4	5	6	7						
0	Х	Х	Х	Х	X				State of the board is unchanged	"position".				
1														
2										Function Name:				
3										checkHorizontalWin PlacedInMiddle()				
4										riacedinmiddie()				
5														
6														
7														
Nu	yer mTo s.ge	oW etRo	in = ow :	= 2										

Input:

State:

	0	1	2	3	4	5	6	7
0	Χ	Χ	Χ		Х	Χ	Χ	
1	Х	Х	Х		Х	Х	X	
2	X	X	X		Χ	X	X	
3	X	X	X		Х	X	Χ	
4	X	X	X		Х	X	Χ	
5	Χ	Χ	Χ		Х	Χ	Χ	
6	Х	X	X		Х	Х	X	
7	Χ	Χ	Χ		Х	Χ	Х	

player = 'X' numToWin = 4 Pos.getRow = 2 Pos.getCol = 2

Output:

checkHorizontalWin =
false

State of the board is unchanged

Reason: This test case is unique because we are testing to make sure that there are no false positives detected. We are filling every row to one away from it's win and then putting a space in between it so it's no longer a consecutive string of characters

Function Name:

checkHorizontalWin
PlacedInMiddle()

Input:

State:

	0	1	2	3	4	5	6	7
0	Χ	Χ	Χ		Х	Χ	Χ	
1	Х	Х	Х		Х	Х	Х	
2	Χ	Χ	Χ		Х	Χ	Χ	
3	Х	Х	Х		Х	Х	Х	
4	Χ	Χ	Χ		Χ	Χ	Χ	Χ
5	Χ	Χ	Χ		Χ	Χ	Χ	
6	Χ	X	X		Х	X	X	
7	Χ	Χ	Χ		Х	Χ	Χ	

player = 'X' numToWin = 4 Pos.getRow = 4 Pos.getCol = 7

Output:

checkHorizontalWin =
true

State of the board is unchanged

Reason: This test case is unique because we are testing to make sure that there are no false positives as well that the board does detect the one valid horizontal win. On top of detecting the win we are checking to make sure the fact that the horizontal reaches the column's boundary does not affect the win detection

Function Name:

checkHorizontalDetectin
gProperWin()

Inp	ut:								<pre>Output: checkHorizontalWin = true</pre>	Reason: This test case is unique because we are testing to make sure that there are no false				
Sta	ite:								Cluc	positives from having alternating characters as well as if another character interrupts a particular character it's horizontal streak is				
	0	1	2	3	4	5	6	7	Otata of the beautiful					
0									State of the board is unchanged	no longer counted.				
1	1 X X X X X X													
2										Function Name:				
3										checkHorizontalWinWithM ixedChars()				
4	0	0	0	Χ	0	0	0	Ο		TXeuchars ()				
5	Х	Х	Х		Χ	Χ	Χ							
6	0	0	0		0	0	0							
7	X	Χ	X		Χ	X	Χ							
nur	yer nTc s.ge s.ge	oWi etRo	n = ow	= 4										

Inp	ut:								Output: checkVerticalWin=	Reason: This test case is unique because we are testing to make					
Sta	ite:								true	sure that if the last mark we placed connected into a vertical line that the method can properly					
	0 1 2 3 4 5 6 7								Otata aftha haardia	detect the new vertical line given the middle position in between the					
0	Χ								State of the board is unchanged	line as the lastPos					
1	Х														
2	X									Function Name:					
3	 									checkVerticalWin					
4	Х									PlacedInMiddle()					
5															
6															
7															
pla Nu Pos Pos	mTo s.ge	oW etRo	in = ow	= 0											

