Ashlyn Cooper CPSC 2150 Project 5 Project Report

#### **Requirements Analysis**

#### **Functional Requirements**

- 1. As a player, I can click a position so that I can play my token.
- 2. As a player, I can choose to play again so that I can replay tic tac toe.
- 3. As a player, I can have my input validated by the program so that I can place my token in a valid position.
- 4. As a player, I can view output prompting for an input so that I can see when I should make an input.
- 5. As a player, I can view an outputted message at the end of the game, so that I know how the game ended.
- 6. As a player, I can win horizontally, as that I can win the game.
- 7. As a player, I can win vertically, so that I can win the game.
- 8. As a player, I can win diagonally so that I can win the game.
- 9. As a player, I can make a move after my opponent (if they have not one), so that I can take my turn.
- 10. The program can end the game without either player winning, so that the game can end in a tie.
- 11. As a user, I can input the number of columns and number of rows, so that I can set the size of the gameboard.
- 12. As a user, I can input the number in a row needed to win, so that the game is played with that rule.
- 13. As a user, I can input the number of players, so that the game can be played with that number of players.

#### Non-functional Requirements

- 1. The system must be written in Java.
- 2. The program can display the game board to the screen so that the player can see the
- 3. The program can alternate between players so that the program can be played by at least 2 players and up to 10.
- 4. Player 1 will always go first.
- 5. The program can be repeated so that the program has the ability to let the players play again.
- 6. The game board is of size NUM ROWS x NUM COLUMNS as indicated by user input.
- 7. Coordinate (0,0) represents the top left corner of the game board.

#### Design

#### Class Diagrams

- board:

- NUM\_ROWS: int [1]

- NUM\_COLUMNS : int [1]

+ GameBoardMem(void) : void

+ getNumRows(void) : int

+ getNumToWin(void) : int

+ getNumColumns(void) : int

- NUM\_TO\_WIN : int [1]

### <<Interface>> **IGameBoard** START\_X: int [1] - START\_Y : int [1] - MINROWS : int [1] - MINCOLS : int [1] - MAXROWS : int [1] - MAXCOLS : int [1] - MINWIN : int [1] - MAXWIN : int [1] + checkSpace(BoardPosition) : bool + placeMarker(BoardPosition, char) : void + checkForWinner(BoardPosition) : bool + checkForDraw(void) : bool + checkHorizontalWin(BoardPosition, char) : bool + checkVerticalWin(BoardPosition, char) : bool + checkDiagonalWin(BoardPosition, char) : bool + whatsAtPos(BoardPosition) : char + isPlayerAtPos(BoardPosition, char) : bool + toString(void) : String + getNumRows(void) : int + getNumColumns(void) : int + getNumToWin(void) : int GameBoardMem GameBoard - board: char [8][8] - NUM\_ROWS : int [1] - NUM\_COLUMNS : int [1] - NUM\_TO\_WIN : int [1] + GameBoard(void): void + placeMarker(BoardPosition, char): void + placeMarker(BoardPosition, char): void + whatsAtPos(BoardPosition) : char + whatsAtPos(BoardPosition) : char + isPlayerAtPos(BoardPosition, char): boolean + getNumRows(void) : int + getNumColumns(void) : int + getNumToWin(void) : int Extends Extends

AbsGameBoard

+ toString(void) : String

### BoardPosition

- Row : int [1]

- Column : int [1]

+ BoardPosition(int, int): void

+ getRow(void) : int

+ getColumn(void) : int

+ equals(void) : bool

+ toString(void) : String

#### TicTacToeController

curGame : IGameBoard [1]

- screen : TicTacTowView [1]

+ MAX\_PLAYERS : int [1]

- players : Character[10]

numTurns : int [1]numPlayers : int [1]

- playingGame : boolean [1]

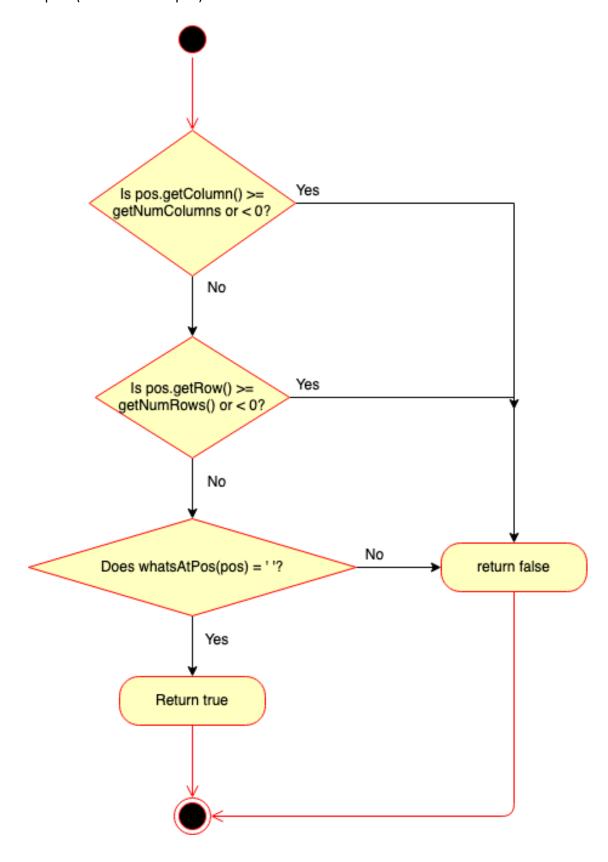
- curPlayer : char [1]

