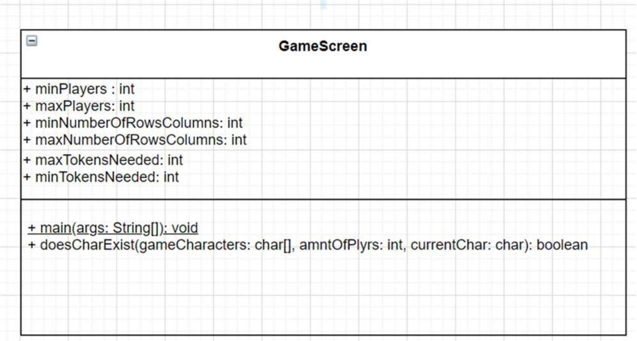
Requirements Analysis:

- **Functional Requirements:**
  1. As a User I need to be able to select where on the board I want to place my X (OR O)
  2. As a User I need to be able to pick another spot on the board if the space is unavailable
  3. As a User I need to be able to pick another spot on the board if the space is invalid
  4. As a User I need to be able to see if I won Horizontally after placing my marker
  5. As a User I need to be able to see if I won Vertically after placing my marker
  6. As a User I need to be able to see if I won Diagonally after placing my marker
  7. As a User I should be able to choose whether or not I want to play again or not
  8. As a User I need to be able to know which players turn it is
  9. As a User I need to know the game is over once a winner has been determined
  10. As a User I need to know if the game ended in a tie/draw
  11. As a User I need to be able to pick if I want to play using the Fast or Memory Efficient version of the game
  12. As a User I need to be able to pick the number of Columns and Rows I want the board to be
  13. As a User I need to be able to pick the number of Tokens in a row needed to win
- **Non-Functional Requirements:**
  1. Creates a Board
  2. Alternates between Players
  3. Code will be written in the Java programming language
  4. 'X' always goes first
  5. Coordinate 0,0 is the top left of the board
  6. Computer must have Java Swing installed (for GUI features)
  7. Will display GUI window(s) when program is run

Design

GameScreen.java:

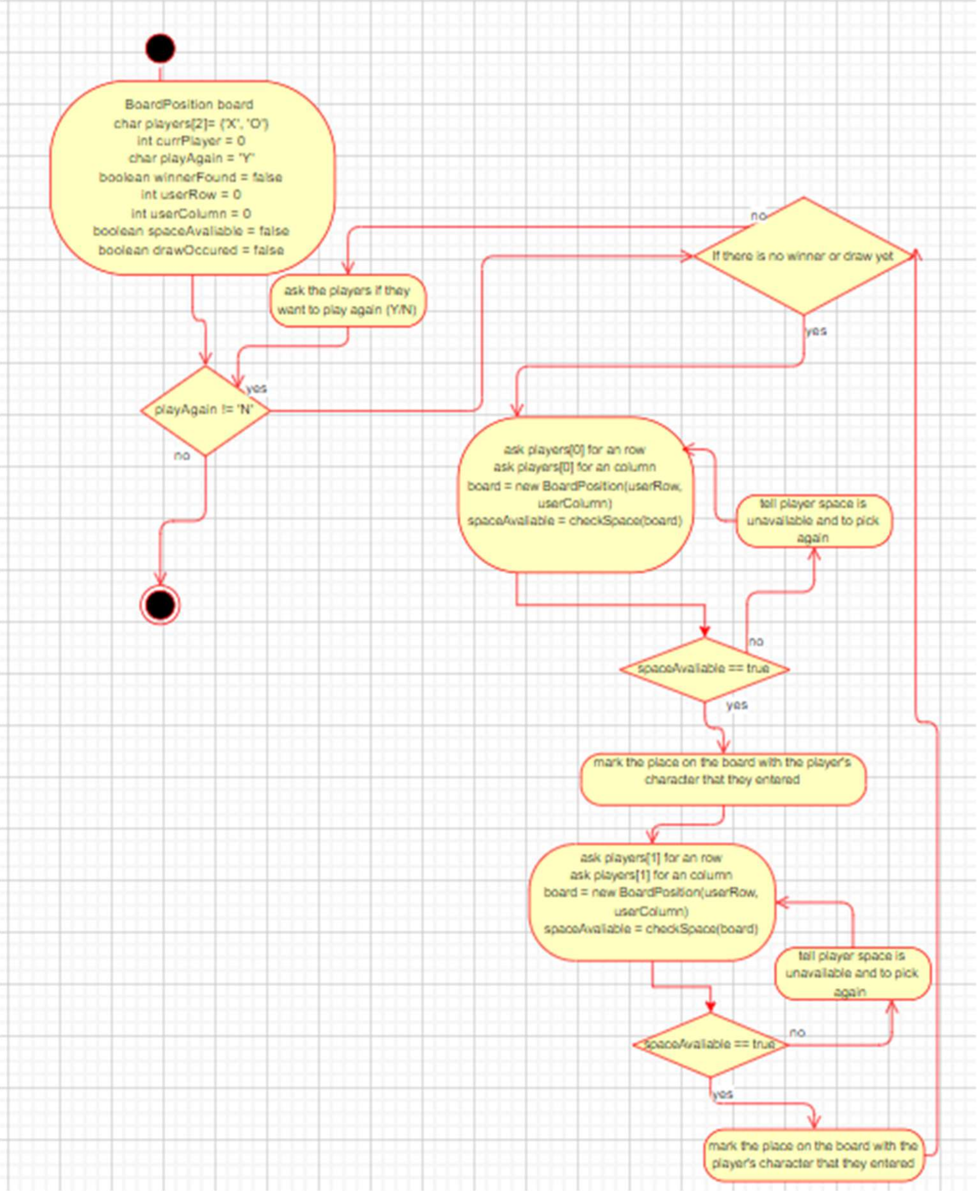


GameScreen

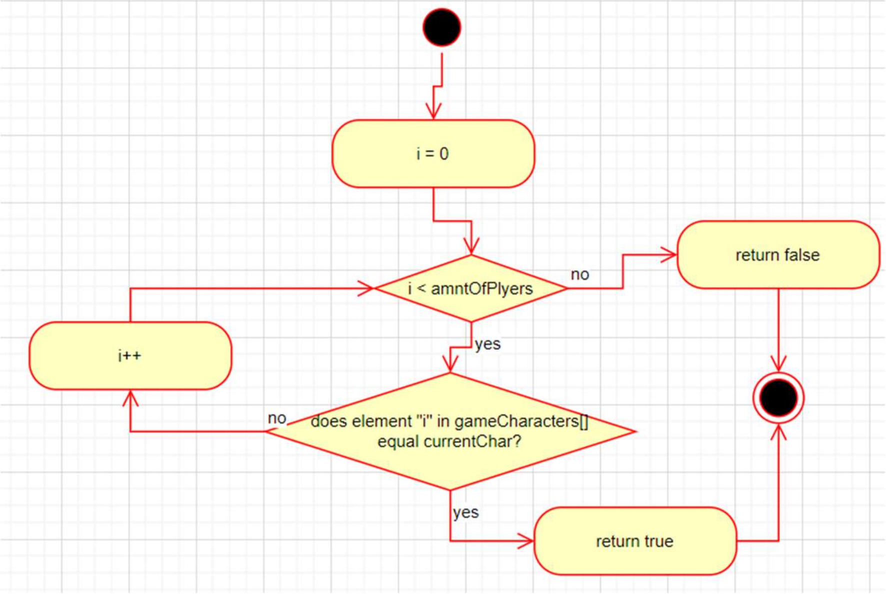
+ minPlayers : int
+ maxPlayers: int
+ minNumberOfRowsColumns: int
+ maxNumberOfRowsColumns: int
+ maxTokensNeeded: int
+ minTokensNeeded: int

+ main(args: String[]): void
+ doesCharExist(gameCharacters: char[], amntOfPlyrs: int, currentChar: char): boolean

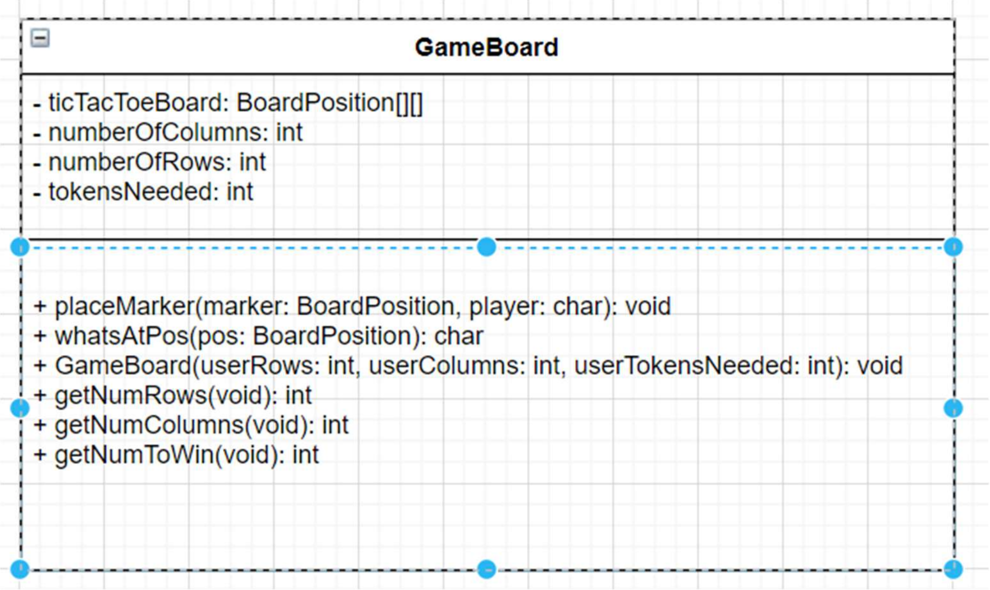
1. public static void main(String args[])


