Sean Danielson
10/17/19

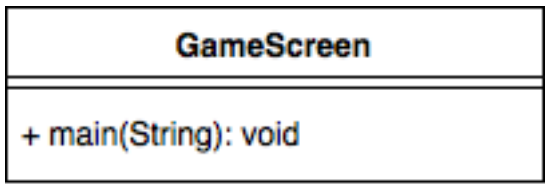
TicTacToe Project Report

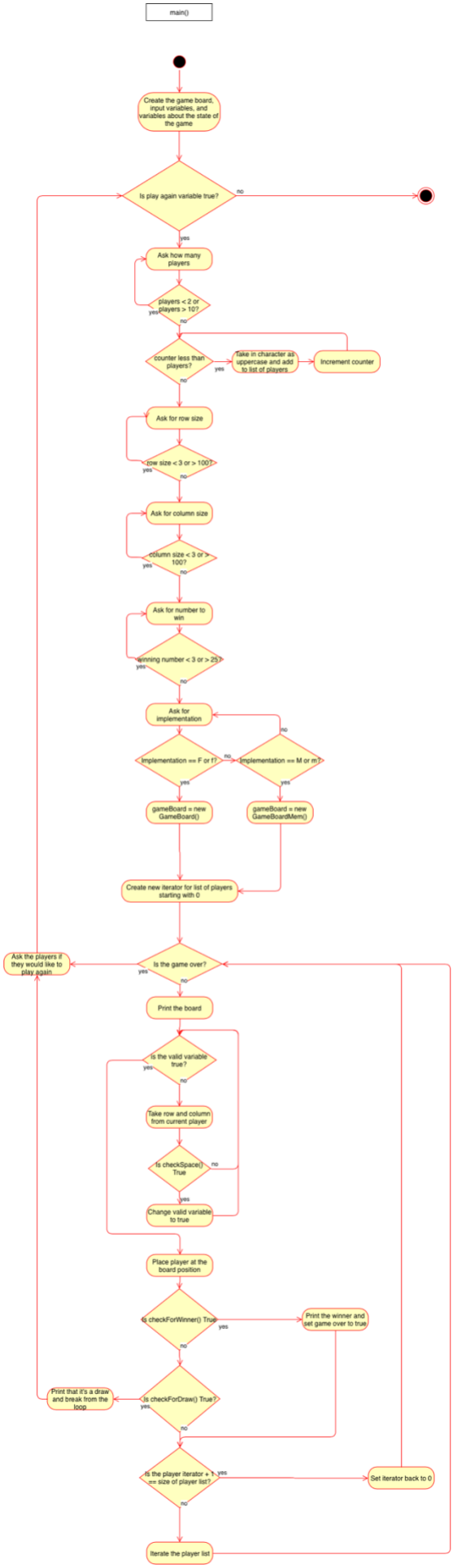
Requirements Analysis

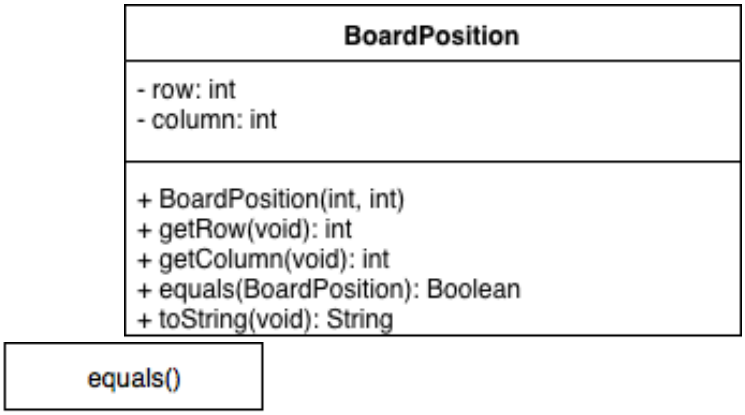
- **Functional Requirements**
  - **As a player, I can easily tell what player's turn it is so I can play on my turn.**
  - **As a player, I can see the game board, so I know what is available and what is not.**
  - **As a player, I can specify whether or not I want to play again so that I can play multiple games.**
  - **As a player, I understand how the coordinate system works so I can input my moves correctly.**
  - **As a player, I can pick again if I pick an unavailable space, so I don't lose my turn**
  - **As a player, if I get the defined winning number in a row horizontally, I will win the game, so I can win the game**
  - **As a player, if I get the defined winning number in a row vertically, I will win the game, so I can win the game**
  - **As a player, if I get the defined winning number in a row diagonally, I will win the game, so I can win the game**
  - **As a player, I can end the game in a tie by taking the last space on the board without getting the winning number in a row, so the game can end**
  - **As a player, I can input the bounds of the board, so that each game is different.**
  - **As a player, I can input the number required in a row to win.**
  - **As a player, I can input the number of players in the game so that I can play with multiple players.**
  - **As a player, I can decide what character to represent myself as on the gameboard so I can be different from other players.**
  - **As a player, I can decide whether or not I want a faster implementation or a memory efficient implementation so I can make tradeoffs for my games.**
  - **As a player, I can reenter the rules of the game if I decide to play again, so I can play multiple games with different settings.**
- **Nonfunctional Requirements**
  - **The system checks for winning moves frequently.**
  - **The system checks for wins in all directions.**
  - **The system checks for draws frequently.**
  - **The system does not have a user interface.**
  - **The system is direct in its prompts.**
  - **The system uses the command line for user interaction.**
  - **The system checks for player character conflicts.**
  - **The system is developed in Java.**
  - **The system was created with Intellij IDEA.**
  - **The system runs on Unix.**
  - **The system codes to the interface of IGameBoard for ease of game board creation between memory efficient and speed efficient implementations.**
  - **The system keeps track of whose turn it is.**
  - **0,0 is the top left of the board**
  - **The board size cannot exceed 100x100.**
  - **The board size cannot be lower than 3x3**

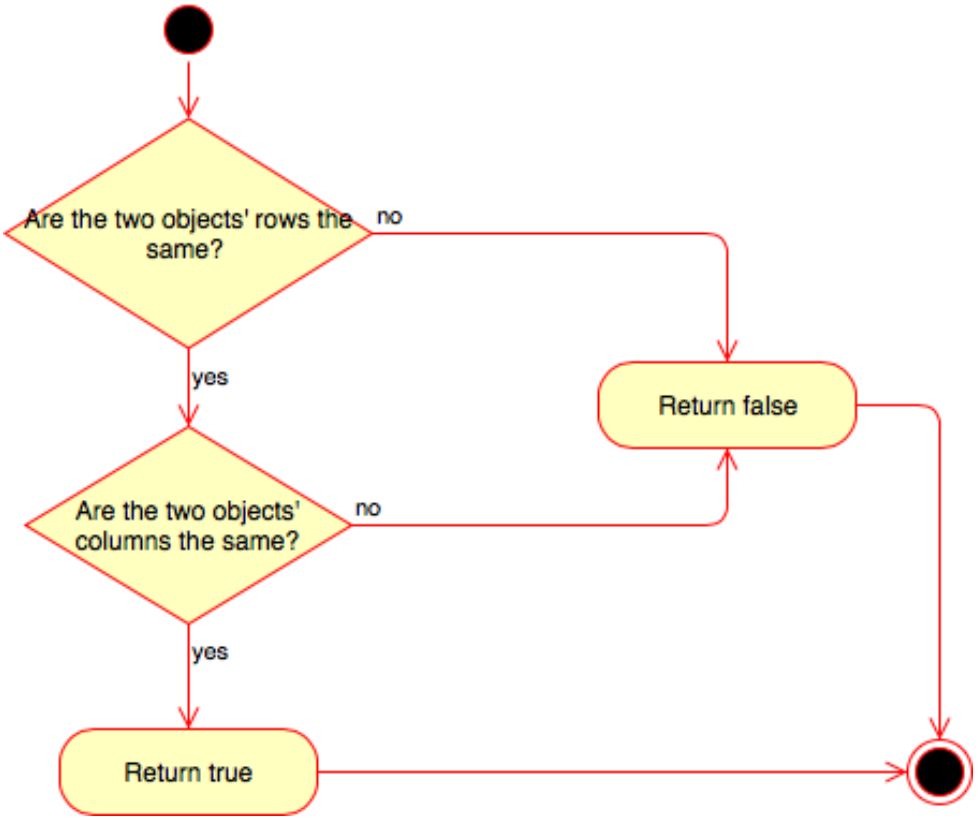
- **The number to win cannot exceed 25**
- **The number to win cannot be lower than 3.**
- **The number of players must be between 2 and 10 inclusive.**