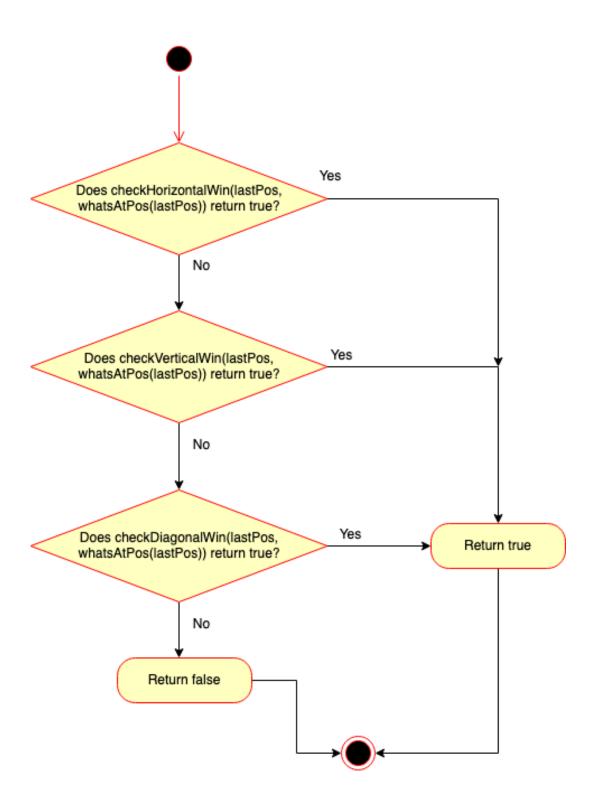
+ TicTacTowController(IGameBoard, TicTacToeView, int): void

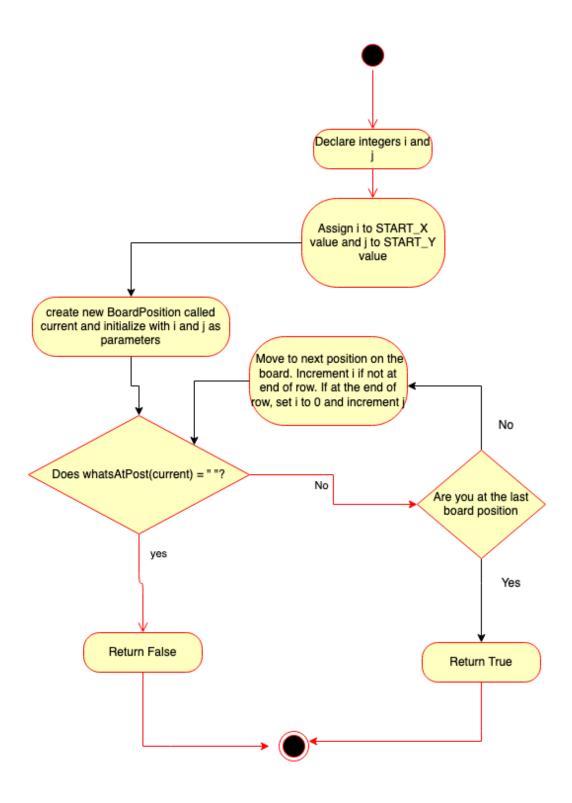
+ processButtonClick(int, int) : void

newGame(void) : void

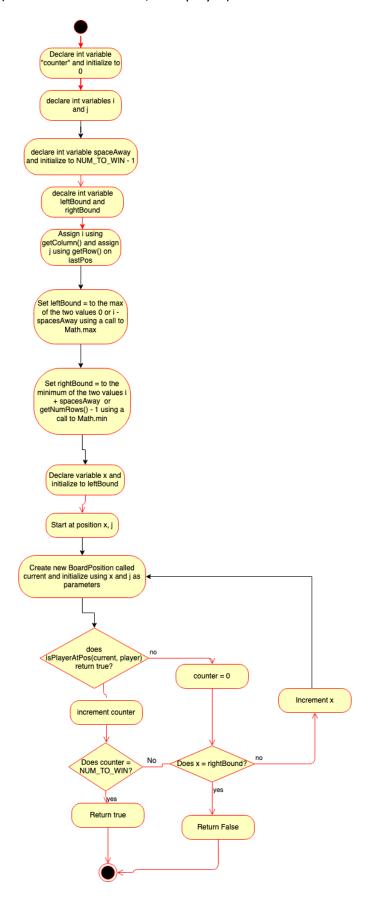
Activity Diagrams
IGameBoard (default methods)
checkSpace(BoardPosition pos)