BoardPosition board
char players[2]= {'X', 'O'}
int currPlayer = 0
char playAgain = 'Y'
boolean winnerFound = false
int userRow = 0
int userColumn = 0
boolean spaceAvaliable = false
boolean drawOccured = false
no
If there is no winner or draw yet
ask the players if they want to play again (Y/N)
yes
yes
playAgain != 'N'
no
ask players[0] for an row
ask players[0] for an column
board = new BoardPosition(userRow, userColumn)
spaceAvaliable = checkSpace(board)
tell player space is unavailable and to pick again
no
spaceAvaliable == true
yes
mark the place on the board with the player's character that they entered
ask players[1] for an row
ask players[1] for an column
board = new BoardPosition(userRow, userColumn)
spaceAvaliable = checkSpace(board)
tell player space is unavailable and to pick again
no
spaceAvaliable == true
yes
mark the place on the board with the player's character that they entered

2. public static boolean doesCharExist(char[] gameCharacters, int amntOfPlyers, char currentChar)



GameBoard.Java

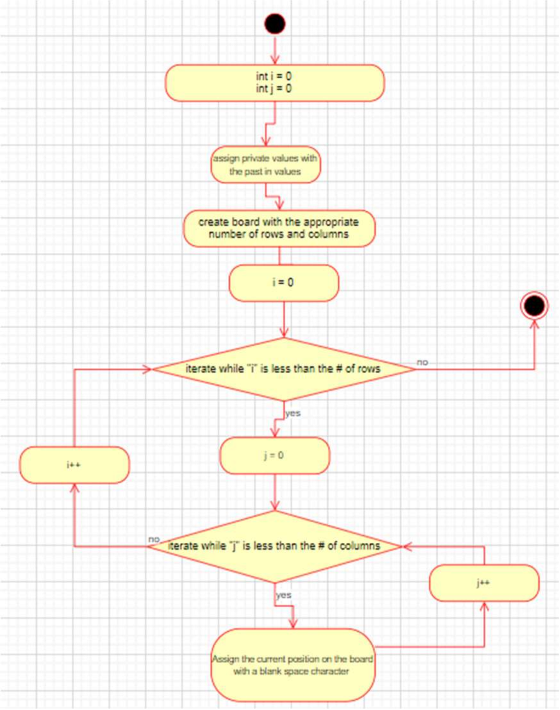


GameBoard

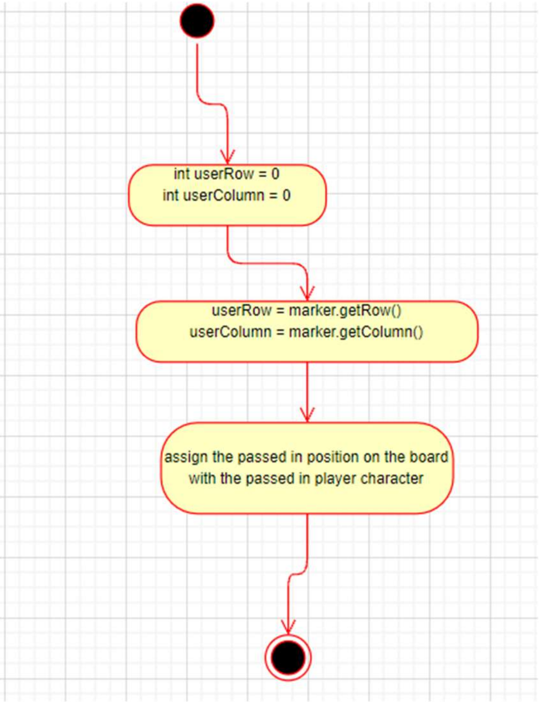
- ticTacToeBoard: BoardPosition[][]
- numberOfColumns: int
- numberOfRows: int
- tokensNeeded: int

+ placeMarker(marker: BoardPosition, player: char): void
+ whatsAtPos(pos: BoardPosition): char
+ GameBoard(userRows: int, userColumns: int, userTokensNeeded: int): void
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int

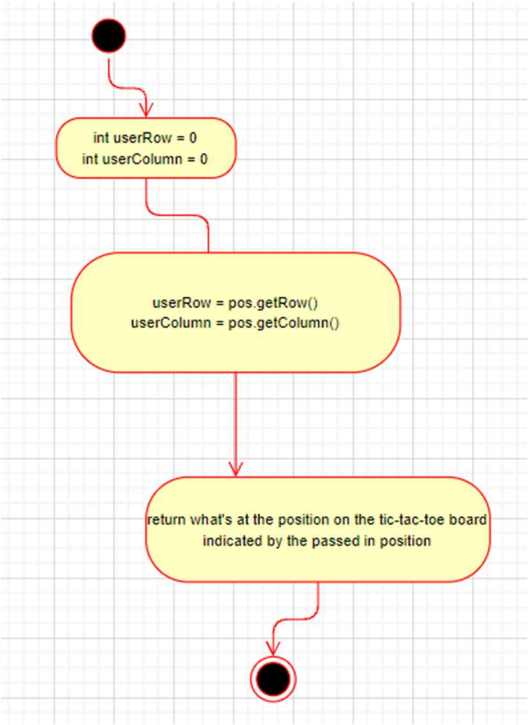
1. public GameBoard()


int i = 0
int j = 0
assign private values with the past in values
create board with the appropriate number of rows and columns
i = 0
iterate while "i" is less than the # of rows
no
yes
j = 0
i++
no
iterate while "j" is less than the # of columns
yes
j++
Assign the current position on the board with a blank space character

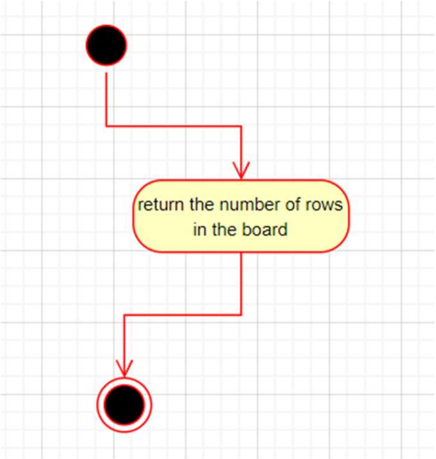
2. public void placeMarker(BoardPosition marker, char player)


int userRow = 0
int userColumn = 0
userRow = marker.getRow()
userColumn = marker.getColumn()
assign the passed in position on the board with the passed in player character

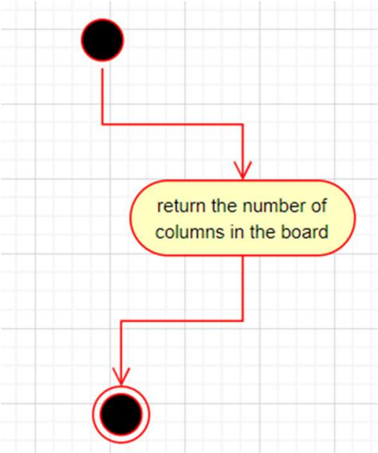
3. public char whatsAtPos(BoardPosition pos)


int userRow = 0
int userColumn = 0
userRow = pos.getRow()
userColumn = pos.getColumn()
return what's at the position on the tic-tac-toe board
indicated by the passed in position

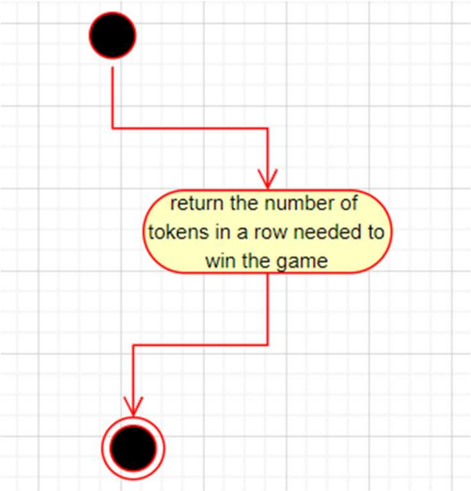
4. public int getNumRows()