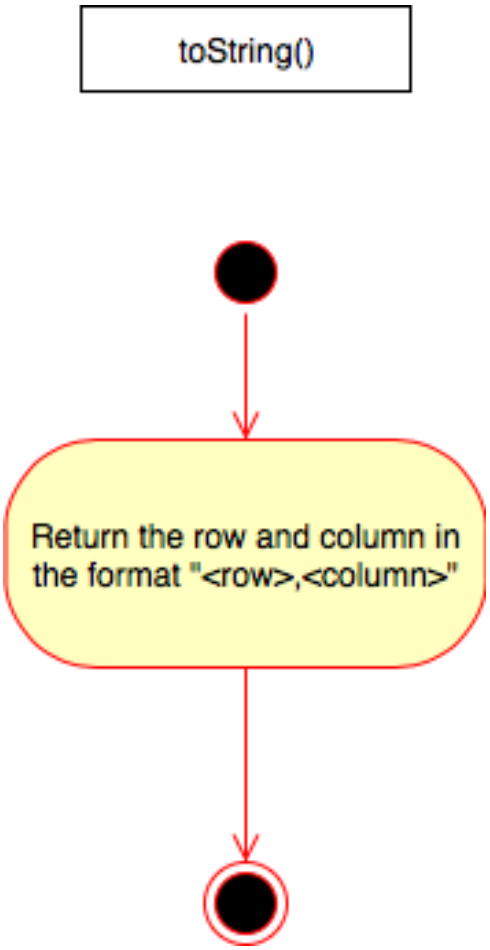
Design

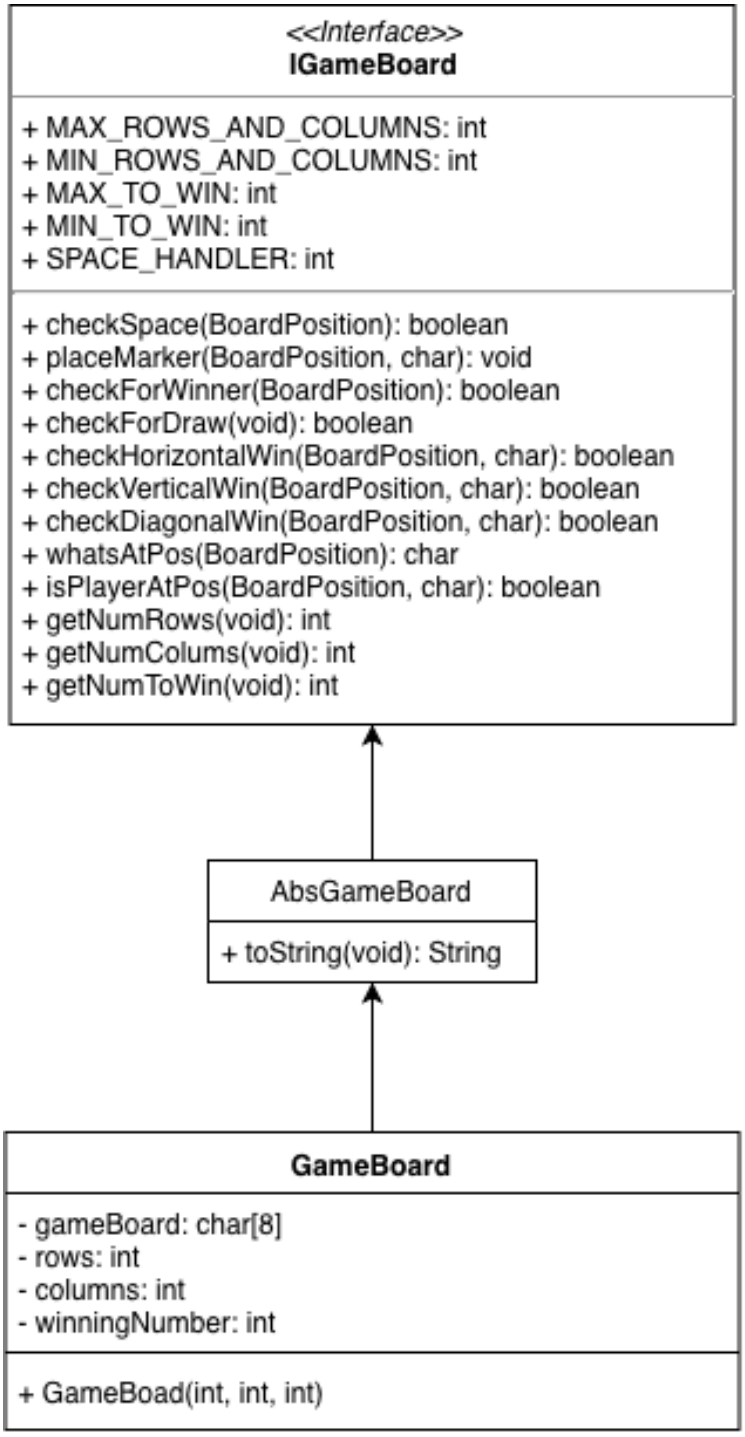


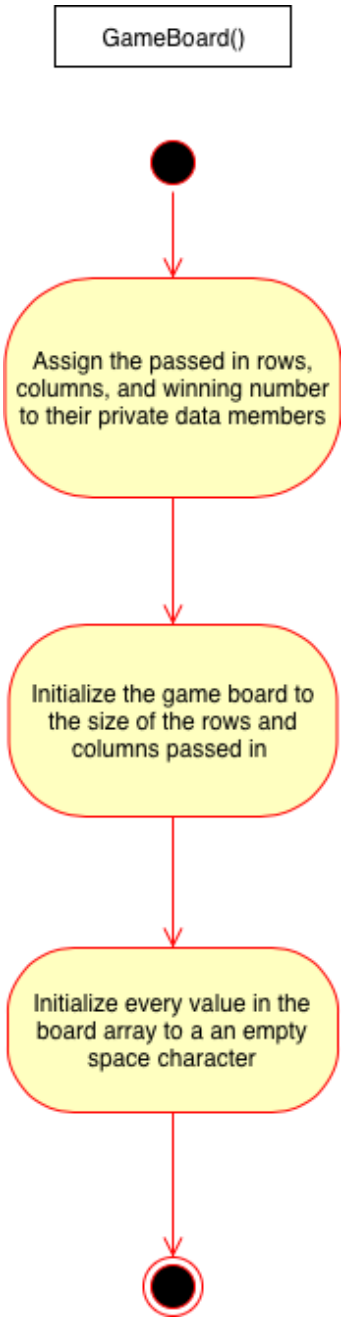

main()
Create the game board, input variables, and variables about the state of the game
Is play again variable true?
no
yes
Ask how many players
players < 2 or players > 10?
yes
no
counter less than players?
yes
Take in character as uppercase and add to list of players
Increment counter
no
Ask for row size
row size < 3 or > 100?
yes
no
Ask for column size
column size < 3 or > 100?
yes
no
Ask for number to win
winning number < 3 or > 25?
yes
no
Ask for implementation
Implementation == F or f?
no
Implementation == M or m?
no
yes
yes
gameBoard = new GameBoard()
gameBoard = new GameBoardMem()
Create new iterator for list of players starting with 0
Is the game over?
yes
Ask the players if they would like to play again
no
Print the board
Is the valid variable true?
yes
no
Take row and column from current player
Is checkSpace() True
no
yes
Change valid variable to true
Place player at the board position
Is checkForWinner() True
yes
Print the winner and set game over to true
no
Is checkForDraw() True?
yes
Print that it's a draw and break from the loop
no
Is the player iterator + 1 == size of player list?
yes
Set iterator back to 0
no
Iterate the player list

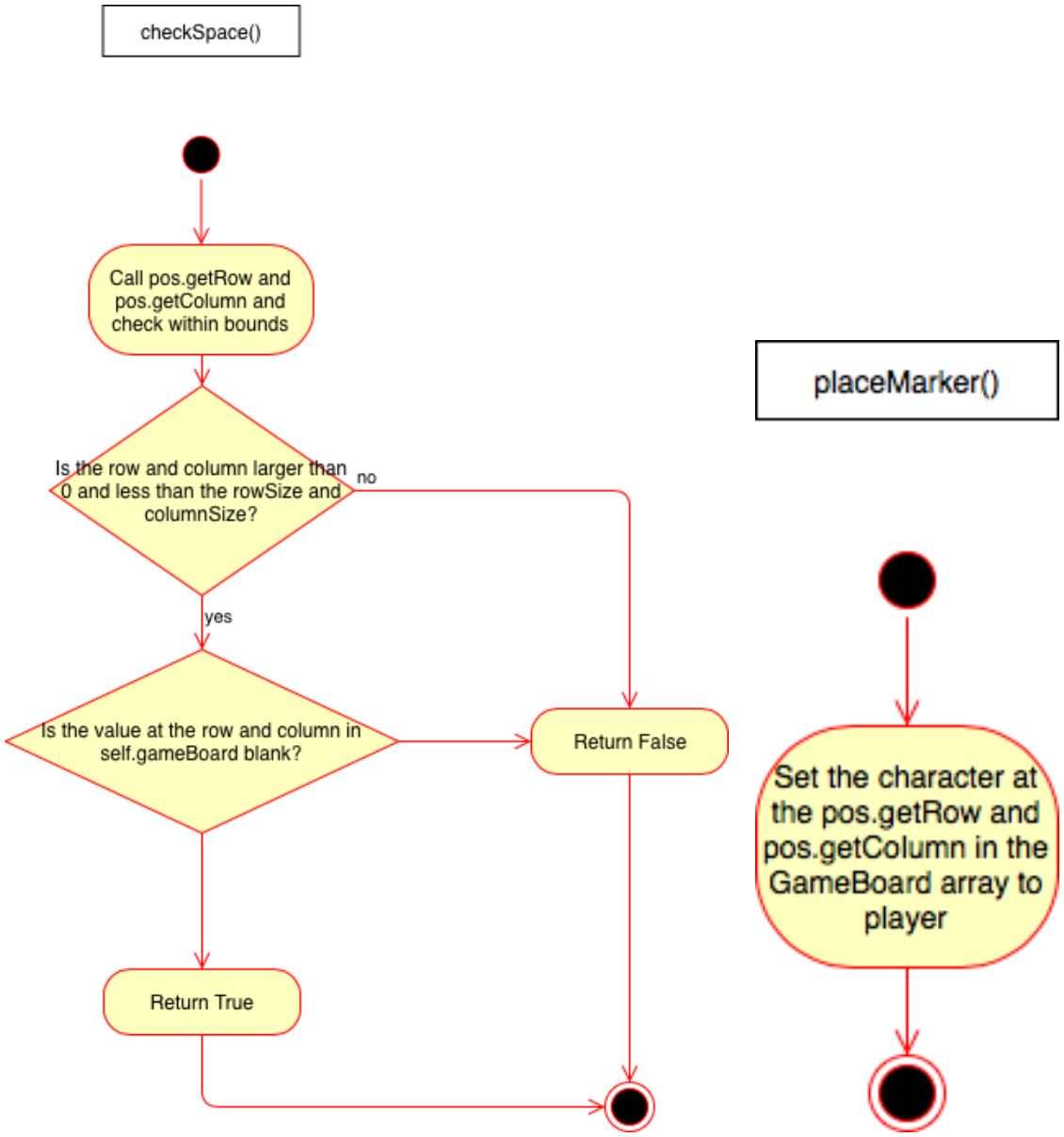

BoardPosition
- row: int
- column: int
+ BoardPosition(int, int)
+ getRow(void): int
+ getColumn(void): int
+ equals(BoardPosition): Boolean
+ toString(void): String
equals()