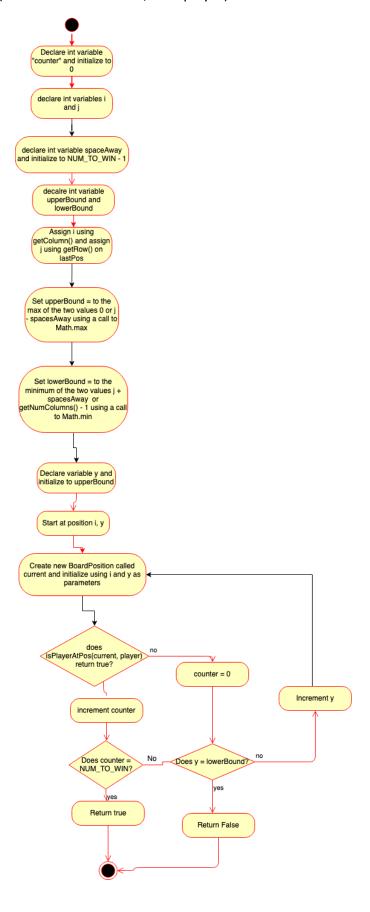




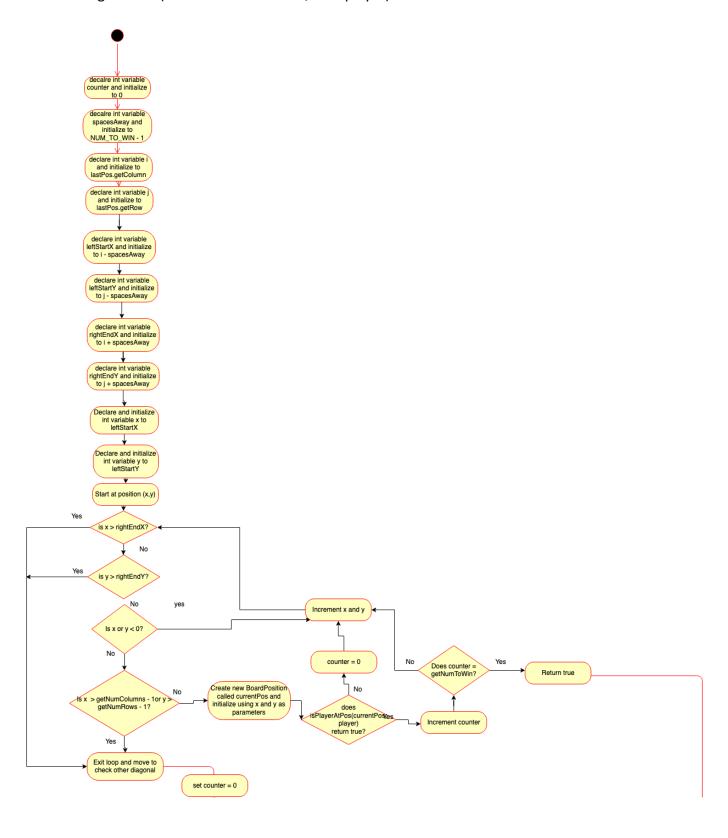
### checkHorizontalWin(BoardPosition lastPos, char player)

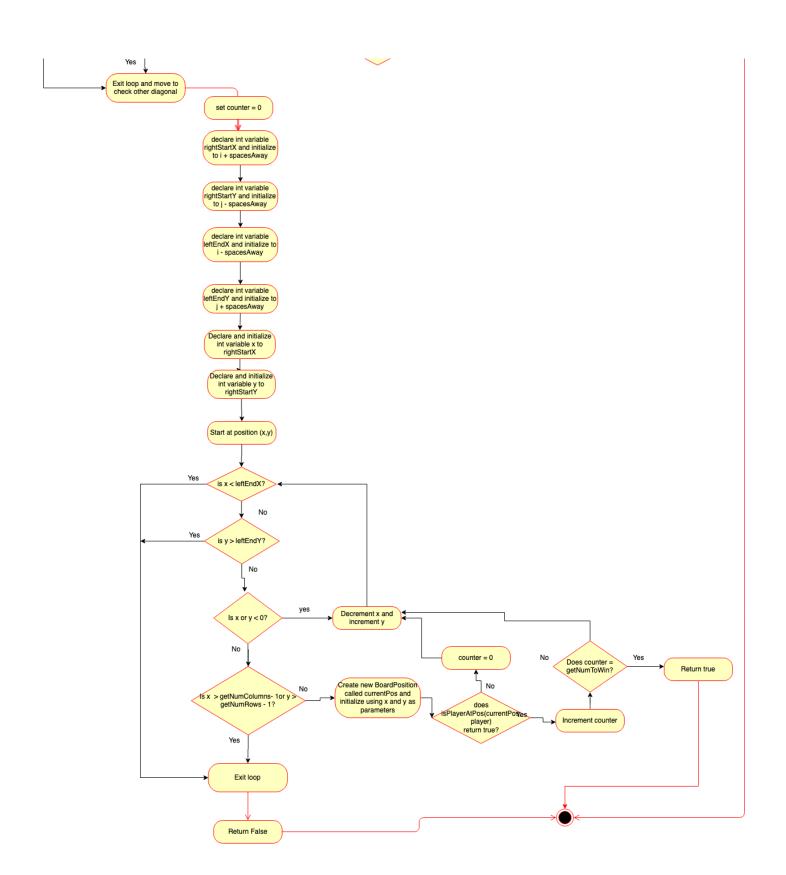


### checkVerticallWin(BoardPosition lastPos, char player)

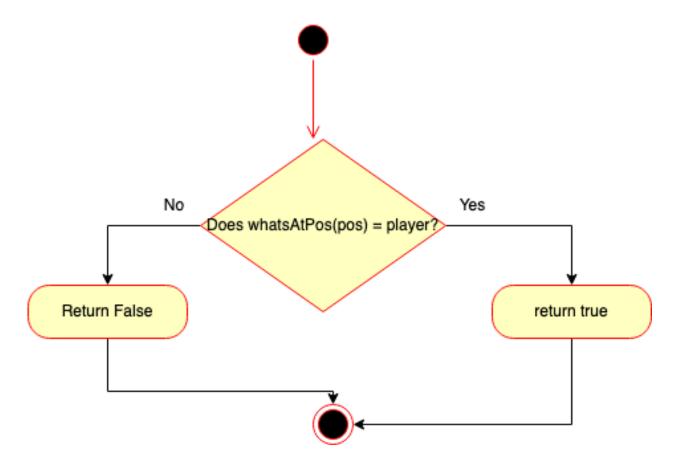


### checkDiagonalWin(BoardPosition lastPos, char player)



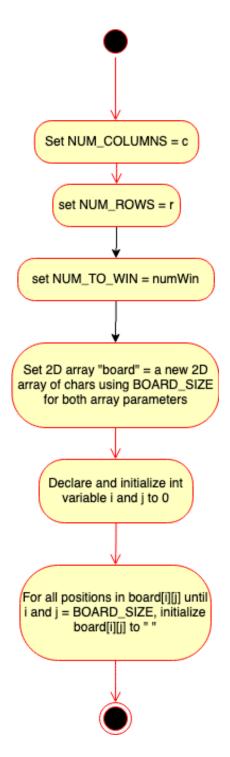


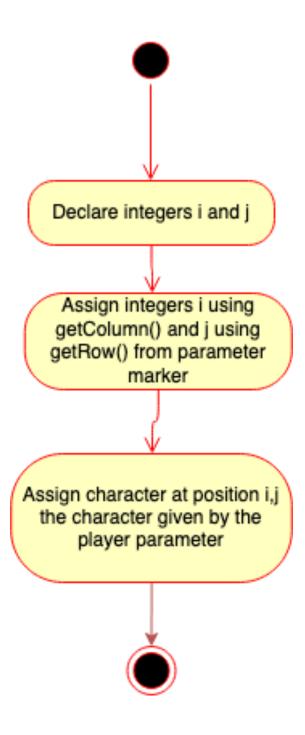
# isPlayerAtPos(BoardPosition pos, char player)