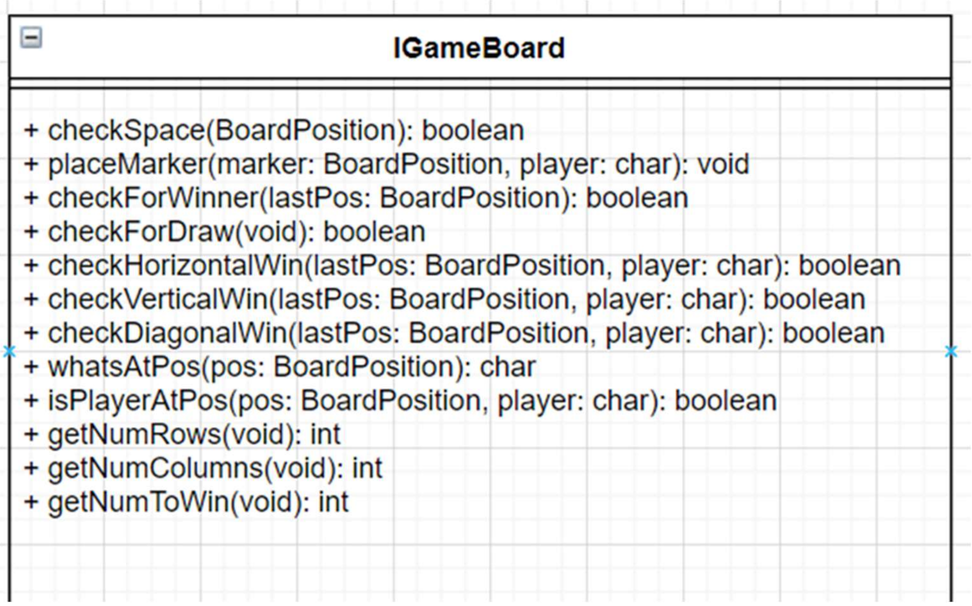
5. public int getNumColumns()



6. public int getNumToWin()



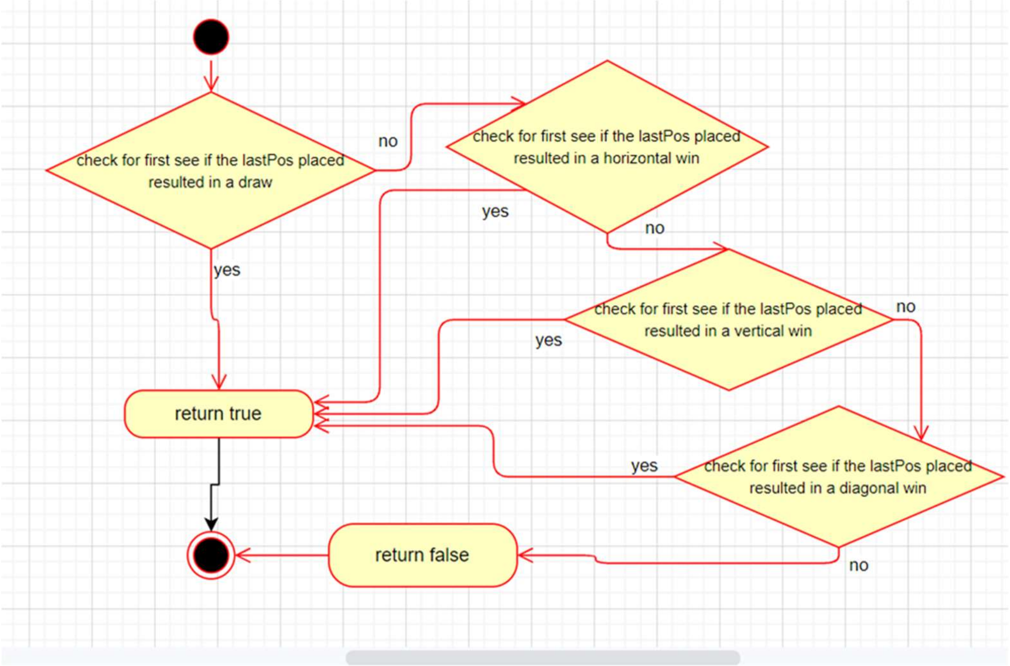
IGameBoard.java:



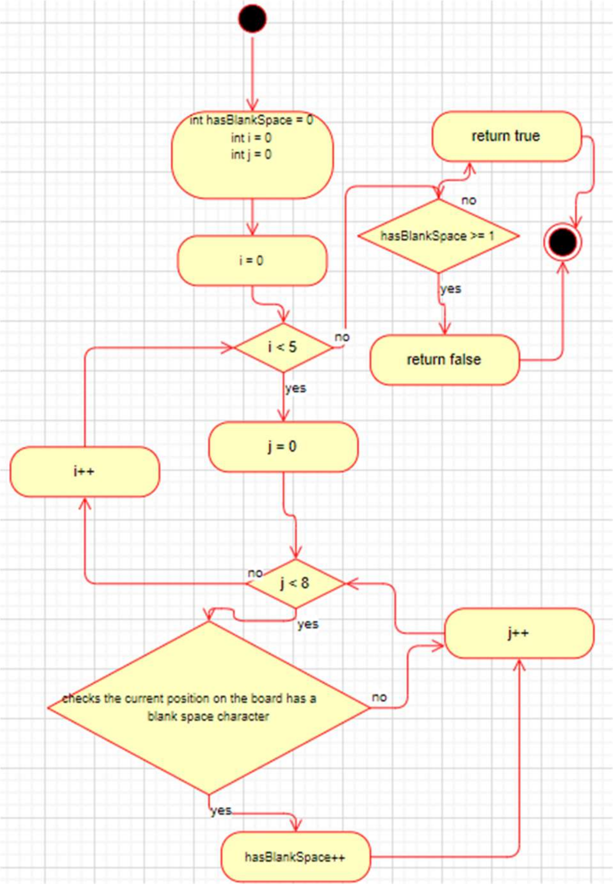
IGameBoard

+ checkSpace(BoardPosition): boolean
+ placeMarker(marker: BoardPosition, player: char): void
+ checkForWinner(lastPos: BoardPosition): boolean
+ checkForDraw(void): boolean
+ checkHorizontalWin(lastPos: BoardPosition, player: char): boolean
+ checkVerticalWin(lastPos: BoardPosition, player: char): boolean
+ checkDiagonalWin(lastPos: BoardPosition, player: char): boolean
+ whatsAtPos(pos: BoardPosition): char
+ isPlayerAtPos(pos: BoardPosition, player: char): boolean
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int

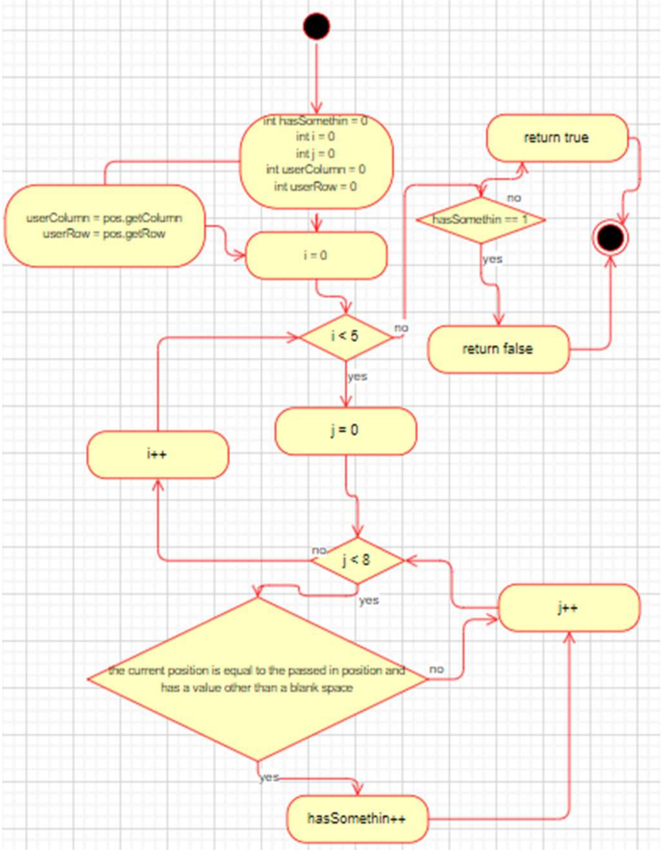
1. default public boolean checkForWinner(BoardPosition lastPos)


check for first see if the lastPos placed resulted in a draw
no
check for first see if the lastPos placed resulted in a horizontal win
yes
no
yes
check for first see if the lastPos placed resulted in a vertical win
no
yes
return true
yes
check for first see if the lastPos placed resulted in a diagonal win
return false
no