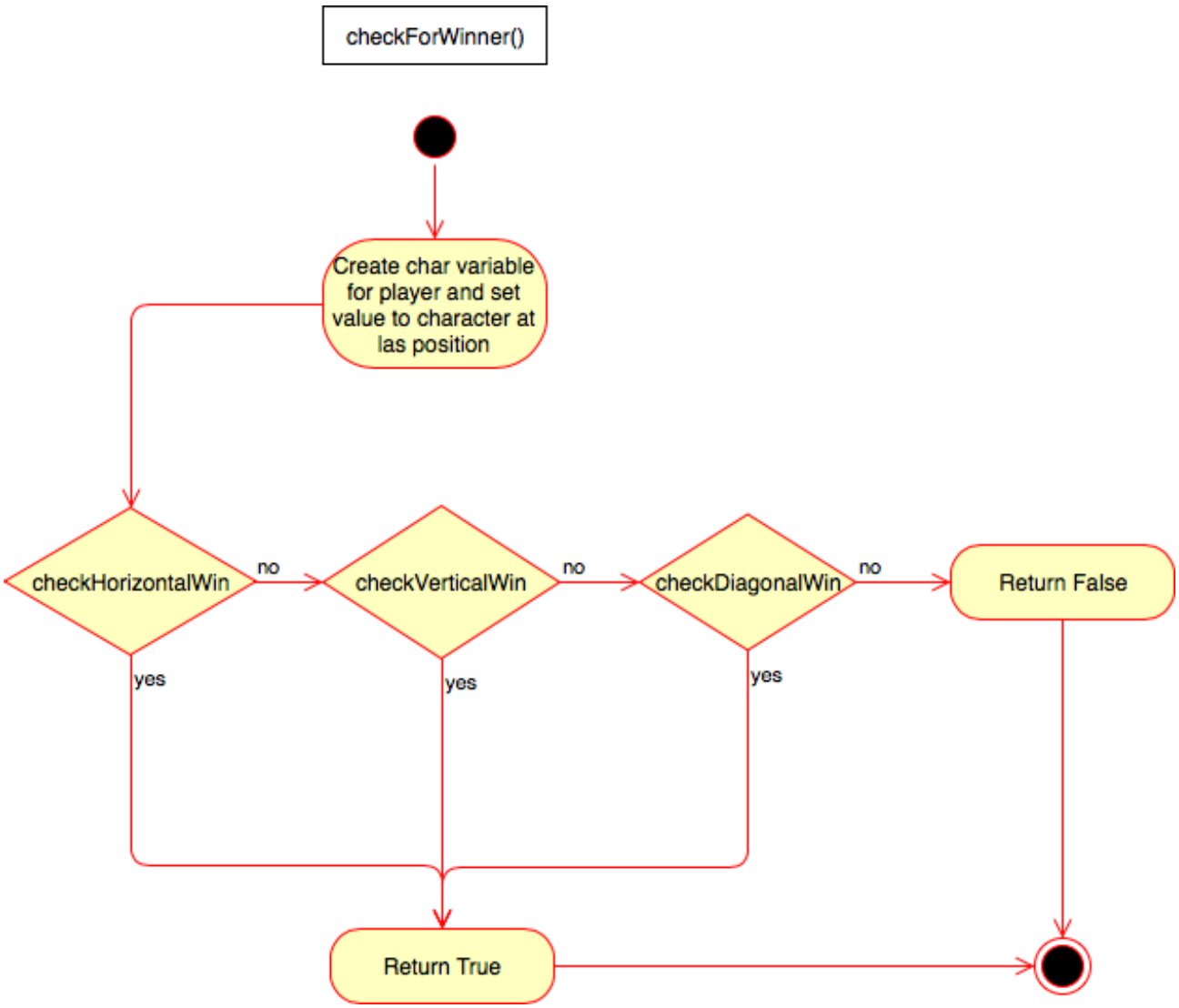

Are the two objects' rows the same?
no
yes
Return false
Are the two objects' columns the same?
no
yes
Return true

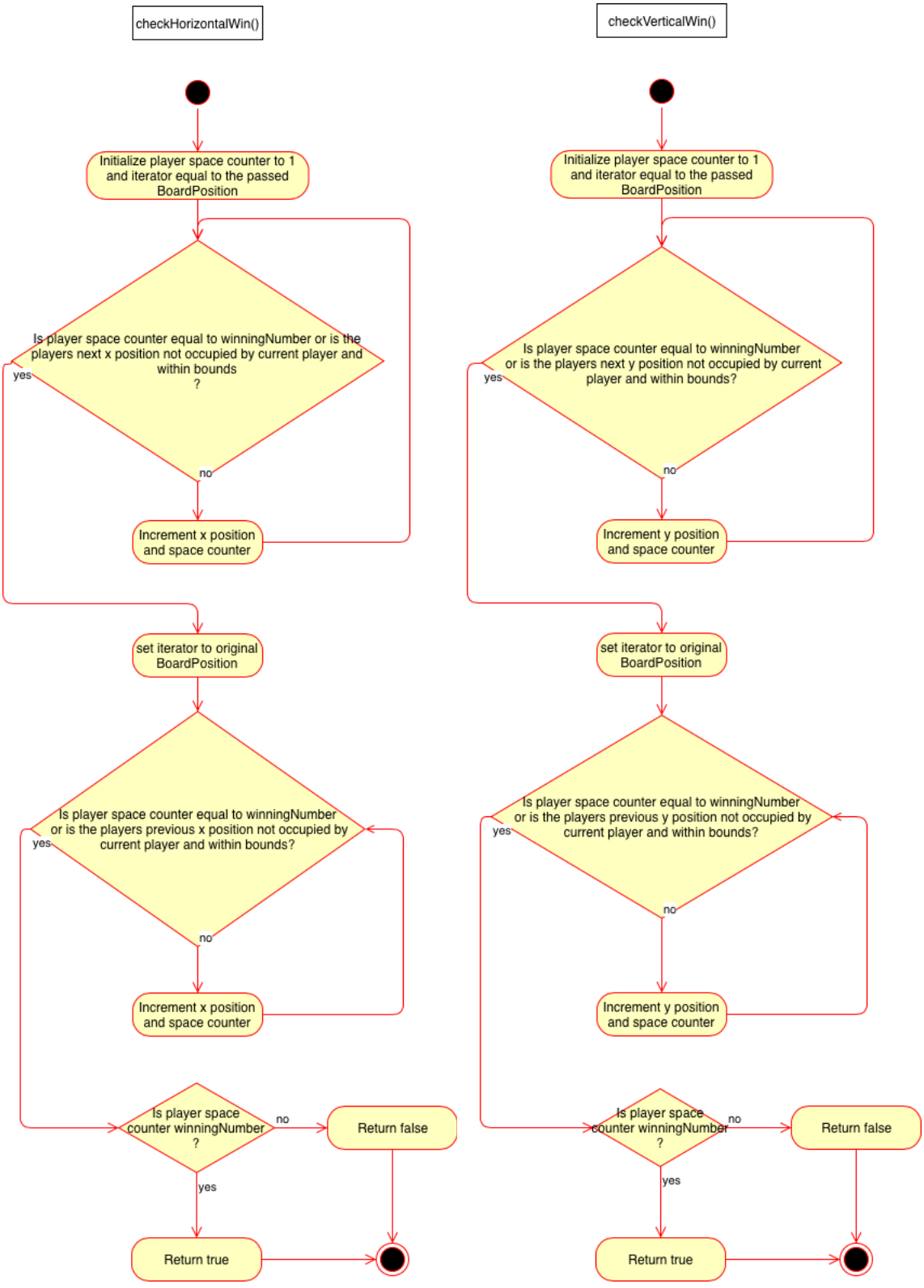

toString()
Return the row and column in the format "<row>,<column>"