Inp	ut:								Output: checkVerticalWin =	Reason: This test case is unique because we are testing to make				
Sta	ite:								false	sure that there are no false positives detected. We are filling every column to one away from it's win and then putting a space in between it so it's no longer a				
	0	1	2	3	4	5	6	7	Ctata at the beautie					
0									State of the board is unchanged consecutive string of ch	consecutive string of characters				
1	X	Х	Х	Χ	Χ	Χ	Χ	Χ						
2	X	Х	Х	Χ	Χ	Χ	Х	X		Function Name:				
3										checkVerticalWinNoFalse				
4	Х	Х	Х	Χ	Χ	Χ	Х	X		Positive()				
5	Х	Х	Х	Χ	Х	Χ	Х	Х						
6	Х	Х	Х	Х	Х	Х	Х	Х						
7														
nur Po:	yer mTo s.ge	oWi etRo	n = ow	= 2										

lı	1	p	u	t	

State:

	0	1	2	3	4	5	6	7
0	Χ	Χ	Χ	Χ	Χ	Χ	Χ	Χ
1	Х	Χ	Х	Х	Χ	Χ	Х	Х
2	Χ	Χ	Χ	Χ	Χ	Χ	Χ	Χ
3								
4	Х	Χ	Х	Х	Χ	Χ	Х	Х
5	Χ	Χ	Χ	Χ	Χ	Χ	Χ	Χ
6	X	X	X	X	Χ	Χ	X	X
7				X				

player = 'X' numToWin = 4 Pos.getRow = 7 Pos.getCol = 4

Output:

checkVerticalWin =
true

State of the board is unchanged

Reason: This test case is unique because we are testing to make sure that there are no false positives as well that the board does detect the one valid vertical win. On top of detecting the win we are checking to make sure the fact that the vertical reaches the rows boundary does not affect the win detection

Function Name:

checkHorizontal
DetectingProperWin()

Inp	ut:								Output: checkVerticalWin =	Reason: This test case is unique because we are testing to make
Sta	ite:								true	sure that there are no false positives from having alternating characters as well as if another
	0	1	2	3	4	5	6	7	Otata aftha haardia	character interrupts a particular character it's vertical streak is no
0	0	Χ	0	Χ	0	X	0	X	State of the board is unchanged	longer counted.
1	0	Х	0	Χ	0	Х	0	Х		
2	0	Х	0	Χ	0	Χ	0	Х		Function Name:
3					Χ					checkHorizontal
4	0	Χ	0	Χ	0	Χ	0	Х		DetectingProperWin()
5	0	Х	0	Χ	0	Χ	0	Х		
6	0	Χ	0	Χ	0	Χ	0	Х		
7					0					
player = 'O' numToWin = 4 Pos.getRow = 7 Pos.getCol = 4										

Input:	

State:

	0	1	2	3	4	5	6	7
0								Х
1							Χ	
2						Χ		
3					Х			
4				Χ				
5			Χ					
6		X						
7	Х							

player = 'X' NumToWin = 4 Pos.getRow = 4 Pos.getCol = 3

Output:

checkDiagonalWin=
true

State of the board is unchanged

Reason: This test case is unique because we are testing to make sure that the right diagonal is being properly detected when it cuts through the center board

Function Name:

checkDiagonalWin
StandardLeftDiagonal()

Inp Sta									<pre>Output: checkDiagonalWin= true</pre>	Reason: This test case is unique because we are testing to make sure that the left diagonal is being properly detected when it cuts
										through the center board
	0	1	2	3	4	5	6	7		
0	Х								State of the board is unchanged	
1		Х								Function Name: checkDiagonalWin
2			Х							StandardLeftDiagonal()
3				Χ						
4					Χ					
5						Х				
6							Х			
7								X		
pla Nui Pos Pos	mTo s.ge	oWi etRo	in = ow	= 4						

	Input: State:									Output: checkDiagonalWin= true	Reason: This test case is unique because we are testing to make sure that the right diagonal is not affected when board size changes
	0	1	2	3	4	5	6	7	8		
0					X					State of the board is unchanged	Function Name:
1				X							checkDiagonalWinOn
2			Х								RightDB()
3		X									
4	Х										
Nu Po	yer mT s.g	oW etR	/in low	<i>i</i> =	1			L			