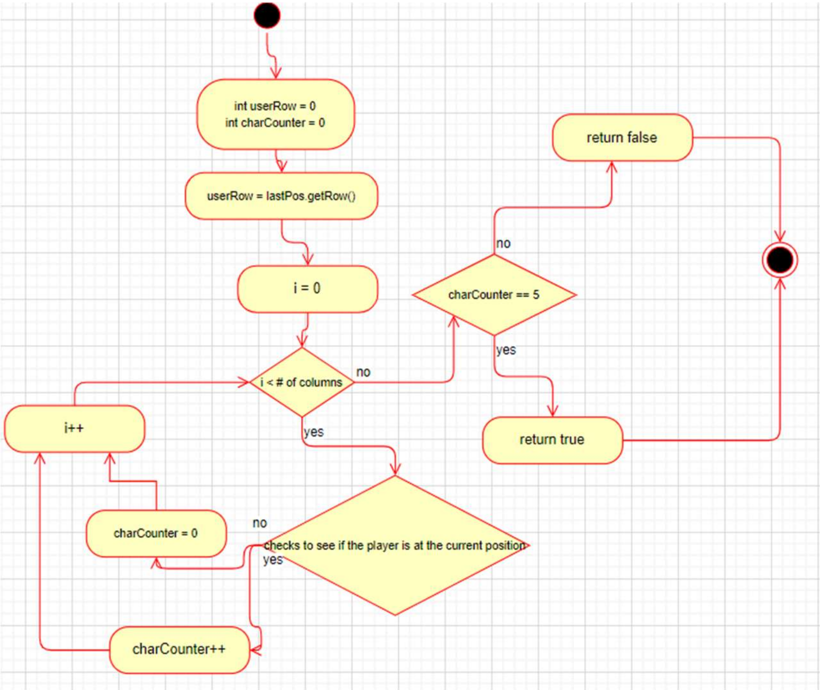
2. default public boolean checkForDraw()



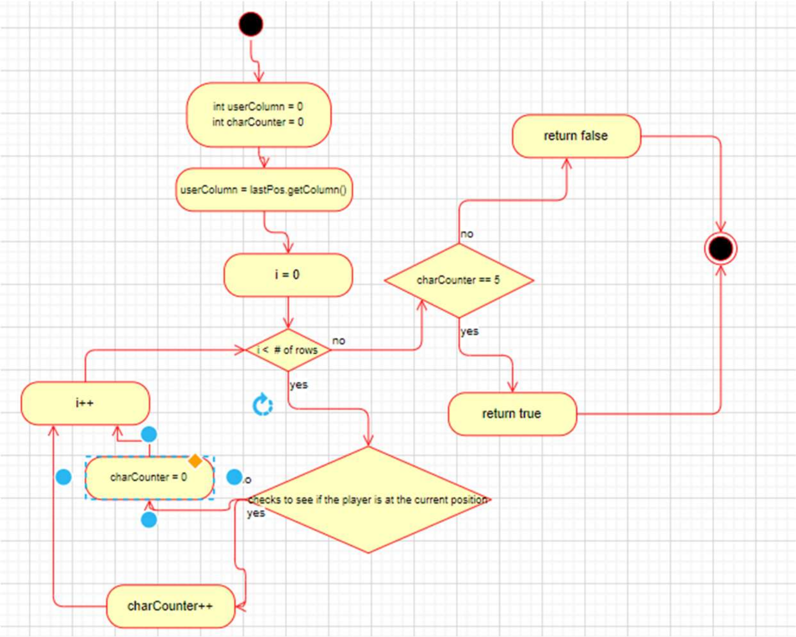
3. default public boolean checkSpace(BoardPosition pos)



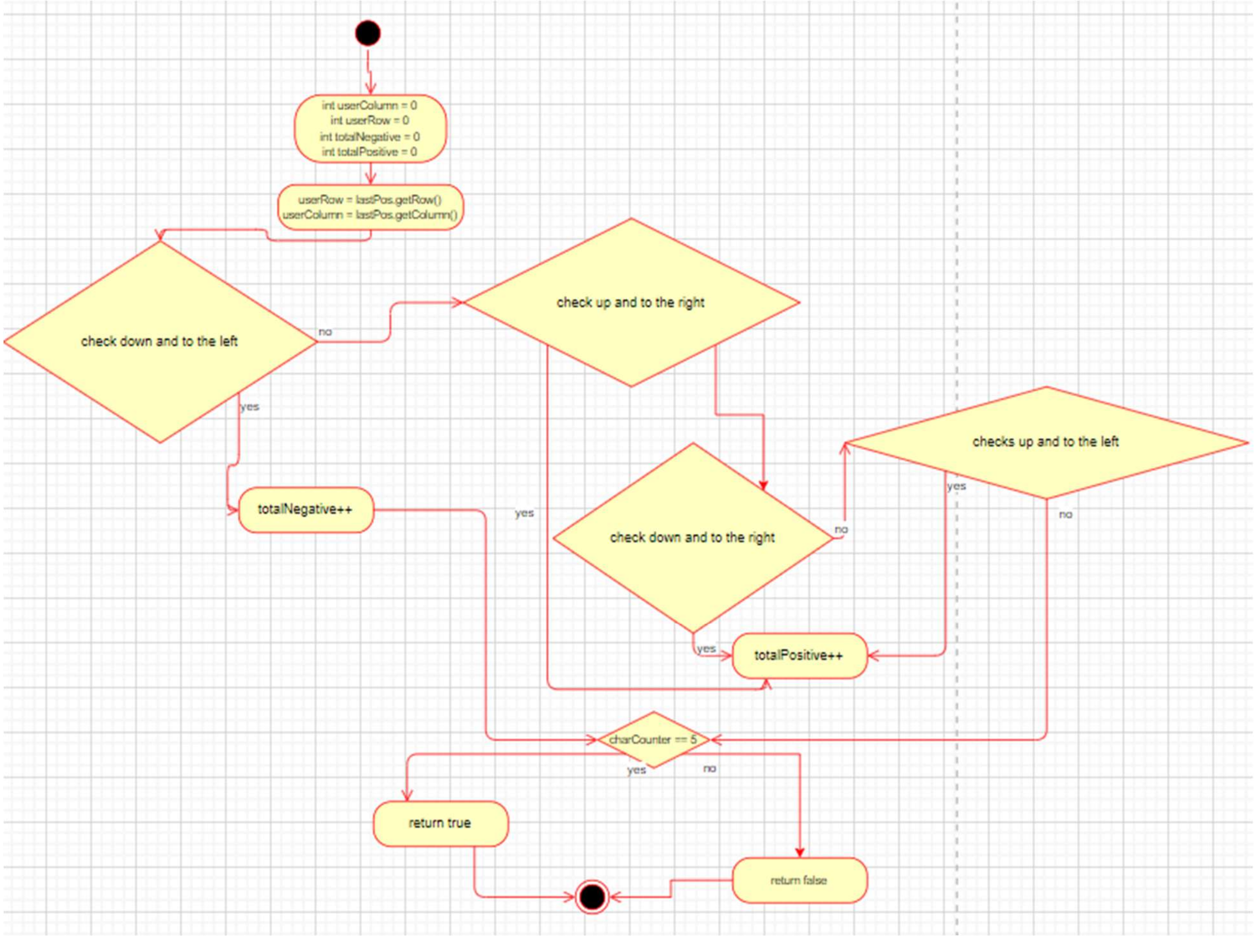
4. default public boolean checkHorizontalWin(BoardPosition lastPos, char player)



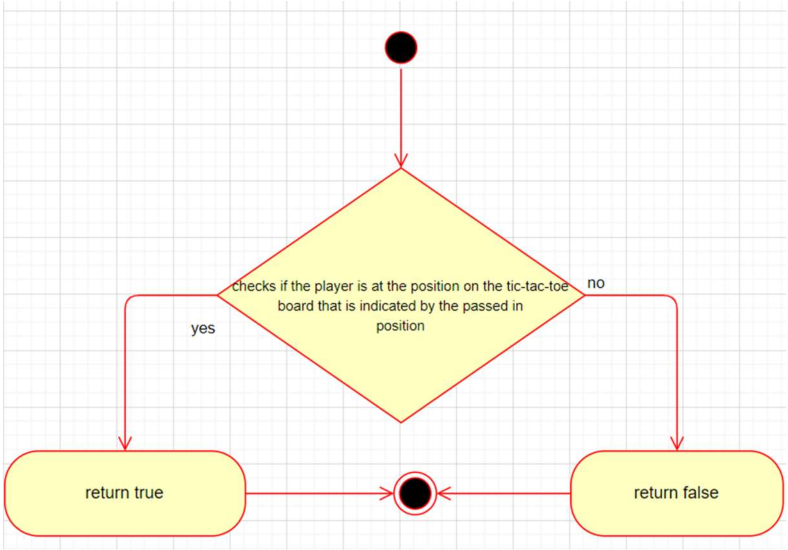
5. default public boolean checkVerticalWin(BoardPosition lastPos, char player)