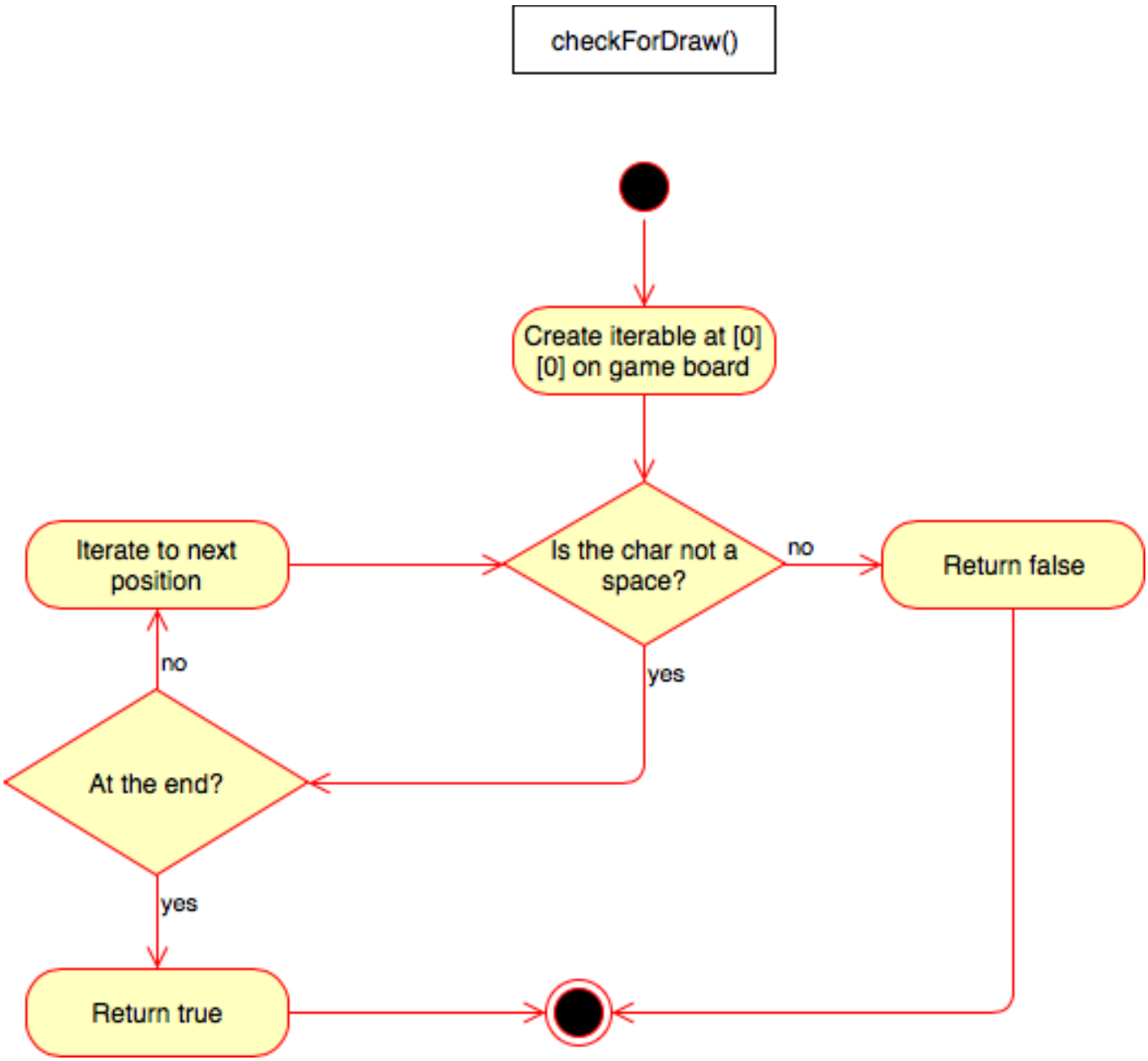

<<Interface>>
IGameBoard
+ MAX_ROWS_AND_COLUMNS: int
+ MIN_ROWS_AND_COLUMNS: int
+ MAX_TO_WIN: int
+ MIN_TO_WIN: int
+ SPACE_HANDLER: int
+ checkSpace(BoardPosition): boolean
+ placeMarker(BoardPosition, char): void
+ checkForWinner(BoardPosition): boolean
+ checkForDraw(void): boolean
+ checkHorizontalWin(BoardPosition, char): boolean
+ checkVerticalWin(BoardPosition, char): boolean
+ checkDiagonalWin(BoardPosition, char): boolean
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ getNumRows(void): int
+ getNumColums(void): int
+ getNumToWin(void): int
AbsGameBoard
+ toString(void): String
GameBoard
- gameBoard: char[8]
- rows: int
- columns: int
- winningNumber: int
+ GameBoad(int, int, int)

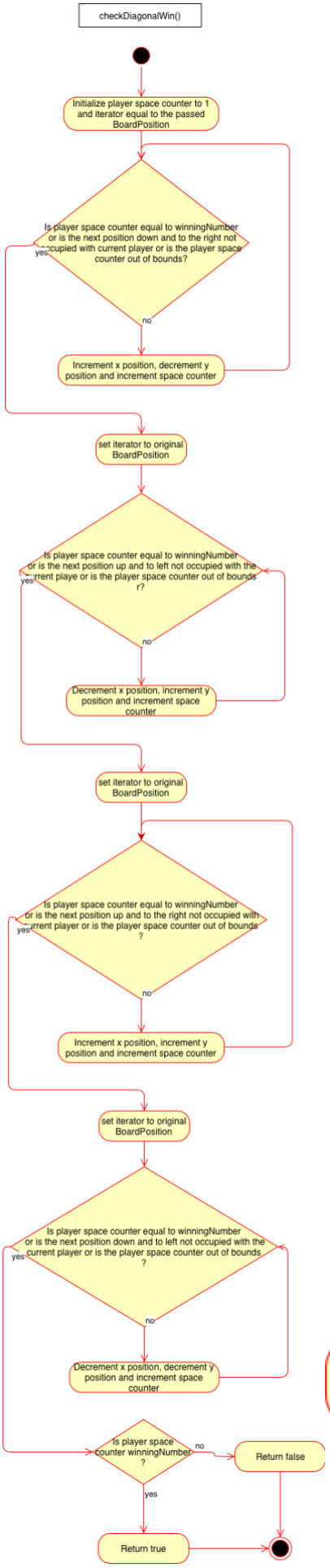

GameBoard()
Assign the passed in rows, columns, and winning number to their private data members
Initialize the game board to the size of the rows and columns passed in
Initialize every value in the board array to a an empty space character

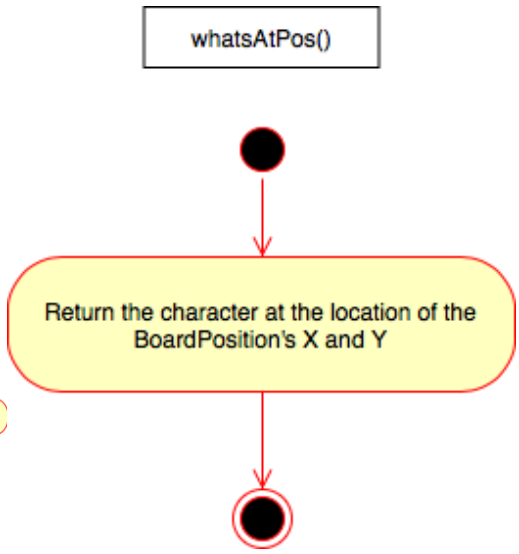

checkSpace()
Call pos.getRow and pos.getColumn and check within bounds
Is the row and column larger than 0 and less than the rowSize and columnSize?
no
yes
Is the value at the row and column in self.gameBoard blank?
Return False
Return True
placeMarker()
Set the character at the pos.getRow and pos.getColumn in the GameBoard array to player

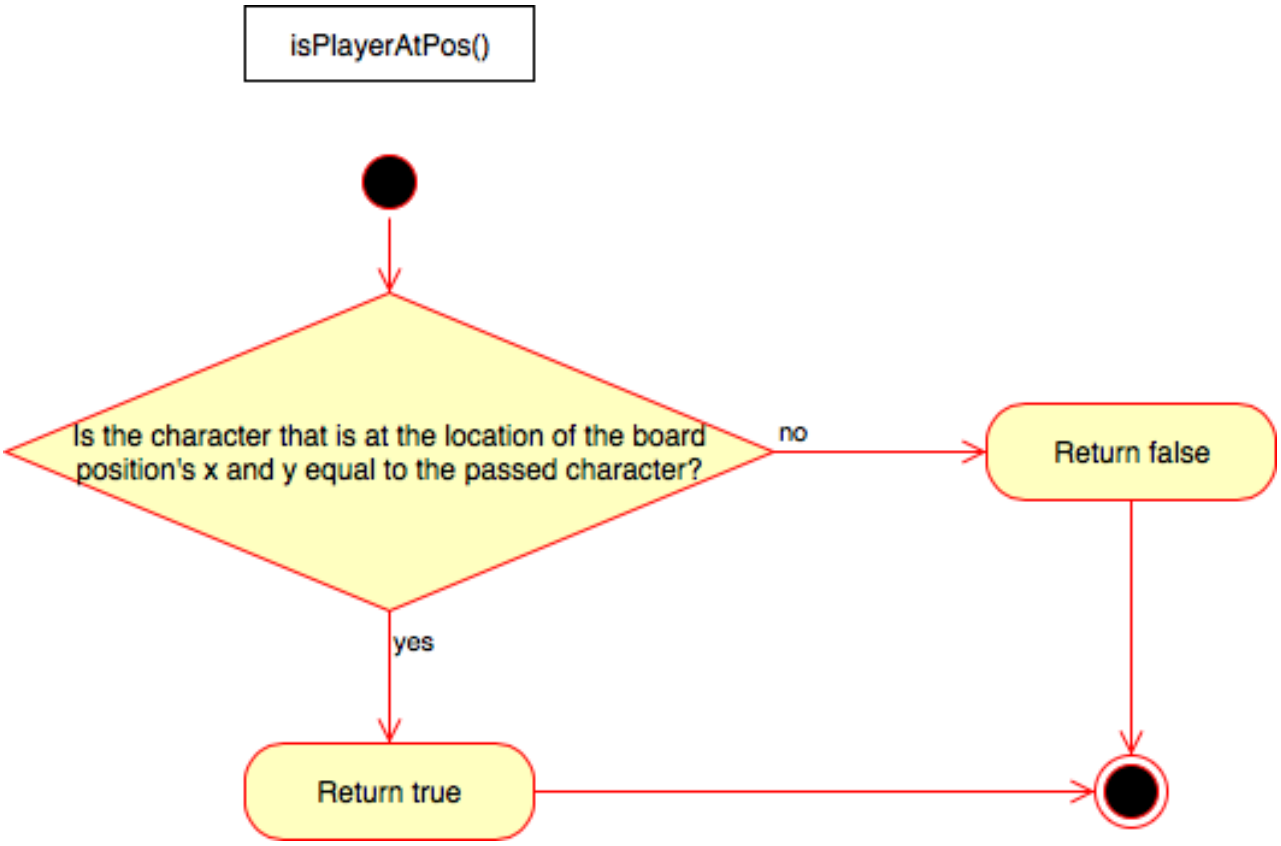

checkForWinner()
Create char variable for player and set value to character at las position
checkHorizontalWin
no
checkVerticalWin
no
checkDiagonalWin
no
Return False
yes
yes
yes
Return True