	Input: State:									Output: checkDiagonalWin= true	Reason: This test case is unique because we are testing to make sure that the left diagonal is not affected when board size changes
	0	1	2	3	4	5	6	7	8	State of the board is	
0										unchanged	Function Name:
1	Χ										<pre>checkDiagonalWinOn LeftDB()</pre>
2		X									LCT CDD ()
3			Χ								
4				X							
Nu Po:	player = 'X' NumToWin = 4 Pos.getRow = 4 Pos.getCol = 4								1		

							T
Inpu	ıt:					Output: checkDiagonalWin=	Reason: This test case is unique because we are testing to make sure that the right diagonal is not
State	e:					true	affected when board size changes. Specifically when total
	0	1	2	3	4		rows is greater than columns as this affects how the diagonal is
0						State of the board is unchanged	calculated
1							
2							Function Name:
3							checkDiagonalWinOnRight
4					Х		WithRHigherThanC()
5				Х			
6			Х				
7		Х					
8	Х						
Num Pos.	er = '; nToW getR getC	in = 4 ow =	8				

						_	·
Inpu						Output: checkDiagonalWin= true	Reason: This test case is unique because we are testing to make sure that the left diagonal is not affected when board size
Stati	ᠸ.						changes. Specifically when total
	0	1	2	3	4	Chata of the heard in	rows is greater than columns as this affects how the diagonal is
0	Х					State of the board is unchanged	calculated
1		Х					
2			Х				Function Name:
3				Х			checkDiagonalWinOnLeftW ithRHigherThanC()
4					Х		TCHRHIGHEITHANC()
5							
6							
7							
8							
Num Pos.	er = '; nToW getR getC	in = 4 ow =	0				

										D TI. ()
Inp	ut:								Output: checkDiagonalWin =	Reason: This test case is unique because we are testing on a board
Sta	ıte:								false	that represents a game board tha could happen during live play. There are different players and
	0	1	2	3	4	5	6	7		many one away wins but only one diagonal that's valid on the entire
0	х	0		Χ		0	X	0	State of the board is unchanged	board. Gives good indication that the diagonal is being calculated
1		0	Χ	0	Χ	Χ		X		properly
2	х	X	Х	0		0	Χ	0		
3	х	Х	Χ	Χ	Χ	Χ		0		Function Name:
4	Х	Х		0		Χ		X		checkSpacePlacementNorm
5		0		0		Х	Χ	Х		alBoard()
6	х	Х	Х	0	Х	Х		Х		
7	х	Х	Х	Х		Χ		Х		
Pos	player = 'X' Pos.getRow = 6 Pos.getCol = 7									

Inp									Output: checkForDraw = false	Reason: This test case is relatively simple it's testing if a full board is being properly detected as a tied game
	0	1	2	3	4	5	6	7	State of the board is unchanged	
0	Х	0	0	0	X	0	0	0	anonanged	Function Name:
1	0	X	X	X	0	Х	Χ	X		checkForDrawNormal()
2	Х	0	0	0	Χ	0	0	0		
3	0	Χ	X	X	0	Χ	Χ	Х		
4	Х	0	0	0	Χ	0	0	0		
5	0	Χ	Χ	Χ	0	Χ	Χ	Х		
6	х	0	0	0	Χ	0	0	0		
7	0	Х	X	X	0	Х	Χ	Х		

Inp									Output: checkForDraw = false	Reason: This test case is relatively simple; it's testing if a board one away from being tied is detected as a tied board. If this method works it's a very good
	0	1	2	3	4	5	6	7	State of the board is unchanged	indication checkForDraw() behaves properly. Missing char is
0	Х	0	0	0	Χ	0	0	0	unonangoa	7,7 (row,col)
1	0	Х	Χ	Χ	0	Χ	Χ	Х		
2	Х	0	0	0	Χ	0	0	0		Function Name: checkForDrawOneChar
3	0	Х	Х	Χ	0	Χ	Χ	Х		Missing()
4	Х	0	0	0	Χ	0	0	0		
5	0	Χ	Χ	Χ	0	Χ	Χ	Х		
6	х	0	0	0	Χ	0	0	0		
7	0	Χ	Χ	Χ	0	Χ	Χ			