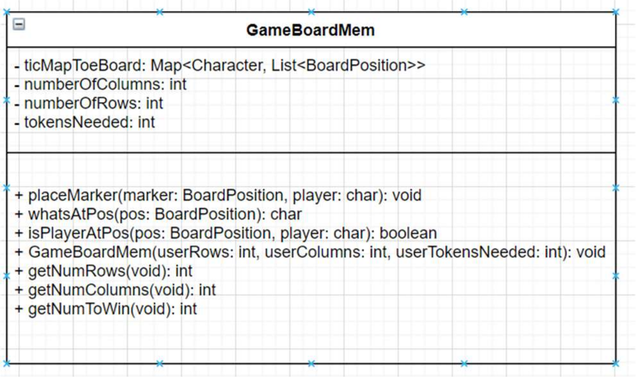
6. default public boolean checkDiagonalWin(BoardPosition lastPos, char player)



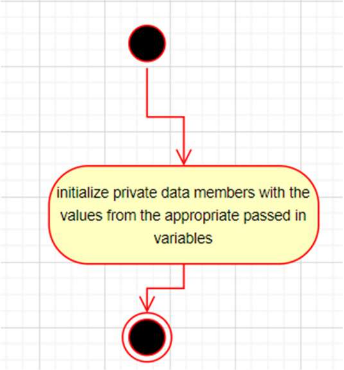
7. default public boolean isPlayerAtPos(BoardPosition pos, char player)


checks if the player is at the position on the tic-tac-toe board that is indicated by the passed in position
no
yes
return true
return false

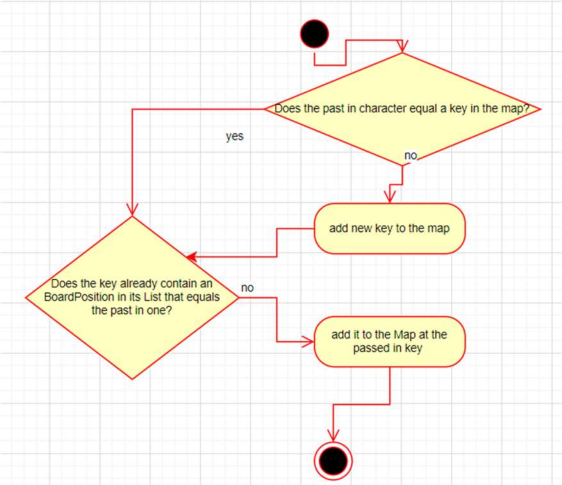
GameBoardMem.java:


GameBoardMem
- ticMapToeBoard: Map<Character, List<BoardPosition>>
- numberOfColumns: int
- numberOfRows: int
- tokensNeeded: int
+ placeMarker(marker: BoardPosition, player: char): void
+ whatsAtPos(pos: BoardPosition): char
+ isPlayerAtPos(pos: BoardPosition, player: char): boolean
+ GameBoardMem(userRows: int, userColumns: int, userTokensNeeded: int): void
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int

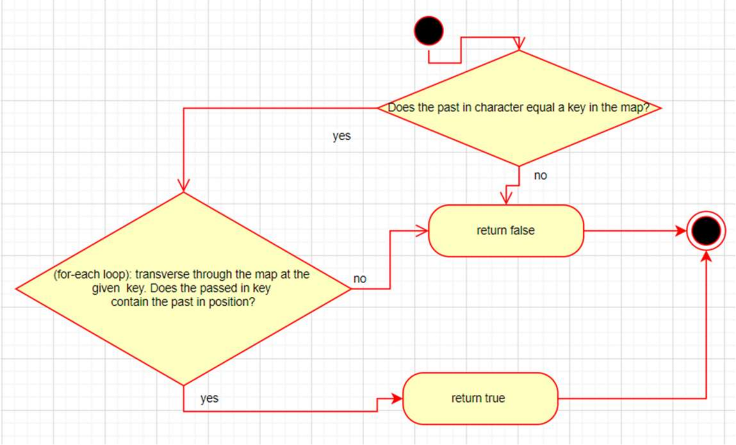
1. public GameBoardMem(int userRows, int userColumns, int userTokensNeeded)


initialize private data members with the values from the appropriate passed in variables

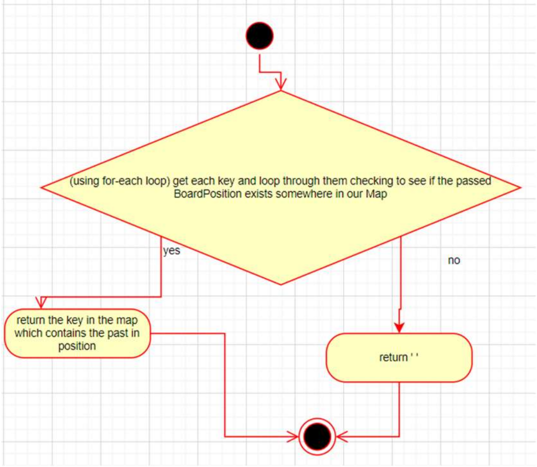
2. public void placeMarker(BoardPosition marker, char player)


Does the past in character equal a key in the map?
yes
no
add new key to the map
Does the key already contain an BoardPosition in its List that equals the past in one?
no
add it to the Map at the passed in key

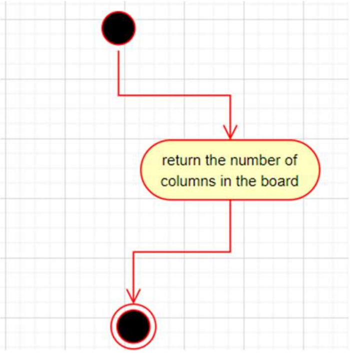
3. public boolean isPlayerAtPos(BoardPosition pos, char player)


Does the past in character equal a key in the map?
yes
no
return false
(for-each loop): transverse through the map at the given key. Does the passed in key contain the past in position?
no
yes
return true

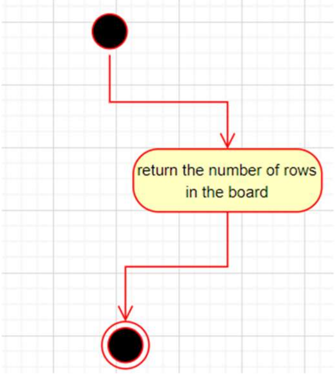
4. public char whatsAtPos(BoardPosition pos)


(using for-each loop) get each key and loop through them checking to see if the passed BoardPosition exists somewhere in our Map
yes
no
return the key in the map which contains the past in position
return ' '

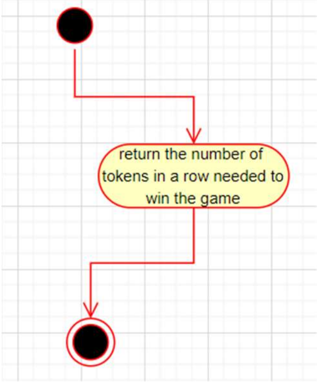
5. public int getNumColumns()



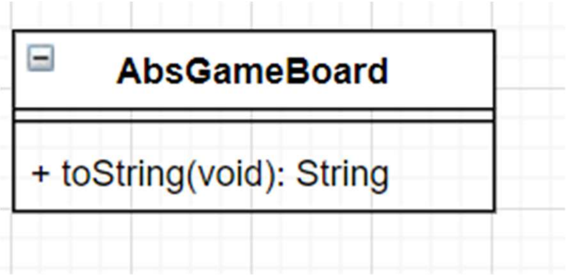
6. public int getNumRows()



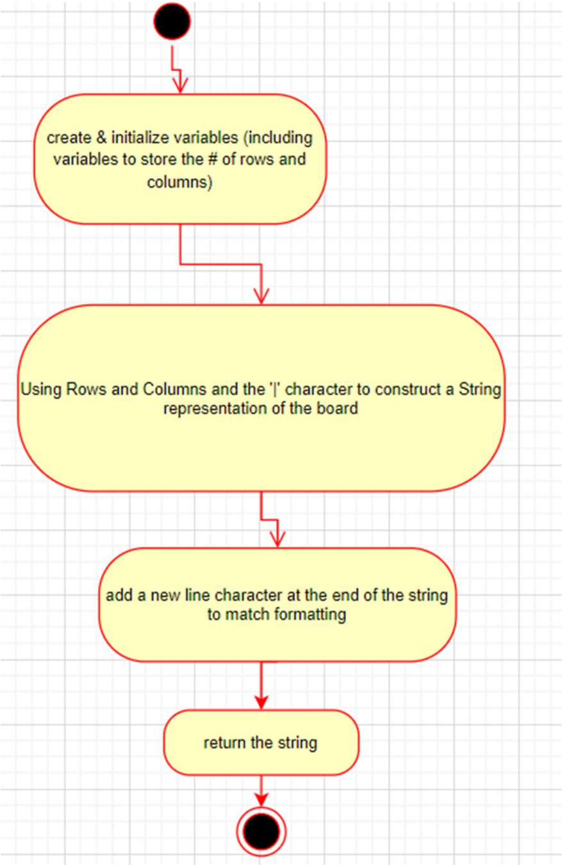
7. public int getNumToWin()