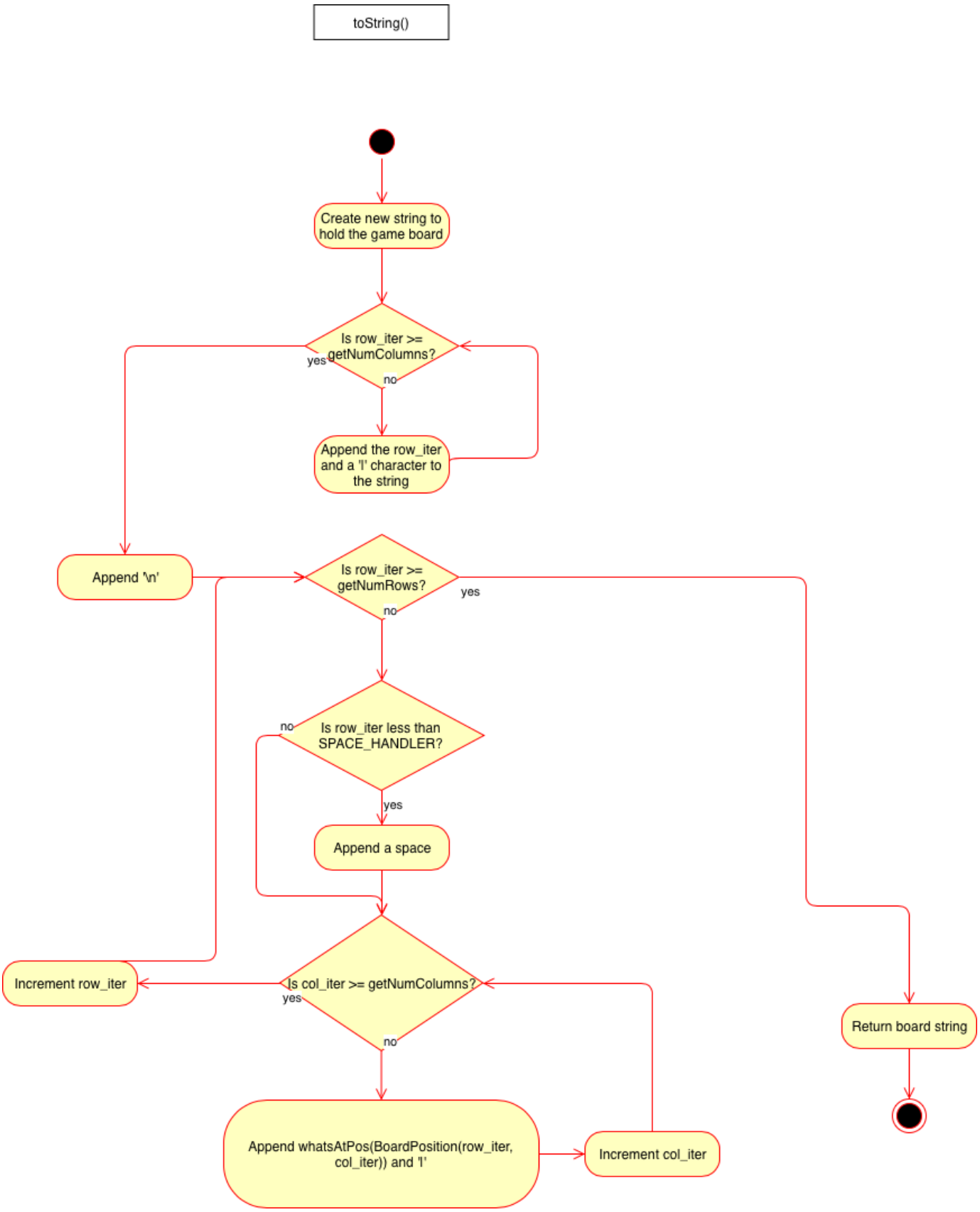

checkHorizontalWin()
Initialize player space counter to 1 and iterator equal to the passed BoardPosition
Is player space counter equal to winningNumber or is the players next x position not occupied by current player and within bounds ?
yes
no
Increment x position and space counter
set iterator to original BoardPosition
Is player space counter equal to winningNumber or is the players previous x position not occupied by current player and within bounds?
yes
no
Increment x position and space counter
Is player space counter winningNumber ?
no
Return false
yes
Return true
checkVerticalWin()
Initialize player space counter to 1 and iterator equal to the passed BoardPosition
Is player space counter equal to winningNumber or is the players next y position not occupied by current player and within bounds?
yes
no
Increment y position and space counter
set iterator to original BoardPosition
Is player space counter equal to winningNumber or is the players previous y position not occupied by current player and within bounds?
yes
no
Increment y position and space counter
Is player space counter winningNumber ?
no
Return false
yes
Return true

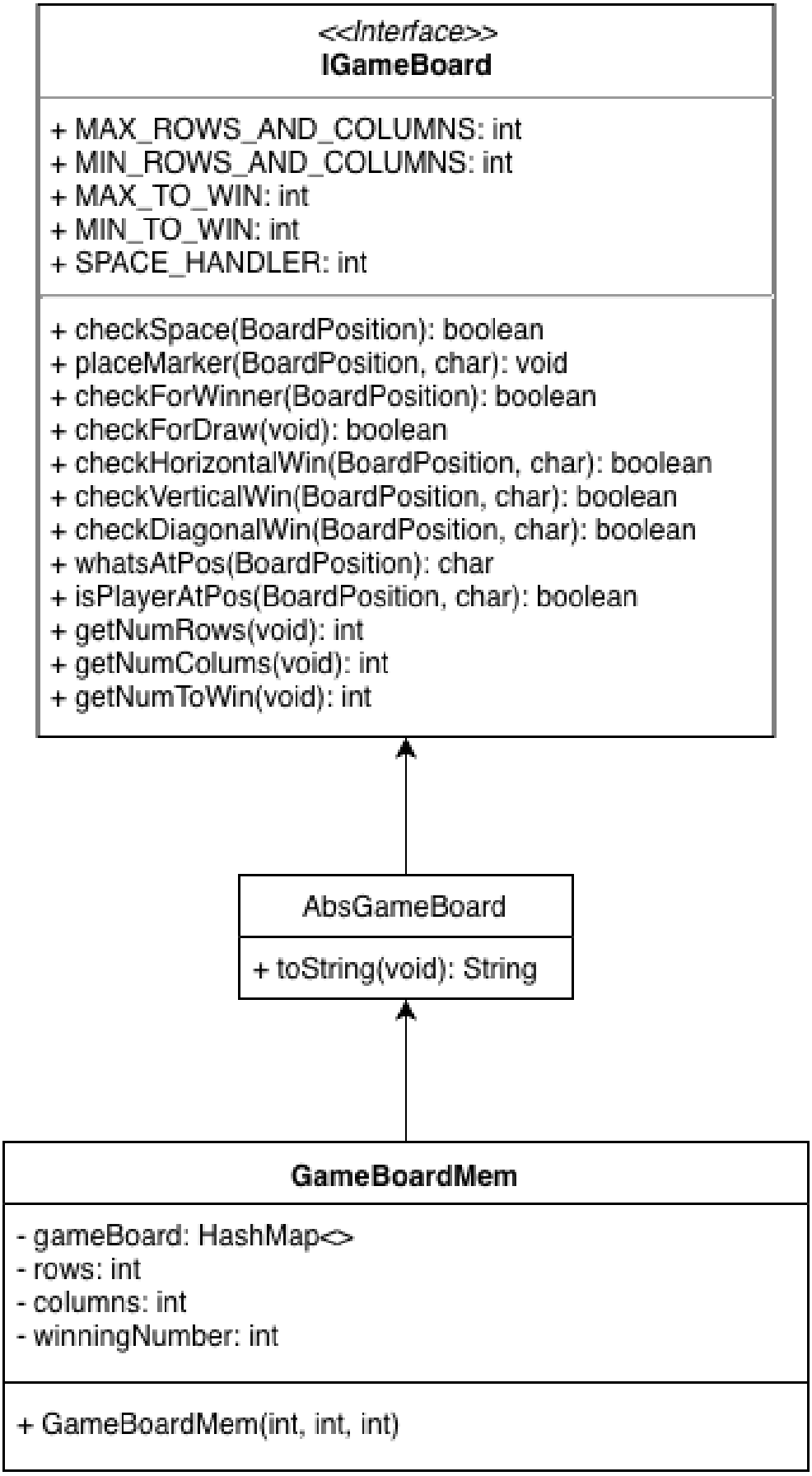

checkForDraw()
Create iterable at [0] [0] on game board
Is the char not a space?
no
Return false
yes
Iterate to next position
no
At the end?
yes
Return true


checkDiagonalWin()
Initialize player space counter to 1 and iterator equal to the passed BoardPosition
Is player space counter equal to winningNumber or is the next position down and to the right not occupied with current player or is the player space counter out of bounds?
yes
no
Increment x position, decrement y position and increment space counter
set iterator to original BoardPosition
Is player space counter equal to winningNumber or is the next position up and to left not occupied with the current playe or is the player space counter out of bounds r?
yes
no
Decrement x position, increment y position and increment space counter
set iterator to original BoardPosition
Is player space counter equal to winningNumber or is the next position up and to the right not occupied with current player or is the player space counter out of bounds ?
yes
no
Increment x position, increment y position and increment space counter
set iterator to original BoardPosition
Is player space counter equal to winningNumber or is the next position down and to left not occupied with the current player or is the player space counter out of bounds ?
yes
no
Decrement x position, decrement y position and increment space counter
Is player space counter winningNumber ?
no
Return false
yes
Return true