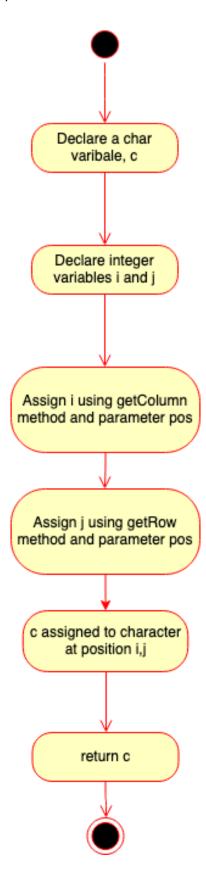


### GameBoard – Primary Implementations

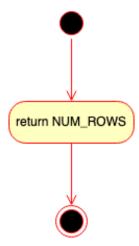
GameBoard(int r, int c, int numWin) (constructor)



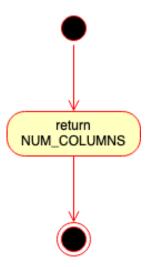




# getNumRows()



## getNumColumns()

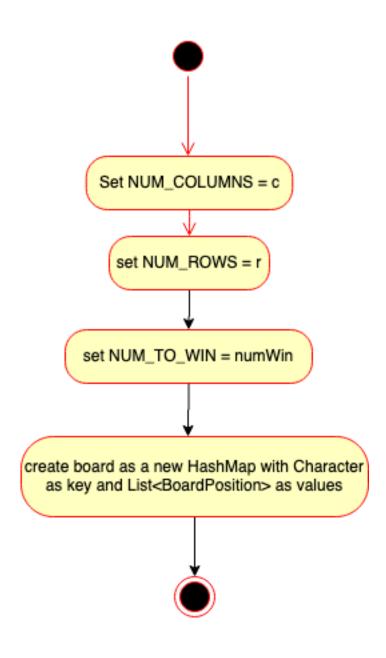


### getNumToWin()

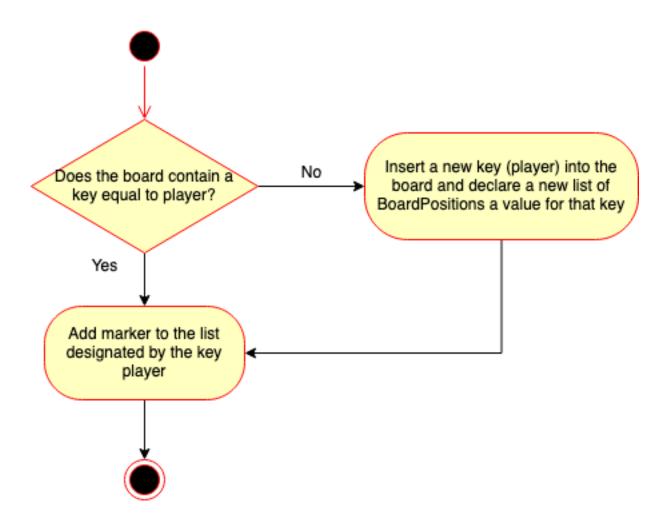


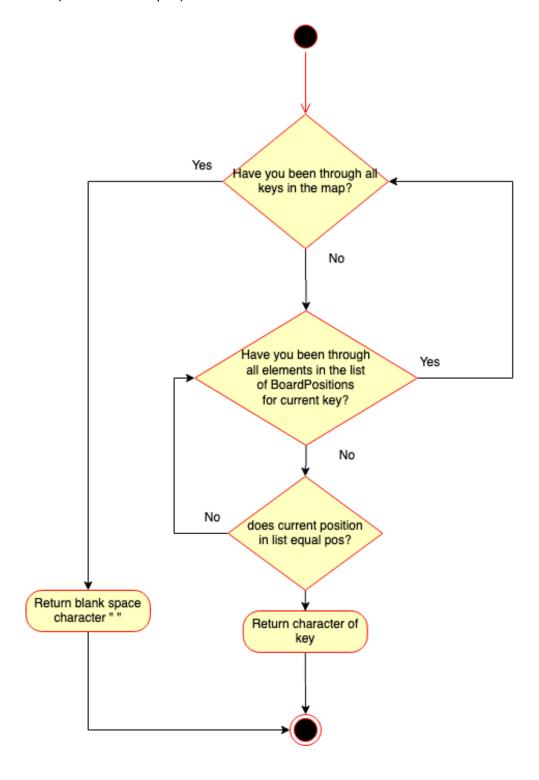
GameBoardMem – Primary Implementations

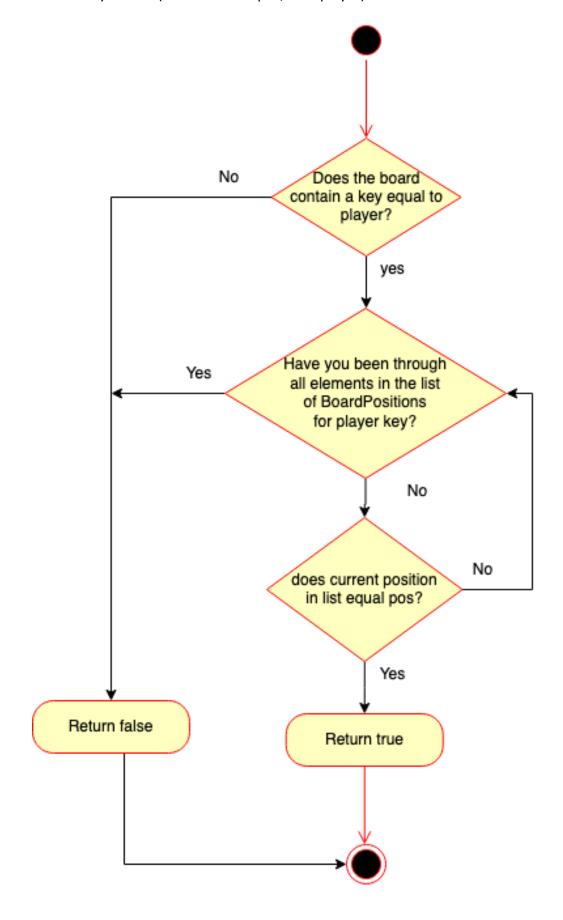
GameBoardMem(int r, int c, int numWin) (constructor)