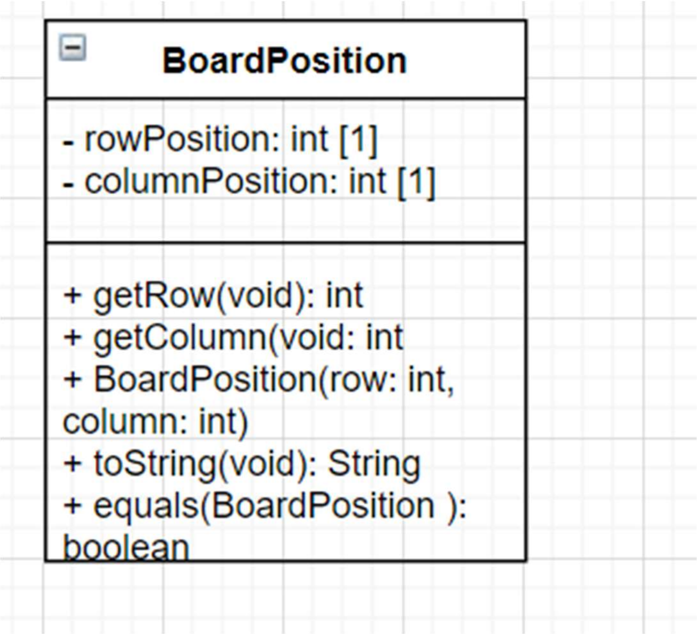
AbsGameBoard.java:



1. (overridden) String toString()


create & initialize variables (including variables to store the # of rows and columns)
Using Rows and Columns and the '|' character to construct a String representation of the board
add a new line character at the end of the string to match formatting
return the string

BoardPosition:

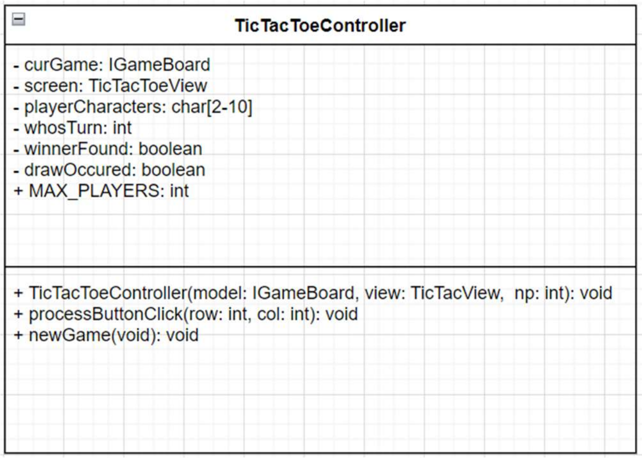


BoardPosition

- rowPosition: int [1]
- columnPosition: int [1]

+ getRow(void): int
+ getColumn(void: int
+ BoardPosition(row: int, column: int)
+ toString(void): String
+ equals(BoardPosition): boolean

TicTacToeController:

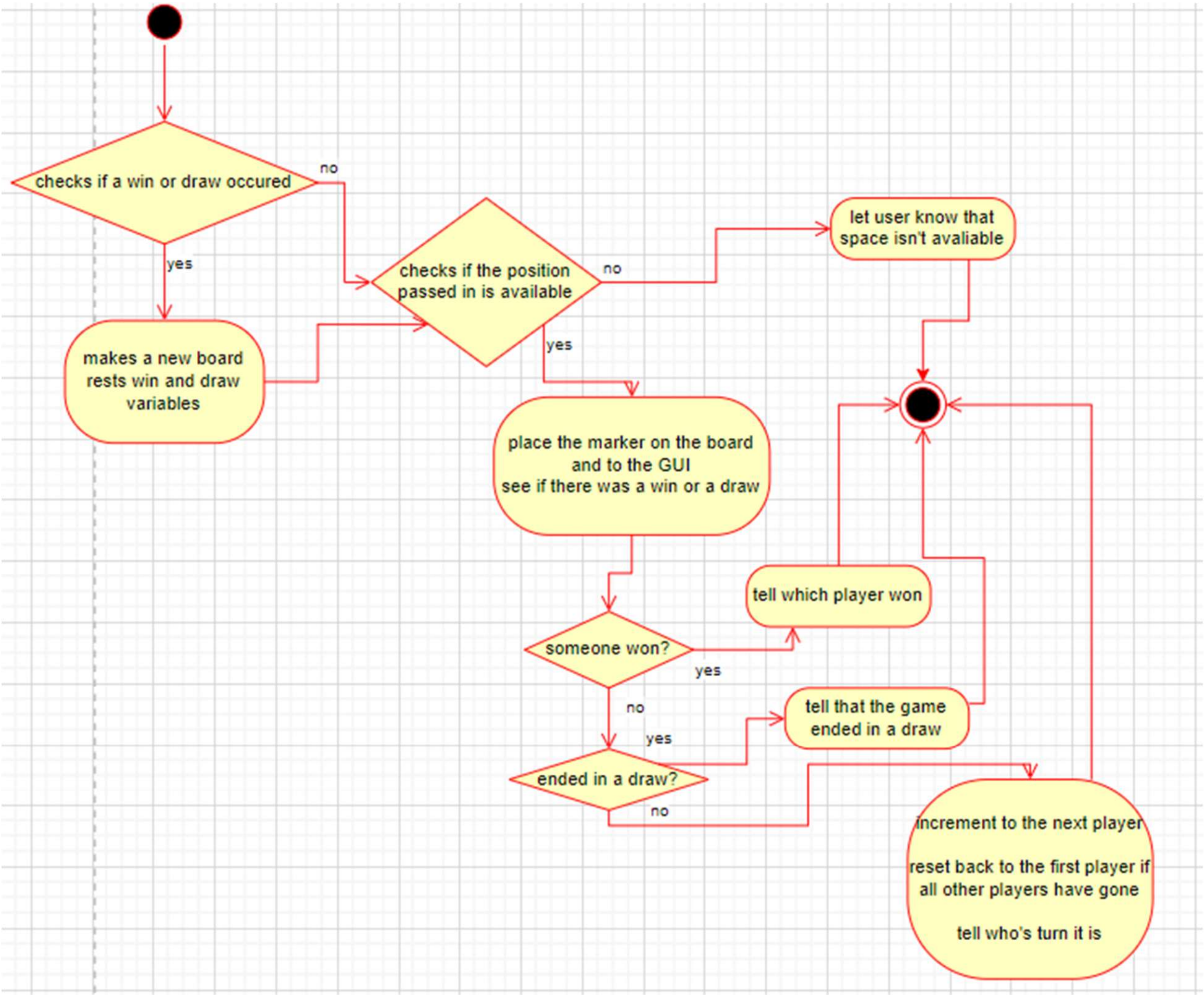


TicTacToeController

- curGame: IGameBoard
- screen: TicTacToeView
- playerCharacters: char[2-10]
- whosTurn: int
- winnerFound: boolean
- drawOccured: boolean
+ MAX_PLAYERS: int

+ TicTacToeController(model: IGameBoard, view: TicTacView, np: int): void
+ processButtonClick(row: int, col: int): void
+ newGame(void): void

public void processButtonClick(int row, int col)



checks if a win or draw occured
no
yes
makes a new board rests win and draw variables
checks if the position passed in is available
no
yes
let user know that space isn't avaliable
place the marker on the board and to the GUI see if there was a win or a draw
someone won?
yes
tell which player won
no
yes
tell that the game ended in a draw
ended in a draw?
no
increment to the next player
reset back to the first player if all other players have gone
tell who's turn it is

Testing:

GameBoard(int userRows, int userColumns, int userTokensNeeded)

<table>
<tr><th>Input:</th><th>Output:</th><th>Reason:</th></tr>
<tr>
<td>testRows = 3
testColumns = 3
testTokensNeeded = 3</td>
<td><table>
<tr><th></th><th>0</th><th>1</th><th>2</th></tr>
<tr><td>0</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table>
testBoard.getNumRows() == 3
testBoard.getNumColumns() == 3
testBoard.getNumToWin() == 3</td>
<td>This test case is unique and distinct because it tests to see if the constructor can make the smallest board possible
<b>Function Name:</b>
testContstructor_smallest_board</td>
</tr>
</table>

GameBoard(int userRows, int userColumns, int userTokensNeeded)

Input:	**Output:**	**Reason:**

testRows = 100
testColumns = 100
testTokensNeeded = 25
	testBoard = 100 x 100
	testBoard.getNumRows() == 100
	testBoard.getNumColumns() == 100
	testBoard.getNumToWin() == 25
		This test case is unique and distinct because it tests to see if the constructor can make the largest board possible
		Function Name:
		testConstructor_largest_board

GameBoard(int userRows, int userColumns, int userTokensNeeded)