-	Input: State:								Output: checkForDraw = false	Reason: This test case is making sure the draw is covering all corners of the board. Previous tests covered the bottom right and this one covers top left in order to
	0	1	2	3	4	5	6	7	State of the board is unchanged	ensure all corners are being counted
0		0	0	0	Χ	0	0	0	unonangeu	
1	0	Х	Х	Χ	0	Χ	Χ	X		Function Name:
2	Х	0	0	0	Х	0	0	0		checkForDrawOneChar
3	0	Χ	Χ	Χ	0	Χ	Χ	X		MissingOnTop()
4	Х	0	0	0	Χ	0	0	0		
5	0	Х	Х	Χ	0	Χ	Χ	X		
6	х	0	0	0	Χ	0	0	0		
7	0	Χ	Χ	Χ	0	Χ	Χ	Х		

Inp									Output: checkForDraw = false	Reason: This test case is testing the minimum possible marks on a board at the time of checkDraw being called.
	0	1	2	3	4	5	6	7	State of the board is	
0	Х								unchanged	Function Name: checkForDrawOnNearlyEmp
1										tyBoard()
2										
3										
4										
5										
6										
7										
		•					•			

Sta									<pre>Output: isPlayerAtPos = false</pre>	Reason: This test case is unique because it's testing if a player is present on the board while the board is empty. This should always return false in this instance				
0	0	1	2	3	4	5	6	7	State of the board is unchanged					
1										<pre>Function Name: isPlayerAtPosEmptyBoard ()</pre>				
2														
3														
4														
5														
6														
7	7													
pos	yer s.ge s.ge	ŧΝι	ım(

Input: State:									<pre>Output: isPlayerAtPos = false</pre>	Reason: This test case is unique because it's testing a board with only one available position. We are checking to see if this position is available. This gives us a very
	0	1	2	3	4	5	6	7		good indication if the method is behaving properly.
0	Х	0	0	0	Χ	0	0	0	State of the board is unchanged	a constant graphs of
1	0	Χ	Χ	X	0	X	X	Х		Function Name:
2	Х	0	0	0	Χ	0	0	0		isPlayerAtPosFull
3	0	Χ	Χ	Χ	0	Χ	Χ	Х		Board()
4	Х	0	0	0	Χ	0	0	0		
5	0	X	X	X	0	X	X			
6	х	0	0	0	Χ	0	0	0		
7	0	Х	Х	X	0	X	X			

Input:									Output: isPlayerAtPos = true	Reason: This test case is unique because it's testing if a player is	
Sta	ite:									present on the board while the board contains one player. We are checking the only spot where that player belongs. On top of that we also checking to make sure (0,0)	
	0	1	2	3	4	5	6	7	State of the board is unchanged		
0	Х								a	lines up with the top left	
1											
2										Function Name: isPlayerAtPosOneCharOnB	
3										()	
4											
5											
6											
7											
player = 'X' pos.getNumCol = 0 pos.getNumRow = 0											

										<u> </u>
Inp	ut:								<pre>Output: isPlayerAtPos = true</pre>	Reason: This test case is unique because it's testing if a player is present on the board while the
Sta	ite:									board contains one player. We are checking the only spot where that
	0	1	2	3	4	5	6	7	State of the board is unchanged	player belongs. On top of that we also checking to make sure (7,0)
0									anonangea	lines up with the bottom left
1										
2										Function Name: isPlayerAtPosBottom
3										Left()
4										
5										
6										
7	Х									
player = 'X' pos.getNumCol = 0 pos.getNumRow = 0										

Immusti												
Input:									<pre>Output: isPlayerAtPos = true</pre>	Reason: This test case is unique because it's testing if a player is		
									isriayeratros – true	present on the board while the		
Sta	State:									board contains one player. We are		
										checking the only spot where that player belongs. On top of that we		
	0	1	2	3	4	5	6	7	State of the board is unchanged	also checking to make sure (0,7)		
0								X	unchanged	lines up with the top right		
1												
2										Function Name:		
3										isPlayerAtPosTop Right()		
4												
5												
6												
7	7 X											
pos	player = 'X' pos.getNumCol = 0 pos.getNumRow = 0											