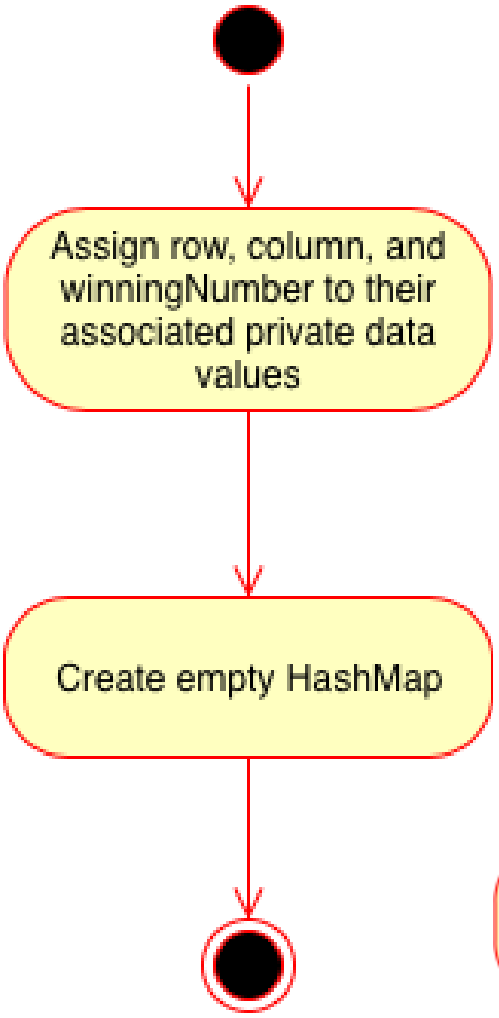

whatsAtPos()
Return the character at the location of the BoardPosition's X and Y


isPlayerAtPos()
Is the character that is at the location of the board position's x and y equal to the passed character?
no
Return false
yes
Return true

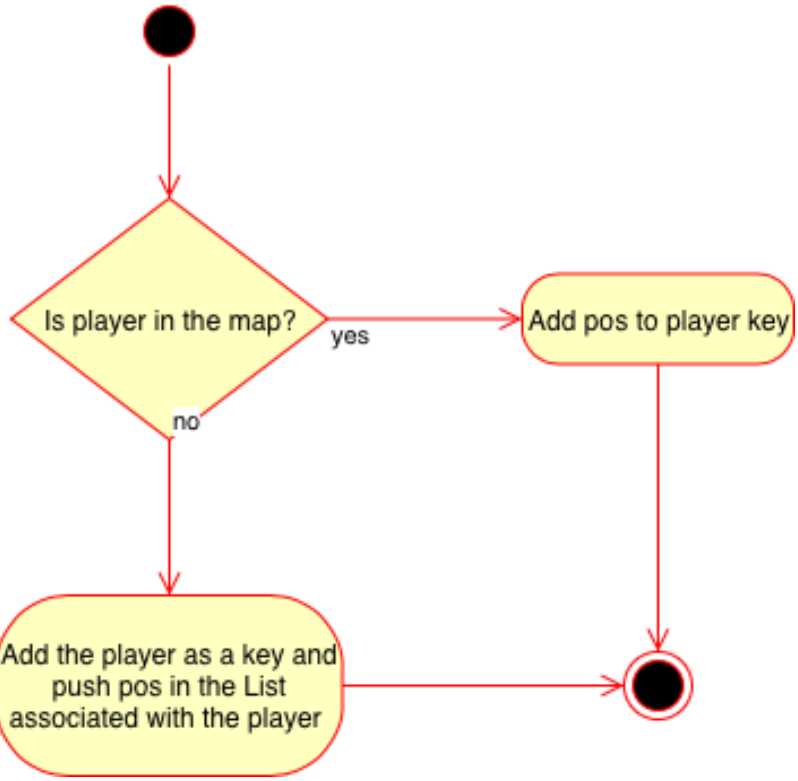

toString()
Create new string to hold the game board
Is row_iter >= getNumColumns?
yes
no
Append the row_iter and a 'l' character to the string
Append '\n'
Is row_iter >= getNumRows?
yes
no
Is row_iter less than SPACE_HANDLER?
no
yes
Append a space
Increment row_iter
Is col_iter >= getNumColumns?
yes
no
Return board string
Append whatsAtPos(BoardPosition(row_iter, col_iter)) and 'l'
Increment col_iter


<<Interface>>
IGameBoard
+ MAX_ROWS_AND_COLUMNS: int
+ MIN_ROWS_AND_COLUMNS: int
+ MAX_TO_WIN: int
+ MIN_TO_WIN: int
+ SPACE_HANDLER: int
+ checkSpace(BoardPosition): boolean
+ placeMarker(BoardPosition, char): void
+ checkForWinner(BoardPosition): boolean
+ checkForDraw(void): boolean
+ checkHorizontalWin(BoardPosition, char): boolean
+ checkVerticalWin(BoardPosition, char): boolean
+ checkDiagonalWin(BoardPosition, char): boolean
+ whatsAtPos(BoardPosition): char
+ isPlayerAtPos(BoardPosition, char): boolean
+ getNumRows(void): int
+ getNumColums(void): int
+ getNumToWin(void): int
AbsGameBoard
+ toString(void): String
GameBoardMem
- gameBoard: HashMap<>
- rows: int
- columns: int
- winningNumber: int
+ GameBoardMem(int, int, int)

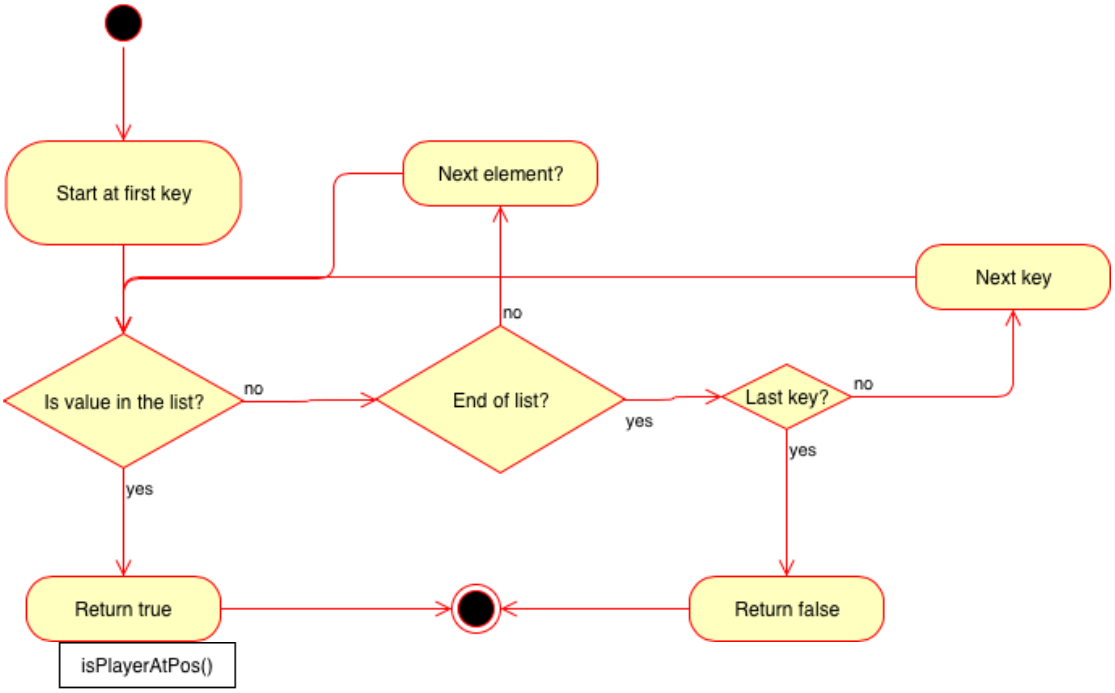
GameBoardMem()