Input:	**Output:**	**Reason:**

testRows = 12
testColumns = 15
testTokensNeeded = 5
	testBoard = 12 x 15
	testBoard.getNumRows() == 12
	testBoard.getNumColumns() == 15
	testBoard.getNumToWin() == 5
		This test case is unique and distinct because it tests to see if the constructor can make a board with an unequal number of rows and columns
		Function Name:
		testConstructor_uneven_rows_columns

boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkSpace(pos) == true</td>
<td><b>Reason:</b>
This test case is unique and distinct because it makes sure that it can detect and return if a specific space is available or not

<b>Function Name:</b>
testCheckSpace_board_empty</td>
</tr>
</table>

boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td></td><td>X</td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table></td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkSpace(pos) == false</td>
<td><b>Reason:</b>
This test case is unique and distinct because it makes sure the function can accurately state that there is already a token at that spot

<b>Function Name:</b>
testCheckSpace_space_unavailable</td>
</tr>
</table>

boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td></td><td>X</td><td></td></tr><tr><td>1</td><td></td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td></td><td>X</td><td></td></tr><tr><td>3</td><td></td><td></td><td>X</td><td></td></tr></table></td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkSpace(0,2)== false
testBoard.checkSpace(1,2)== false
testBoard.checkSpace(2,2)== false
testBoard.checkSpace(3,2)== false

(would just be passing in “pos” each time but for clarity showed All 4 points)</td>
<td><b>Reason:</b>
This test case is unique and distinct because it checks a whole row to see if its available and returns false each time indicating that no space in that column is available

<b>Function Name:</b>
testCheckSpace_column_check</td>
</tr>
</table>

boolean checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr><td><b>Input:</b>
State:<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr><tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr></table>lastPos(1,1)
testBoard.getNumToWin() == 3
Player = ‘X’</td><td><b>Output:</b>

Board is unchanged

testBoard.checkHorizontalWin(lastPos, player) == false;</td><td><b>Reason:</b>
This test case is unique and distinct because it ensures that a win isn’t register if nothing is on the board and 5-in-a-row hasn’t been achieved horizontally

<b>Function Name:</b>
testHorzWin_empty_board</td></tr>
</table>

boolean checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr><td><b>Input:</b>

State:<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr><tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>1</b></td><td><b>X</b></td><td><b>X</b></td><td><b>X</b></td><td></td></tr><tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr></table>lastPos(1,1)
testBoard.getNumToWin() == 3
Player = ‘X’</td><td><b>Output:</b>

Board is unchanged

testBoard.checkHorizontalWin(lastPos, player) == true;</td><td><b>Reason:</b>
This test case is unique and distinct because it tests to make sure that a horizontal win can be achieved

<b>Function Name:</b>
testHorzWin_win_achieved</td></tr>
</table>

boolean checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr><td><b>Input:</b>

State:<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr><tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>1</b></td><td><b>X</b></td><td><b>X</b></td><td><b>T</b></td><td><b>X</b></td></tr><tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr></table>lastPos(1,1)
testBoard.getNumToWin() == 3
Player = ‘X’</td><td><b>Output:</b>

Board is unchanged

testBoard.checkHorizontalWin(lastPos, player) == false;</td><td><b>Reason:</b>
This test case is unique and distinct because it tests to make sure that a horizontal win isn’t registered unless ‘x’ amount of the same characters are in a row horizontally

<b>Function Name:</b>
testHorzWin_almost_won</td></tr>
</table>

boolean checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td><b>X</b></td><td><b>X</b></td><td><b>X</b></td><td><b>X</b></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos(1,1)
testBoard.getNumToWin() == 3
Player = ‘X’</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkHorizontalWin(lastPos, player) == false;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to see that a win will still be achieved and returned even if more than the required number tokens of the same character are present on the board

<b>Function Name:</b>
testHorzWin_more_than_needed</td>
</tr>
</table>

boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos(1,1)
testBoard.getNumToWin() == 3
Player = ‘X’</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkVerticalWin(lastPos, player) == false;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it ensures that a win isn’t register if nothing is on the board and 5-in-a-row hasn’t been achieved vertically

<b>Function Name:</b>
testVertWin_empty_board</td>
</tr>
</table>

boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos(1,1)
testBoard.getNumToWin() == 3
Player = ‘X’</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkVerticalWin(lastPos, player) == true;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to make sure that a vertical win can be achieved

<b>Function Name:</b>
testVertWin_win_achieved</td>
</tr>
</table>

boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td><b>T</b></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
</table>
lastPos(1,1)
testBoard.getNumToWin() == 3
Player = ‘X’</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkVerticalWin(lastPos, player) == false;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to make sure that a horizontal win isn’t registered unless ‘x’ amount of the same characters are in a row vertically

<b>Function Name:</b>
testVertWin_almost_won</td>
</tr>
</table>

boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
</table>
lastPos(1,1)
testBoard.getNumToWin() == 3
Player = ‘X’</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkVerticalWin(lastPos, player) == false;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to see that a win will still be achieved and returned even if more than the required number tokens of the same character are present on the board

<b>Function Name:</b>
testVertWin_more_than_needed</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos(1,1)
Player = ‘X ’
testBoard.getNumToWin() == 3</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkDiagonalWin(lastPos, player) == false;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to make sure that a call to checkDiagonalWin doesn’t return a win by default

<b>Function Name:</b>
testDiagWin_empty_board</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td><b>X</b></td><td></td><td><b>T</b></td><td><b>T</b></td></tr>
<tr><td><b>1</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td><b>X</b></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos = (2,2)
player = ‘X’
testBoard.getNumToWin() == 3</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkDiagonalWin(lastPos, player) == true;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to see if a diagonal win going up and to left and down and to right can be achieved

<b>Function Name:</b>
testDiagWin_dwnright_upleft_win_achieved</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td><b>X</b></td><td></td><td><b>T</b></td><td><b>T</b></td></tr>
<tr><td><b>1</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td><b>T</b></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td><b>X</b></td></tr>
</table>
lastPos = (0,3)
player = ‘T’
testBoard.getNumToWin() == 3</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkDiagonalWin(lastPos, player) == false;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it makes sure that a win going up and to the left and down and to the right won’t be achieved Unless all ‘x’ number of players marker are the same player

<b>Function Name:</b>
testDiagWin_dwnright_upleft_almost_won</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td><b>T</b></td><td><b>T</b></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td><b>X</b></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td><b>X</b></td></tr>
</table>
lastPos = (1,1)
player = ‘X’
testBoard.getNumToWin() == 3</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkDiagonalWin(lastPos, player) == true;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to make sure that a win going up and to the left and down and to the right can be achieved if ended (or started) from the bottom right corner

<b>Function Name:</b>
testDiagWin_dwnright_upleft_bottom_right_corner</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><td></td><td><b>0</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td></tr>
<tr><td><b>0</b></td><td><b>T</b></td><td><b>T</b></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td><b>X</b></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td><b>X</b></td><td></td><td></td><td></td></tr>
</table>

lastPos = (1,2)
player = ‘X’
testBoard.getNumToWin() == 3</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkDiagonalWin(lastPos, player) == true;</td>
<td><b>Reason:</b>
This test case is unique and distinct because tests to see if a diagonal win going up and to right and down and to left can be achieved

<b>Function Name:</b>
testDiagWin_upright_dwnleft_win_achieved</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><td></td><td><b>0</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td></tr>
<tr><td><b>0</b></td><td><b>T</b></td><td><b>T</b></td><td></td><td><b>X</b></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td><b>T</b></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td><b>X</b></td><td></td><td></td><td></td></tr>
</table>
lastPos = (0,0)
player = ‘T’
testBoard.getNumToWin() == 3</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkDiagonalWin(lastPos, player) == false;</td>
<td><b>Reason:</b>
This test case is unique and distinct because makes sure that a win going up and to the right and down and to the left won’t be achieved
Unless all ‘x’ number of players marker are the same player

<b>Function Name:</b>
testDiagWin_upright_dwnleft_almost_won</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><td></td><td><b>0</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td></tr>
<tr><td><b>0</b></td><td><b>T</b></td><td><b>T</b></td><td></td><td><b>X</b></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td><b>X</b></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td><b>X</b></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos = (2,1)
player = ‘X’
testBoard.getNumToWin() == 3</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkDiagonalWin(lastPos, player) == true;</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to make sure that a win going up and to the left and down and to the right can be achieved if ended (or started) from the upper right corner

<b>Function Name:</b>
testDiagWin_upright_dwnleft_upper_right_corner</td>
</tr>
</table>

boolean checkForDraw()

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkForDraw() == false</td>
<td><b>Reason:</b>
This test case is unique and distinct because it makes sure that a draw isn’t registered if nothing is on the board

<b>Function Name:</b>
testCheckForDraw_empty_board</td>
</tr>
</table>

boolean checkForDraw()