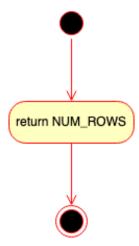
### placeMarker(BoardPosition marker, char player)



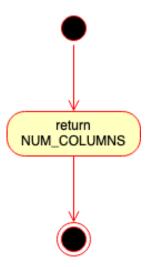




# getNumRows()



## getNumColumns()

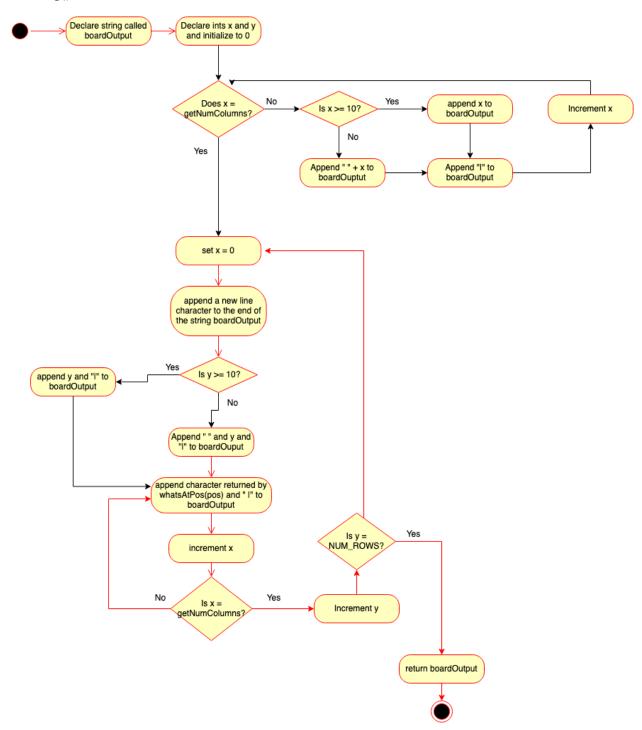


### getNumToWin()



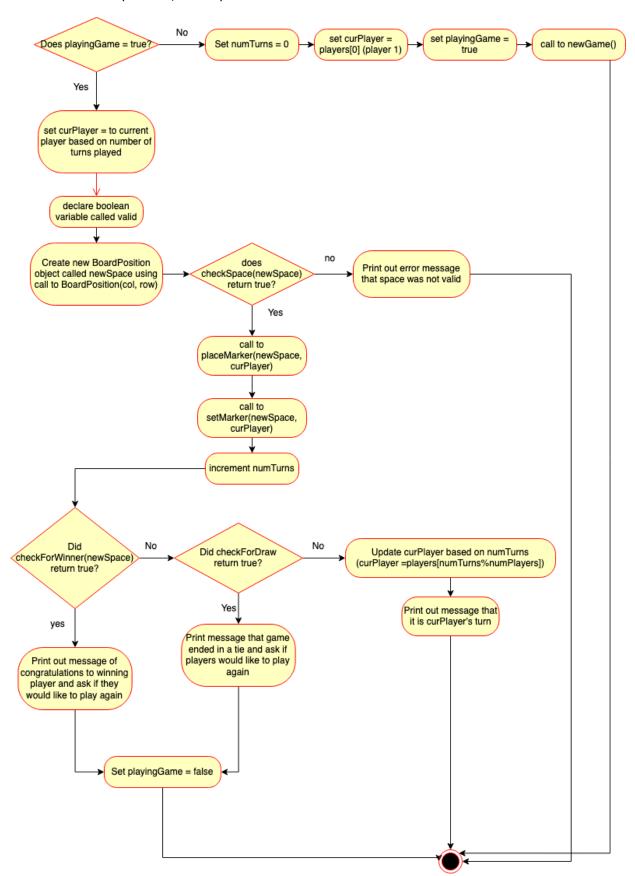
### AbsGameBoard

### toString()



#### TicTacToeController

### processButtonClick(int row, int col)



## <u>Testing</u>

Below are the test cases for each method.

### Constructor

Input	Out	out						Reason
•	-		UM	ТО	WIN	l= 3	3	This test case is distinct because
r = 3			_	RO\	-			it tests the constructor's
c = 3				_ COL			= 3	boundary case. 3 is the
numWin = 3				_				minimum number of rows,
			0	1		1	2	columns, and spaces in a row to
	0							win needed.
	1							
	2							Function Name:
		ı						test_Constructor_Min_Size
	State	e: N	UM <sub>.</sub>	_TO_	WIN	l= 2	25	This test case is distinct because
r = 100				_RO\				it tests the other boundary case
c = 100		N	UM	_COL	.UM	NS	=	for the constructor. The max
numWin = 25	100							number of rows and columns
	١							the board can have is 100 and
				empt	•			the max needed in a row to win
	with	SIDE	es tr	nat go	o fro	m	0-99.	can be 25.
		0		1			99	Function Name:
	0							test_Constructor_Max_Size
	1							
	99							
	State		_	_TO_	-		1	This test case is distinct because
r = 6				_RO\			_	it tests a routine scenario for
c = 6		N	UM	_COL	.UM	NS	= 6	the constructor. 6 is between
numWin = 4			_		_		1_1	the minimum and maximum for
		0	1	2	3	4	5	the number of rows and
	0			1				columns. 4 is in between the
	1			+-				minimum and maximum for the
	2			+-			-	number needed to win.  Function Name:
	3							test_Constructor_Routine
	4							
	5		1					