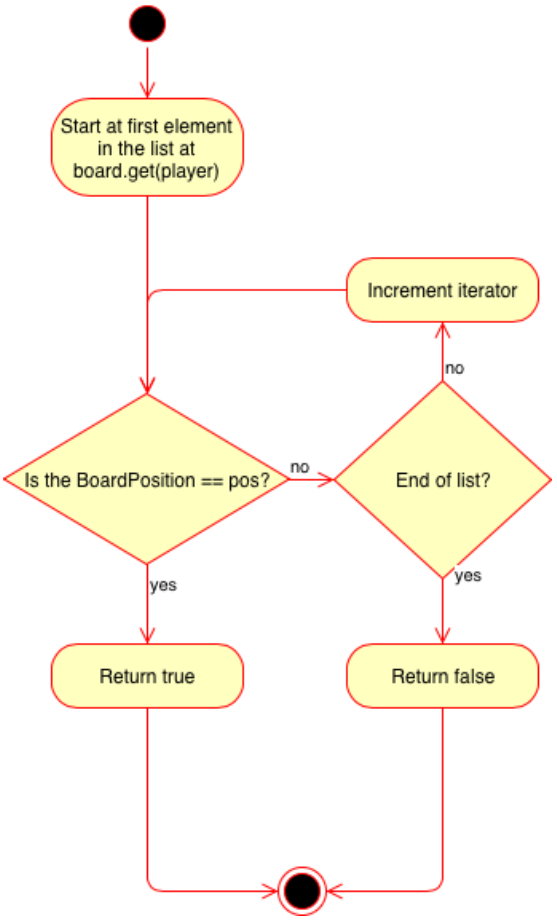
placeMarker()



whatsAtPos()



isPlayerAtPos()



Testing

GameBoard(int rowSize, int columnSize, int winningNumber) / GameBoardMem(int rowSize, int columnSize, int winningNumber)

Input	Output	Reason:
State: uninitialized rowSize = 100 columnSize = 100 winningNumber = 25	State: The board is empty (Too large to show) getNumRows = 100 getNumColumns = 100 getNumToWin = 25	This test case is unique because it tests that the upper bound limit on the board size as well as the number to win Function name: testConstructor_maxVals

Input	Output	Reason:
State: uninitialized rowSize = 3 columnSize = 3 winningNumber = 3	State: getNumRows = 3 getNumColumns = 3 getNumToWin = 3	This test case is unique because it tests that the lower bound limit on the board size as well as the number to win Function name: testConstructor_minVals

	0	1	2
0			
1			
2			

Input	Output	Reason:
State: uninitialized rowSize = 3 columnSize = 100 winningNumber = 3	State: The board is empty (To large to show) getNumRows = 3 getNumColumns = 100 getNumToWin = 3	This test case is unique because it tests that you can have different values as rows and columns as well as testing the two ends of the spectrum Function name: testConstructor_min_Max_min

checkSpace(BoardPosition pos)

Input	Output	Reason:
State: | | 0 | 1 | 2 | | 0 | X | | | | 1 | | | | | 2 | O | | | | 3 | | | | Pos.getRow = 0 Pos.getColumn = 0	checkSpace == false The state of the board is unchanged	This test is unique because the location on the board is occupied Function name: testcheckSpace_occupied

Input	Output	Reason:
State: | | 0 | 1 | 2 | | 0 | X | | | | 1 | | | | | 2 | O | | | | 3 | | | | Pos.getRow = 3 Pos.getColumn = 0	checkSpace == true The state of the board is unchanged	This test is unique because the location on the board is unoccupied and tests the size of the row Function name: testcheckSpace_unoccupied_bottomLeft

Input	Output	Reason:
State: | | 0 | 1 | 2 | | 0 | X | | | | 1 | | | | | 2 | O | | | | 3 | | | | Pos.getRow = 0 Pos.getColumn = 2	checkSpace == true The state of the board is unchanged	This test is unique because the location on the board is unoccupied and tests the size of the column Function name: testcheckSpace_unoccupied_topRight

checkHorizontalWin(BoardPosition lastpos, char player)

Input	Output	Reason:
State: | | 0 | 1 | 2 | | 0 | X | | | | 1 | | | | | 2 | O | O | O | | 3 | | | | lastpos.getRow = 2 lastpos.getColumn = 1 player = O	checkHorizontalWin == true The state of the board is unchanged	This test is unique because the last item placed is in the middle of consecutive characters of the same team Function name: testcheckHorizontalWin_mid_O_True

Input	Output	Reason:
State: | | 0 | 1 | 2 | 3 | | 0 | X | O | O | O | | 1 | | | | | | 2 | O | | | X | | 3 | | | | | lastpos.getRow = 0 lastpos.getColumn = 1 player = O	checkHorizontalWin == true The state of the board is unchanged	This test is unique because the last item placed is on the left side of consecutive characters of the same team AND there are more characters to the left that are not the same character Function name: testcheckHorizontalWin_Left_O_True

Input	Output	Reason:
State: | | 0 | 1 | 2 | 3 | | 0 | X | O | | | | 1 | O | O | X | | | 2 | O | | | X | | 3 | | | | | lastpos.getRow = 1 lastpos.getColumn = 2 player = X	checkHorizontalWin == false The state of the board is unchanged	This test is unique because the last item placed is not in the middle of consecutive characters of the same team (not a win) AND there are characters to the left of the last placed character that are not the same character Function name: testcheckHorizontalWin_alone_X_false

Input State:	Output	Reason:
<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>O</td><td>O</td><td>O</td></tr><tr><td>1</td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td>O</td><td></td><td></td><td>X</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table> lastpos.getRow = 0 lastpos.getColumn = 3 player = O	checkHorizontalWin == true The state of the board is unchanged	This test is unique because the last item placed is on the right side of consecutive characters of the same team AND is on the column bound of the board Function name: testcheckHorizontalWin_right_O_true

checkVerticalWin(BoardPosition lastpos, char player)

Input State:	Output	Reason:
<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>O</td><td></td><td></td></tr><tr><td>1</td><td></td><td>O</td><td></td><td></td></tr><tr><td>2</td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table> lastpos.getRow = 1 lastpos.getColumn = 1 player = O	checkVerticalWin == true The state of the board is unchanged	This test is unique because it tests that there is a vertical win when the last character was placed between the same character. Function name: testcheckVerticalWin_mid_O_true

Input State:	Output	Reason:
<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>O</td><td></td><td></td></tr><tr><td>1</td><td></td><td>O</td><td></td><td></td></tr><tr><td>2</td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table> lastpos.getRow = 2 lastpos.getColumn = 1 player = O	checkVerticalWin == true The state of the board is unchanged	This test is unique because it tests that there is a vertical win when the last character was placed at the bottom of consecutive characters, being the end of the winning line. Function name: testcheckVerticalWin_bottom_O_true

Input State:	Output	Reason:
<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>O</td><td></td><td></td></tr><tr><td>1</td><td></td><td>O</td><td></td><td></td></tr><tr><td>2</td><td></td><td>X</td><td></td><td>X</td></tr><tr><td>3</td><td></td><td>O</td><td></td><td></td></tr></table> lastpos.getRow = 2 lastpos.getColumn = 1 player = X	checkVerticalWin == false The state of the board is unchanged	This test is unique because the last item placed is not in the middle of consecutive characters of the same team (not a win) AND there are characters above and below the last placed character that are not the same character Function name: testcheckVerticalWin_X_false

Input State:	Output	Reason:
<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>O</td><td></td><td></td></tr><tr><td>1</td><td></td><td>O</td><td></td><td></td></tr><tr><td>2</td><td></td><td>O</td><td></td><td>X</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table> lastpos.getRow = 0 lastpos.getColumn = 1 player = O	checkVerticalWin == true The state of the board is unchanged	This test is unique because it tests that there is a vertical win when the last character was placed at the top of consecutive characters, being the end of the winning line. Function name: testcheckVerticalWin_top_O_true

checkDiagonalWin(BoardPosition lastpos, char player)

Input State:	Output	Reason:
<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></td><td></td><td></td></tr><tr><td>1</td><td></td><td>X</td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td>X</td><td>O</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table> lastpos.getRow = 1 lastpos.getColumn = 1 player = X	checkDiagonalWin == true The state of the board is unchanged	This test is unique because it tests that there is a diagonal win when the last character was placed between the same characters on a top left to bottom right diagonal Function name: testcheckDiagonalWin_ leftdiag _mid_X_true

Input
State:

	0	1	2	3
0				
1		X		
2			X	O
3				X

lastpos.getRow = 1
lastpos.getColumn = 1
player = X

Output
checkDiagonalWin == true

The state of the board is unchanged

Reason:
This test is unique because it tests that there is a diagonal win when the last character was placed at the top left of the winning line, with the line being on only one side of the characters

Function name:
testcheckDiagonalWin_ leftdiag _topLeft_X_true

Input
State:

	0	1	2	3
0	X			
1		X		
2			X	O
3				

lastpos.getRow = 2
lastpos.getColumn = 2
player = X

Output
checkDiagonalWin == true

The state of the board is unchanged

Reason:
This test is unique because it tests that there is a diagonal win when the last character was placed at the bottom right of the winning line, with the line being on only one side of the characters

Function name:
testcheckDiagonalWin_leftdiag_b ottomRight_X_true

Input
State:

	0	1	2	3
0				
1		X		
2			O	O
3				

lastpos.getRow = 1
lastpos.getColumn = 1
player = X

Output
checkDiagonalWin == false

The state of the board is unchanged

Reason:
This test is unique because it tests that there is not a diagonal win when the last character was placed between a different character and a space

Function name:
testcheckDiagonalWin_X_false

Input State: <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>X</td></tr><tr><td>1</td><td></td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td>X</td><td></td><td>O</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table> lastpos.getRow = 1 lastpos.getColumn = 2 player = X	Output checkDiagonalWin == true The state of the board is unchanged	Reason: This test is unique because it tests that there is a diagonal win when the last character was placed between the same characters on a bottom left to top right diagonal. Function name: testcheckDiagonalWin_rightdiag_mid_X_true

Input State: <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>X</td><td></td></tr><tr><td>2</td><td></td><td>X</td><td></td><td>O</td></tr><tr><td>3</td><td>X</td><td></td><td></td><td></td></tr></table> lastpos.getRow = 1 lastpos.getColumn = 2 player = X	Output checkDiagonalWin == true The state of the board is unchanged	Reason: This test is unique because it tests that there is a diagonal win when the last character was placed at the top right of the winning line, with the line being on only one side of the characters Function name: testcheckDiagonalWin_rightdiag_topRight_X_true