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td><b>X</b></td><td><b>T</b></td><td><b>X</b></td><td><b>T</b></td></tr>
<tr><td><b>1</b></td><td><b>T</b></td><td><b>X</b></td><td><b>X</b></td><td><b>X</b></td></tr>
<tr><td><b>2</b></td><td><b>X</b></td><td><b>X</b></td><td><b>T</b></td><td><b>X</b></td></tr>
<tr><td><b>3</b></td><td><b>X</b></td><td><b>T</b></td><td><b>X</b></td><td><b>X</b></td></tr>
</table>
</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkForDraw() == true</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to makes sure that a draw can be achieved

<b>Function Name:</b>
testCheckForDraw_draw_occurred</td>
</tr>
</table>

boolean checkForDraw()

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td><b>T</b></td><td><b>X</b></td><td><b>T</b></td></tr>
<tr><td><b>1</b></td><td><b>T</b></td><td><b>X</b></td><td><b>X</b></td><td><b>X</b></td></tr>
<tr><td><b>2</b></td><td><b>X</b></td><td><b>X</b></td><td><b>T</b></td><td><b>X</b></td></tr>
<tr><td><b>3</b></td><td><b>X</b></td><td><b>T</b></td><td><b>X</b></td><td><b>X</b></td></tr>
</table>
</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkForDraw() == false</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to see if a it will still return that a draw hasn’t occurred if there is a space available in the upper left corner

<b>Function Name:</b>
testCheckForDraw_upper_left_corner</td>
</tr>
</table>

boolean checkForDraw()

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td><b>X</b></td><td><b>T</b></td><td><b>X</b></td><td></td></tr>
<tr><td><b>1</b></td><td><b>T</b></td><td><b>X</b></td><td><b>X</b></td><td><b>X</b></td></tr>
<tr><td><b>2</b></td><td><b>X</b></td><td><b>X</b></td><td><b>T</b></td><td><b>X</b></td></tr>
<tr><td><b>3</b></td><td><b>X</b></td><td><b>T</b></td><td><b>X</b></td><td><b>X</b></td></tr>
</table>
</td>
<td><b>Output:</b>

Board is unchanged

testBoard.checkForDraw() == false</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to see if a it will still return that a draw hasn’t occurred if there is a space available in the upper right corner

<b>Function Name:</b>
testCheckForDraw_upper_right_corner</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr><td><b>Input:</b>
State:
<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr><tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr></table>Pos = (0,0)</td><td><b>Output:</b>

Board is unchanged

testBoard.whatsAtPos(pos) == ‘ ‘</td><td><b>Reason:</b>
This test case is unique and distinct because it makes sure that a blank space is returned if no players character is at the checked position

<b>Function Name:</b>
testWhatsAtPos_empty_board</td></tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr><td><b>Input:</b>
State:
<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr><tr><td><b>0</b></td><td><b>X</b></td><td></td><td></td><td></td></tr><tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr></table>Pos = (0,0)</td><td><b>Output:</b>

Board is unchanged

testBoard.whatsAtPos(pos) == ‘X‘</td><td><b>Reason:</b>
This test case is unique and distinct because it makes sure that if a players character is at the past in position it can detect and return that character
<b>Function Name:</b>
testWhatsAtPos_upper_left_corner</td></tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr><td><b>Input:</b>
State:
<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr><tr><td><b>0</b></td><td><b>T</b></td><td></td><td></td><td></td></tr><tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr></table>Pos(0,0)</td><td><b>Output:</b>

Board is unchanged

testBoard.whatsAtPos(pos) == ‘T’</td><td><b>Reason:</b>
This test case is unique and distinct because it tests to see if the method can return players characters correctly if they aren’t ‘X’ or ‘O’

<b>Function Name:</b>
testWhatsAtPos_not_x_or_o</td></tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr><td><b>Input:</b>
State:
<table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr><tr><td><b>0</b></td><td><b>X</b></td><td></td><td></td><td></td></tr><tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr><tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr></table></td><td><b>Output:</b>

Board is unchanged

testBoard.whatsAtPos(pos) == ‘ ‘</td><td><b>Reason:</b>
This test case is unique and distinct because it tests whether or not the method would return a blank character (it’s checking the right position)

<b>Function Name:</b>
testWhatsAtPos_next_to_player</td></tr>
</table>

Pos(0,1)		

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td></td><td><b>0</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td></tr>
<tr><td><b>0</b></td><td><b>X</b></td><td><b>T</b></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos(0,1)</td>
<td><b>Output:</b>
Board is unchanged
testBoard.whatsAtPos(pos) == ‘T‘</td>
<td><b>Reason:</b>
This test case is unique and distinct because it makes sure the method will return the correct character even if there were markers of 2 different player on the board.
<b>Function Name:</b>
testWhatsAtPos_two_players</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td></td><td><b>0</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td><b>X</b></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos(0,3)
Player = ‘X’</td>
<td><b>Output:</b>
Board is unchanged
testBoard.isPlayerAtPos(pos,player) == true</td>
<td><b>Reason:</b>
This test case is unique and distinct because it makes sure that if a players character is at the upper right corner of the board, It can detect and return, that players character is there
<b>Function Name:</b>
testIsPlayerAtPos_upper_right_corner</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td></td><td><b>0</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td></tr>
<tr><td><b>0</b></td><td><b>X</b></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos(0,0)
Player = ‘X’</td>
<td><b>Output:</b>
Board is unchanged
testBoard.isPlayerAtPos(pos,player) == true</td>
<td><b>Reason:</b>
This test case is unique and distinct because it makes sure that if a players character is at the upper left corner of the board it, it can detect and return, that players character is there.
<b>Function Name:</b>
testIsPlayerAtPos_upper_left_corner</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td></td><td><b>0</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td></tr>
<tr><td><b>0</b></td><td><b>T</b></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos(0,0)
Player = ‘T’</td>
<td><b>Output:</b>
Board is unchanged
testBoard.isPlayerAtPos(pos,player) == true</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to see if the method can detect players characters correctly if they aren’t ‘X’ or ‘O’
<b>Function Name:</b>
testIsPlayerAtPos_not_x_or_o</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td></td><td><b>0</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td></tr>
<tr><td><b>0</b></td><td><b>X</b></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos(0,1)
Player = ‘X’</td>
<td><b>Output:</b>
Board is unchanged
testBoard.isPlayerAtPos(pos,player) == false</td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests whether or not the method is checking the right position
<b>Function Name:</b>
testIsPlayerAtPos_next_to_player</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td></td><td><b>0</b></td><td><b>1</b></td><td><b>2</b></td><td><b>3</b></td></tr>
<tr><td><b>0</b></td><td><b>X</b></td><td><b>T</b></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
Pos(0,1)
Player = ‘T’</td>
<td><b>Output:</b>
Board is unchanged
testBoard.isPlayerAtPos(pos,player) == true</td>
<td><b>Reason:</b>
This test case is unique and distinct because it makes sure the method will detect the correct character even if there were markers of 2 different player on the board.
<b>Function Name:</b>
testIsPlayerAtPos_two_players</td>
</tr>
</table>

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
Marker = (3,3)
Player = ‘X’</td>
<td><b>Output:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td><b>X</b></td></tr>
</table></td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to make sure that an marker can be placed in the bottom right corner

<b>Function Name:</b>
testPlaceMarker_bottom_right_corner</td>
</tr>
</table>

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
Marker = (0,0)
Player = ‘X’</td>
<td><b>Output:</b>

State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td><b>X</b></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to make sure that an a marker can be placed in the upper left corner

<b>Function Name:</b>
testPlaceMarker_upper_left_corner</td>
</tr>
</table>

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table>
Marker = (2,2)
Player = ‘X’</td>
<td><b>Output:</b>
State:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td><b>0</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>1</b></td><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2</b></td><td></td><td></td><td><b>X</b></td><td></td></tr>
<tr><td><b>3</b></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to see if a marker can be placed at random position in the middle of the board and not just on the edges of the board

<b>Function Name:</b>
testPlaceMarker_position_in_middle</td>
</tr>
</table>

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>
Marker = (3,3)
Player = ‘X’</td>
<td><b>Output:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>2</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
<tr><td>3</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table></td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to see if a marker can be placed at every position on the board

<b>Function Name:</b>
testPlaceMarker_full_board</td>
</tr>
</table>

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>K</td><td>K</td><td>K</td><td>K</td></tr>
<tr><td>1</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>B</td><td>B</td><td>B</td><td>B</td></tr>
<tr><td>3</td><td>E</td><td>E</td><td>E</td><td></td></tr>
</table>
Marker = (3,3)
Player = ‘E’</td>
<td><b>Output:</b>

State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>K</td><td>K</td><td>K</td><td>K</td></tr>
<tr><td>1</td><td>O</td><td>O</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>B</td><td>B</td><td>B</td><td>B</td></tr>
<tr><td>3</td><td>E</td><td>E</td><td>E</td><td>E</td></tr>
</table></td>
<td><b>Reason:</b>
This test case is unique and distinct because it tests to see if its possible to fill the board with different character markers.
<b>Function Name:</b>
testPlaceMarker_full_of_different_characters</td>
</tr>
</table>

Deployment:

N/A for this assignment