# boolean checkSpace(BoardPosition pos)

Inpu	t				Output	Reason		
State	e: (nu	m to	win	= 3)	checkSpace = true	This test case is distinct because it checks if		
	0	1	2	3		a space is available at a boundary position		
0					State of board is	on the gameboard (0,0). The test case is		
1					unchanged	also distinct as it is testing a space without		
2						a character.		
3								
	getRo getCo					Function Name: test_checkSpace_at_upperLeftCorner		
State: (num to win = 3)					checkSpace = false	This test case is distinct because it is testing a scenario in which the player is checking a		
	U			3	State of the board is	space that is off of the board (out of		
0					unchanged	bounds), so checkSpace is expected to		
2					unchanged	return false, as it is an invalid position.		
						return raise, as it is an invalid position.		
3						Function Name:		
Pos.getRow = 4 Pos.getColumn = 4						test_checkSpace_at_outOfBounds		
State	e: (nu	m to	win	= 3)	checkSpace = false	This test case is distinct because it is testing		
	0	1	2	3		a scenario in which the player is checking a		
0					State of the board is	space that is already taken by another		
1		х			unchanged	player token, so the space is unavailable.		
2					_			
3						Function Name: test checkSpace Unavailable		
	getRo getCo							

boolean checkHorizontalWin(BoardPosition lastPos, char player)

state of the board is unchanged  of 4 consecutive x's as opposed to on the end, so the function need to count x's on the right and left.  player = 'x' lastPos.getRow = 2 lastPos.getColumn = 2  State(number to win = 4)  0 1 2 3 4 0 checkHorizontalWin = true 1	Inpu	t					Output	Reason
State of the board is unchanged   State of the board and make sure it   State of the board and mak	State	e(nur	mber	to v	vin =	4)		This test case is unique and
state of the board is unchanged    1		0	1	2	3	4	checkHorizontalWin = true	distinct because the last x was
2	0							placed in the middle of the string
This test case is unique and distinct because the last x was placed on the left boundary of the board.	1						state of the board is	of 4 consecutive x's as opposed to
State (number to win = 4)	2	х	х	Х	х		unchanged	on the end, so the function needs
player = 'x' lastPos.getRow = 2 lastPos.getColumn = 2  State(number to win = 4)	3	О	О	О	х	0		to count x's on the right and left.
lastPos.getRow = 2 lastPos.getColumn = 2  State(number to win = 4)	4							
lastPos.getColumn = 2   State(number to win = 4)	play							
State(number to win = 4)				w = :	2			test_Horizontal_MiddleWin
State(number to win = 4)    O		_						
checkHorizontalWin = true distinct because the last x was placed on the left boundary of the board, so the function needs to start counting at the 0 column of the board and make sure it does not start counting from outside of the board.  State(number to win = 4)  State(number to win = 4)  O						4)		This test case is unique and
state of the board is unchanged    1						<del></del> -	checkHorizontalWin = true	· ·
state of the board is unchanged    1	0							placed on the left boundary of the
2xxxx3000x04111player = 'x'the board and make sure it does not start counting from outside of the board.IastPos.getRow = 2 lastPos.getColumn = 0Function Name: test_Horizontal_left_BoundersState(number to win = 4)This test case is unique and distinct because the last x was placed on the right boundary of the board, so the function needs to stop counting at the 5th column of the board and make sure it	-						state of the board is	1 .
the board and make sure it does not start counting from outside of the board.    State(number to win = 4)	l <del></del>	х	х	х	х		unchanged	start counting at the 0 column of
A	3	0	О	0	х	0		the board and make sure it does
lastPos.getColumn = 0  State(number to win = 4)  0	-							not start counting from outside of
lastPos.getRow = 2 lastPos.getColumn = 0  State(number to win = 4)  0 1 2 3 4 0 checkHorizontalWin = true 1 2 x x x x x x 3 0 0 0 x 0  Function Name:  test_Horizontal_left_Bound distinct because the last x was placed on the right boundary of the board, so the function needs to stop counting at the 5th column of the board and make sure it	plave	er = '	x'	I				the board.
State(number to win = 4)   State(number to win = 4)   CheckHorizontalWin = true   This test case is unique and				w = :	2			
State(number to win = 4)  O 1 2 3 4  O CheckHorizontalWin = true  CheckHorizontalWin = true  O CheckHorizontal left Bound  O CheckHorizontal left Bound  O CheckHorizontal left Bound  O CheckHorizontal left Bound  O CheckHorizontalWin = true  O Chec	_							
012340123412234223443000002344434544544566667667898999999999999999999910 <t< td=""><td colspan="6"></td><td></td><td>test_Horizontal_left_Bound</td></t<>								test_Horizontal_left_Bound
0       Image: state of the board is growing at the placed on the right boundary of the board, so the function needs to stop counting at the 5th column of the board and make sure it growing at the state of the board and m	State							· ·
1state of the board is 2the board, so the function needs to stop counting at the 5th column of the board and make sure it		0	1	2	3	4	checkHorizontalWin = true	
2 x x x x unchanged to stop counting at the 5th column of the board and make sure it	-							
3 o o o x o of the board and make sure it	I							
	I		Х	Х	Х	Х	unchanged	, ,
4	I	0	0	0	Х	0		
le a cont	L							
player = 'x' board.								board.
lastPos.getRow = 2		_						Function Name
lastPos.getColumn = 4  Function Name: test_Horizontal_right_Bou	lastP	os.g	etCo	lumr	า = 4			test_Horizontal_right_Bound
State(number to win = 4)  This test case is unique and	State	e(nur	mber	to v	vin =	4)		This test case is unique and
0 1 2 3 4 checkHorizontalWin = distinct because the last x was							checkHorizontalWin =	•
0 false placed in a position that does no	0						false	placed in a position that does not
	1	Х	Х		Х	х		result in there being enough x's in
2 0 0 0 state of the board is a row to satisfy the condition of	2		О	0	0		state of the board is	a row to satisfy the condition of 4
3 o unchanged in a row needed to win, so we	3		0				unchanged	in a row needed to win, so we
expect the function to return	l <del></del>							expect the function to return
false.		1	1	1	1			false.