Input State: <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>X</td></tr><tr><td>1</td><td></td><td></td><td>X</td><td></td></tr><tr><td>2</td><td></td><td>X</td><td></td><td>O</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table> lastpos.getRow = 2 lastpos.getColumn = 1 player = X	Output checkDiagonalWin == true The state of the board is unchanged	Reason: This test is unique because it tests that there is a diagonal win when the last character was placed at the bottom left of the winning line, with the line being on only one side of the characters Function name: testcheckDiagonalWin_rightdiag_bottomLeft_X_true

checkForDraw()

Input State:	Output	Reason:
<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>	checkForDraw == false The state of the board is unchanged	This test is unique because it tests for a draw on a completely empty board Function name: testcheckForDraw_empty_false

Input State:	Output	Reason:
<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>O</td><td>X</td><td>O</td></tr><tr><td>1</td><td>O</td><td>P</td><td>Z</td><td>X</td></tr><tr><td>2</td><td>X</td><td>P</td><td>Z</td><td>O</td></tr><tr><td>3</td><td>O</td><td>X</td><td>O</td><td>X</td></tr></table>	checkForDraw == true The state of the board is unchanged	This test is unique because it tests for a draw on a completely full board where there is an actual draw Function name: testcheckForDraw_full_true

Input State:	Output	Reason:
<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>O</td><td>X</td><td></td></tr><tr><td>1</td><td>O</td><td>P</td><td>Z</td><td></td></tr><tr><td>2</td><td>X</td><td>P</td><td>Z</td><td></td></tr><tr><td>3</td><td>O</td><td>X</td><td>O</td><td></td></tr></table>	checkForDraw == false The state of the board is unchanged	This test is unique because it tests for a draw on a board that is full except for the max column, in order to test that the column size is being calculated correctly Function name: testcheckForDraw_rightEmpty_false

Input State:	Output	Reason:
<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>O</td><td>X</td><td>O</td></tr><tr><td>1</td><td>O</td><td>P</td><td>Z</td><td>X</td></tr><tr><td>2</td><td>X</td><td>P</td><td>Z</td><td>O</td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>	checkForDraw == false The state of the board is unchanged	This test is unique because it tests for a draw on a board that is full except for the max row, in order to test that the row size is being calculated correctly Function name: testcheckForDraw_bottomEmpty_false

whatsAtPos(BoardPosition pos)

Input	Output	Reason:
State: pos.getRow = 3 pos.getColumn = 0	whatsAtPos == O The state of the board is unchanged	This tests that whatsAtPos processes the max number of rows correctly Function name: testwhatsAtPos_bottomLeft

	0	1	2	3
0	X	O	X	O
1	O	P	Z	X
2	X	P	Z	O
3	O			

Input	Output	Reason:
State: pos.getRow = 0 pos.getColumn = 3	whatsAtPos == O The state of the board is unchanged	This tests that whatsAtPos processes the max number of columns correctly Function name: testwhatsAtPos_topRight

	0	1	2	3
0	X	O	X	O
1	O	P	Z	X
2	X	P	Z	O
3				

Input	Output	Reason:
State: pos.getRow = 2 pos.getColumn = 2	whatsAtPos == Z The state of the board is unchanged	This tests that whatsAtPos returns what's at a basic location in the middle of the board Function name: testwhatsAtPos_basic

	0	1	2	3
0	X	O	X	O
1	O		Z	X
2	X	P	Z	O
3	O			

Input State:	Output	Reason:
<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>O</td><td>X</td><td>O</td></tr><tr><td>1</td><td>O</td><td></td><td>Z</td><td>X</td></tr><tr><td>2</td><td>X</td><td>P</td><td>Z</td><td>O</td></tr><tr><td>3</td><td>O</td><td></td><td></td><td></td></tr></table> pos.getRow = 1 pos.getColumn = 1	whatsAtPos == ‘ ‘ The state of the board is unchanged	This tests that whatsAtPos returns a space if the space that’s being checked is empty. This is important especially for GameBoardMem because spaces are not saved in memory Function name: testwhatsAtPos_space

Input State:	Output	Reason:
<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>O</td><td></td><td>Z</td><td>X</td></tr><tr><td>2</td><td></td><td>P</td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table> pos.getRow = 2 pos.getColumn = 1	whatsAtPos == P The state of the board is unchanged	This tests that whatsAtPos returns the correct character when the character being checked was the last one added to the board. This is unique especially for GameBoardMem when all the characters are being stored in a map. Function name: testwhatsAtPos_lastcharPlaced

isPlayerAtPos(BoardPosition pos, char player)

Input State:	Output	Reason:
<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>O</td><td>X</td><td>O</td></tr><tr><td>1</td><td>O</td><td></td><td>Z</td><td>X</td></tr><tr><td>2</td><td>X</td><td>P</td><td>Z</td><td>O</td></tr><tr><td>3</td><td>O</td><td></td><td></td><td></td></tr></table> pos.getRow = 0 pos.getColumn = 1 player = O	isPlayerAtPos == true The state of the board is unchanged	This tests that isPlayerAtPos correctly returns that the specified character is at pos. Function name: testisPlayerAtPos_basic_true

Input State: pos.getRow = 1 pos.getColumn = 1 player = O	Output isPlayerAtPos == false The state of the board is unchanged	Reason: This tests that isPlayerAtPos correctly returns that the specified character is not at pos when pos is empty. Function name: testisPlayerAtPos_empty_false

	0	1	2	3
0	X	O	X	O
1	O		Z	X
2	X	P	Z	O
3	O			

Input State: pos.getRow = 1 pos.getColumn = 2 player = O	Output isPlayerAtPos == false The state of the board is unchanged	Reason: This tests that isPlayerAtPos correctly returns that the specified character is not at pos when pos is occupied by another character. Function name: testisPlayerAtPos_occupied_false

	0	1	2	3
0	X	O	X	O
1	O		Z	X
2	X	P	Z	O
3	O			

Input State: pos.getRow = 3 pos.getColumn = 0 player = O	Output isPlayerAtPos == true The state of the board is unchanged	Reason: This tests that isPlayerAtPos correctly returns that the specified character is at pos when pos is at the max row Function name: testisPlayerAtPos_bottomLeft_true

	0	1	2	3
0	X	O	X	O
1	O		Z	X
2	X	P	Z	O
3	O			

<table>
<tr>
<td>Input
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>O</td><td>X</td><td>O</td></tr><tr><td>1</td><td>O</td><td></td><td>Z</td><td>X</td></tr><tr><td>2</td><td>X</td><td>P</td><td>Z</td><td>O</td></tr><tr><td>3</td><td>O</td><td></td><td></td><td></td></tr></table>
pos.getRow = 0
pos.getColumn = 3
player = O</td>
<td>Output
isPlayerAtPos == true

The state of the board is unchanged</td>
<td>Reason:
This tests that isPlayerAtPos correctly returns that the specified character is at pos when pos is at the max column

Function name:
testisPlayerAtPos_topRight_true</td>
</tr>
</table>

placeMarker(BoardPosition marker, char player)

Input	Output	Reason:
State: | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | | | | marker.getRow = 0 marker.getColumn = 0 player = O	State: | | 0 | 1 | 2 | | 0 | O | | | | 1 | | | | | 2 | | | |	This tests that, in general, place marker puts the character player at the location on the board, marker Function name: testplaceMarker_basic

Input	Output	Reason:
State: | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | | | | marker.getRow = 2 marker.getColumn = 0 player = O	State: | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | O | | |	This tests that placeMarker will put the character player on the max row Function name: testplaceMarker_bottomLeft

Input	Output	Reason:
State: | | 0 | 1 | 2 | | 0 | | | | | 1 | | | | | 2 | | | | marker.getRow = 0 marker.getColumn = 2 player = O	State: | | 0 | 1 | 2 | | 0 | | | O | | 1 | | | | | 2 | | | |	This tests that placeMarker will put the character player on the max column Function name: testplaceMarker_topRight

Input	Output	Reason:
State: | | 0 | 1 | 2 | | 0 | Z | | O | | 1 | | | | | 2 | | | | marker.getRow = 1 marker.getColumn = 1 player = S	State: | | 0 | 1 | 2 | | 0 | Z | | O | | 1 | | S | | | 2 | | | |	This test is unique because it adds a brand new character to the board Function name: testplaceMarker_newChar

Input
State:

	0	1	2	3
0	R	O	X	S
1	V	T	Z	L
2		K	U	W
3	M	N	Q	G

marker.getRow = 2
marker.getColumn = 0
player = Y

Output
State:

	0	1	2	3
0	R	O	X	S
1	V	T	Z	L
2	Y	K	U	W
3	M	N	Q	G

Reason:
This test is unique because it tests that the board can hold different characters in every space

Function name:
testplaceMarker_diffChar_full

Deployment

Included in the project is a makefile with the following targets:

- **default: compiles all the code. Runs with the *make* command**
- **run: runs the code. Runs with the command *make run***
- **test: compiles the test cases with the command *make test***
- **testGB: runs the gameboard test cases with the command *make testGB***
- **testGBmem: runs the gameboardmem test cases with the command *make testGBmem***
- **clean: removes all compiled files from the package. Runs with the command *make clean***

In order to run Extended Tic Tac Toe, do the following:

1. **Make your way to the directory that contains the makefile and the package cpsc2150.extendedTicTacToe in the command line**
2. **When you are in the directory with the makefile, type the command *make***
3. **Type *make run***
4. **When the game is done, type *make clean***