player = 'x'	
lastPos.getRow = 1	Function Name:
lastPos.getColumn = 0	test_Horizontal_No_Win

boolean checkVerticalWin(BoardPosition lastPos, char player)

Inpu	t					Output	Reason
State	e(nur	nber	to wi	n = 4	)		This test case is unique and
	0	1	2	3	4	checkVerticalWin = true	distinct because the last x
0			х				was placed in the middle of
1	0		Х			state of the board is	the string of 4 consecutive x's
2	Х		Х			unchanged	as opposed to on the end, so
3	0		х	0			the function needs to count
4	0						x's both above and below the
playe	er = ':	x'					final x placed.
lastP	os.ge	etRov	v = 1				
lastP	os.ge	etColu	umn :	= 2			Function Name: test_Vertical_Middle
State	e(nur	nber	to wi	n = 4	)		This test case is unique and
	0	1	2	3	4	checkVerticalWin = true	distinct because the last x
0			х				was placed on upper
1	0		х			state of the board is	boundary of the board, so
2	Х		х			unchanged	the function needs to make
3	0		х	0			sure it does not go into a
4	0						negative index (past row 0)
playe	er = ':	x'					when checking for a win.
lastPos.getRow = 0							
lastP	os.ge	etColu	umn :	= 2			Function Name: test_Vertical_Upper
State	e(nur	nber	to wi	n = 4	)		This test case is unique and
	0	1	2	3	4	checkVerticalWin = true	distinct because the last x
0							was placed on the lower
1	0		х			state of the board is	boundary of the board, so
2	х		х			unchanged	the function needs to make
3	0		х	0			sure it does not go to an
4	0		х				index out of bounds (past the
playe	er = ':	x'					last row) on the gameboard.
1	_	etRov					
lastPos.getColumn = 2							Function Name: test_Vertical_Lower
State	e(nur	nber	to wi	n = 4	)		This test case is unique and
	0	1	2	3	4	checkVerticalWin = false	distinct because the last x

0 1 2	0 X		X		state of the board is unchanged	was placed in a position that does not result in there being enough x's in a row to satisfy the condition of 4 in a row
4	0		X	0		needed to win, so we expect the function to return false.
lastF	er = 'z Pos.ge Pos.ge	etRov		= 2		Function Name: test_Vertical_No_Win

boolean checkDiagonalWin(BoardPosition lastPos, char player)

Inpu	t					Output	Reason
State	e(nur	nber	to w	in = 3	3)		This test case is unique and
	0	1	2	3	4	checkDiagonalWin = true	distinct because the last x was
0	Х						placed at the corner (0,0) which
1	0	Х				state of the board is	is a boundary case where it is at
2	0		Х			unchanged	the minimum possible location.
3							It is also different because the
4							diagonal is at a left to right
playe	er = ':	x'					angle.
lastP	os.ge	etRov	w = 0	)			Function Name:
lastP	os.ge	etCol	umn	= 0			test_Diagonal_Upper_Left
State(number to win = 3)					3)		This test case is unique and
	0	1	2	3	4	checkDiagonalWin = true	distinct because the last x was
0					Х		placed at the upper right-hand
1	0			х		state of the board is	corner which is another
2	0		х			unchanged	boundary case. It is also
3							different because the diagonal
4							is at a right to left angle.
playe	er = ':	x'					
lastP	os.ge	etRov	<i>N</i> = 0	)			Function Name:
lastP	os.ge	etCol	umn	= 4			test_Diagonal_Upper_Right
State	e(nur	nber	to w	in = 3	3)		This test case is unique and
	0	1	2	3	4	checkDiagonalWin = true	distinct because the last x was
0							placed at another boundary
1	0					state of the board is	case (the lower righ- hand
2	0		х			unchanged	corner). Also, the diagonal is
3				х			oriented left to right going into
4					х		the corner

nla	or = 1	.,				1	
playe							F
lastP	_						Function Name:
lastP	os.ge	etCol	umn	= 4			test_Diagonal_Lower_Right
C+a+a	\/nu=	nhar	to **	/in = 3	2)		This tost case is unique and
State	o (nur	nber 1	2	$\frac{dn = 3}{3}$	<del>i </del>	checkDiagonalWin = true	This test case is unique and distinct because the last x was
<del>  -</del>	U	1		3	4	checkblagonarviii = true	
0				-		state of the board is	placed in another boundary
1	0	-		1			case (lower left-hand corner).
2	0		Х	1		unchanged	Also the diagonal is oriented
3		Х					going right to left into the
4	х						corner.
playe	er = ':	x'					
lastP	os.ge	etRov	<b>N</b> = 4	ļ			Function Name:
lastP	os.ge	etCol	umn	= 0			test_Diagonal_Lower_Left
State		nber	1	/in = 3	<del></del>		This test case is unique and
	0	1	2	3	4	checkDiagonalWin = true	distinct because the last x was
0							placed in the middle of a win
1	0	Х				state of the board is	and represents a routine win
2	О		Х			unchanged	where the win is bounded
3				Х			without hitting an edge of the
4				1			board.
	player = 'x'						
lastP			N = 2	2			Function Name:
lastP	_						test_Diagonal_Routine
	0			_			
State	e(nur	nber	to w	/in = 3	3)		This test case is unique and
	0	1	2	3	4	checkDiagonalWin = true	distinct because the last x was
0			Х				placed at an edge of the board.
1	О			Х		state of the board is	Also, each of the ends of the
2	О			1	х	unchanged	diagonal is at an edge or
3				1			boundary of the board, so the
4				1			function cannot go out of
playe	-r = '	 х'	1	1			bounds at both ends.
			N = 7	,			
lastPos.getRow = 2 lastPos.getColumn = 4							Function Name:
iastrus.getCulullill = 4							test_Diagonal_Edges
State	State(number to win = 3)						This test case is unique and
	0	1	2	3	4	checkDiagonalWin = false	distinct because the last x was
0		Ť	-	+			placed in a position that did not
1	0	Х		Х		state of the board is	result in a win in the diagonal.
2	0	_	v	<del>  ^  </del>		unchanged	. coare in a rem in the diagonali
3	0		Х	+			Function Name:
				<u> </u>			

4	test_Diagonal_No_Win
player = 'x'	
lastPos.getRow = 1	
lastPos.getColumn = 3	

### boolean checkForDraw()

Input				Output	Reason
State: (	number	to win	= 3)		This is a distinct test case
	0	1	2	checkForDraw = true	because it represents a
0	х	0	х		scenario that ended in a tie.
1	х	х	0	state of the board does not	All spaces are filled in no
2	0	х	0	change	wins are present.
					Function Name: test_Draw_True
State: (	number	to win	= 3)		This is a distinct test case
	0	1	2	checkForDraw = false	because it represents the
0	х				board after the first move
1				state of the board does not	has been played. Only one
2				change	token is on the board, so a tie
					is not possible. Also, the
					token was placed at the
					boundary (0,0).
					Function Name: test_Draw_FirstSpace
State: (	number	to win	= 3)		This is a distinct test case
	0	1	2	checkForDraw = false	because it represents a case
0	х	0	х		where the last token played
1	х	х	0	state of the board does not	has not quite filled up the
2	0	х		change	board (only one position is
					left).
					Function Name:
					test_Draw_Almost_Full
State: (	number	to win	= 3)		This is a distinct test case
	0	1	2	checkForDraw = false	because it represents a
0	Х		х		routine scenario in which

2	X	0	0	state of the board does not change	some of the board has been filled but not all of it.
					Function Name: test_Draw_Routine

## char whatsAtPos(BoardPosition pos)

Input				Output	Reason
State: (	numbe	er to wi	n = 3)		This is a distinct test case
	0	1	2	whatsAtPos = 'x'	because it looks at the position
0	Х				at the board that is located at the
1				state of the board does not	most minimum boundary, the
2				change	upper left-hand corner (0,0).
pos.get					Function Name: test_whatsAtPos_upperLeft
State: (	numbe	er to wi	n = 3)		This is a distinct test case
	0	1	2	whatsAtPos = 'x'	because it looks at the position
0					at the board that is located at the
1				state of the board does not	most maximum boundary, the
2			х	change	lower right-hand corner (2,2)
pos.get pos.get					test_whatsAtPos_lowerRight
State: (	numbe	er to wi	n = 3)		This is a distinct test case
	0	1	2	whatsAtPos = 'o'	because it looks at a board
0					position that represents a
1		0		state of the board does not	routine scenario. The board
2				change	position is in the middle of the
pos.get					Function Name: test_whatsAtPos_Middle
State: (	numbe	er to wi	n = 3)		This is a distinct test case
	0	1	2	whatsAtPos = ''	because it looks at a position on
					the board that is not currently

0 1 2		0		state of the board does not change	occupied by a player token, so we should expect the function to return a blank space character.
	tRow = getColu	: 2 ımn = 2			Function Name: test_whatsAtPos_EmptySpace
State:	(numbe	er to wi	n = 3)		This is a distinct test case
	0	1	2	whatsAtPos = 'x'	because it looks at a position on
0					the board that is at a boundary
1			х	state of the board does not	for the columns, but not at a
2				change	boundary for the rows. It is at the
	tRow = tColum				right most column but is not near the edge of the board with regards to row location.
					Function Name: test_whatsAtPos_Edge

boolean isPlayerAtPos(BoardPosition pos, char player)

Input				Output	Reason
State: (number to win					This is a distinct test case because it looks
= 3)				isPlayeratPos = true	at the position at the board that is located
	0	1	2		at the most minimum boundary, the upper
0	Х			state of the board	left-hand corner (0,0).
1				does not change	
2					Function Name:
					test_isPlayerAt_at_upperLeftCorner
player = 'x'					
pos.ge	etRow	<i>i</i> = 0			
pos.ge	etColu	ımn =	0		
State: (number to win					This is a distinct test case because it looks
= 3)				isPlayeratPos = true	at the position at the board that is located
	0	1	2		at the most maximum boundary, the lower
0				state of the board	right-hand corner (2,2)
1				does not change	
2			Х		Function Name:
					test_isPlayerAt_at_LowerRight
playe	r = 'x'				

pos.getRow = 2 pos.getColumn = 2		
State: (number to win = 3)	isPlayeratPos = true	This is a distinct test case because it looks at a board position that represents a
0 1 2	state of the board	routine scenario. The board position is in
0 0 x	does not change	the middle of the board and not at any boundary and another player is on the
2	aces not enange	board too.
player = 'x' pos.getRow = 1		Function Name: test_isPlayerAt_at_Middle
pos.getColumn = 1		
State: (number to win = 3)    0	isPlayeratPos = false state of the board does not change	This is a distinct test case because it looks at a position on the board that is not currently occupied by the player token, so the player is not at that position, so we expect the function to return false.  Function Name:
player = 'x'		test_isPlayerAt_No_Players
pos.getRow = 1		
pos.getColumn = 1		
State: (number to win = 3)  0 1 2	isPlayeratPos = false	This is a distinct test case because it looks at a case where there is a player in the space, but it is not the player being
0 x	state of the board does not change	searched for, as compared to just an empty
2 0	does not change	space.
player = 'x' pos.getRow = 1 pos.getColumn = 1		Function Name: test_isPlayerAt_Wrong_Player

# <u>Deployment</u>

Run GUI Configuration of program through IntelliJ.