CPSC 2150 Project 4 Report

Sean Hermes

Requirements Analysis

Functional Requirements:

1. As a player I need to place a X down so that I can further the game.
2. As a player I need to place a O down so that I can further the game.
3. As a player I can start a new game so I could play again.
4. As a player I can not start a new game so that I could be done playing.
5. As a player I can view the board so that I can plan out future moves.
6. As a player I can input the number of rows so I can make a bigger or smaller board
7. As a player I can input the number of columns so I can make a bigger or smaller board
8. As a player I can input the number of players that will be playing so I can play against other people
9. As a player I can input my players character, so I know which piece is mine
10. As a player I can choose a faster implementation of the game so I can play quicker
11. As a player I can choose a more memory efficient implementation of the game so the game takes less memory
12. As a player I can place a piece to the right of my piece so that I can win.
13. As a player I can place a piece to the left of my opponent's piece so that I can get an advantage.
14. As a player I can place a piece to the right of my opponent's piece so that I can get an advantage.
15. As a player I can place a piece above my opponent's piece so that I can get an advantage.
16. As a player I can place a piece below my opponent's piece so that I can get an advantage.
17. As a player I can place a piece diagonally of my opponent's piece so that I can get an advantage.
18. As a player I can place a piece to the left of my piece so that I can win.
19. As a player I can place a piece above my piece so that I can win.
20. As a player I can place a piece below my piece so that I can win.
21. As a player I can place a piece diagonal of my piece so that I can win.

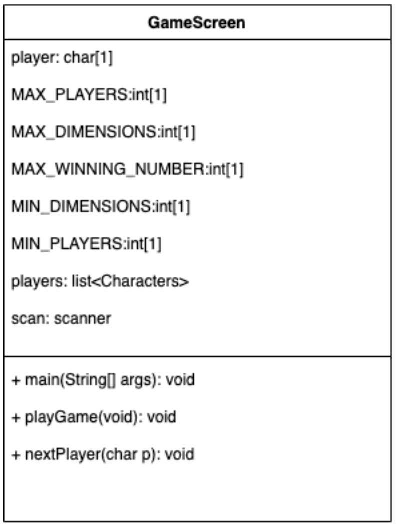
Non-Functional Requirements

1. The system must be written in java
2. The system must be able to run on the SOC computers
3. The system must run without crashing
4. The system must be able to show the board to the player
5. The system must keep a fixed game board that is a 2d char array with 5 rows and 8 columns
6. The system must keep a map of keys which represent the players and values which represent each position a player has chosen.
7. The system must overload the equals() method
8. The System must overload the toString() method

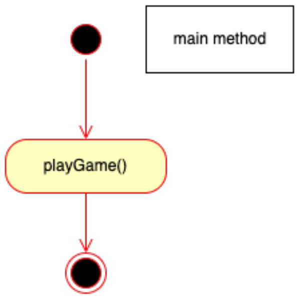
System Design

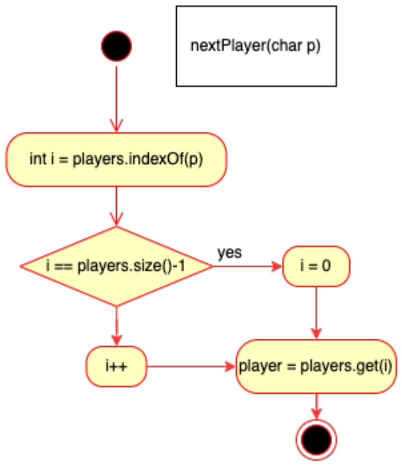
Class 1: GameScreen

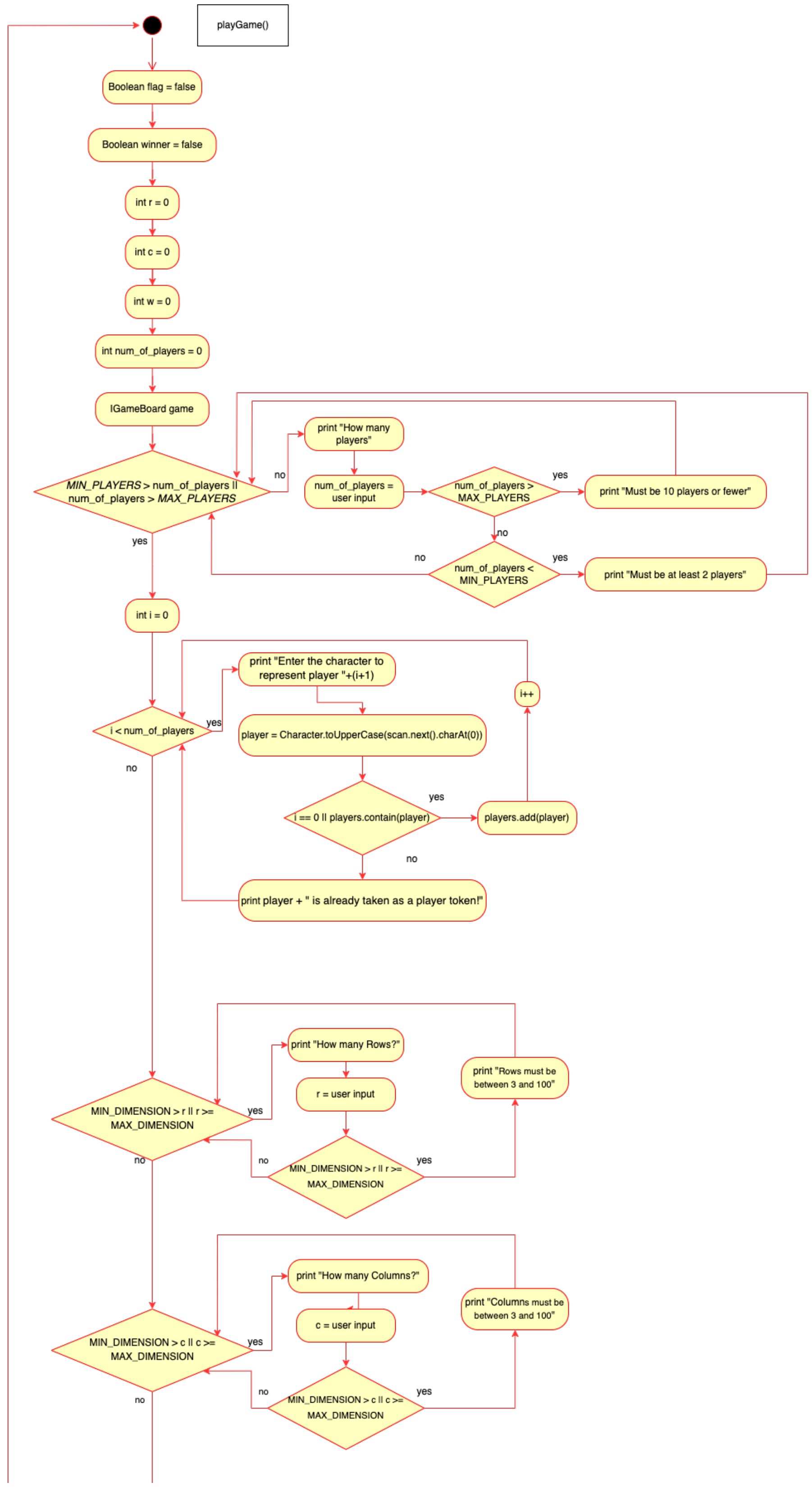
Class diagram

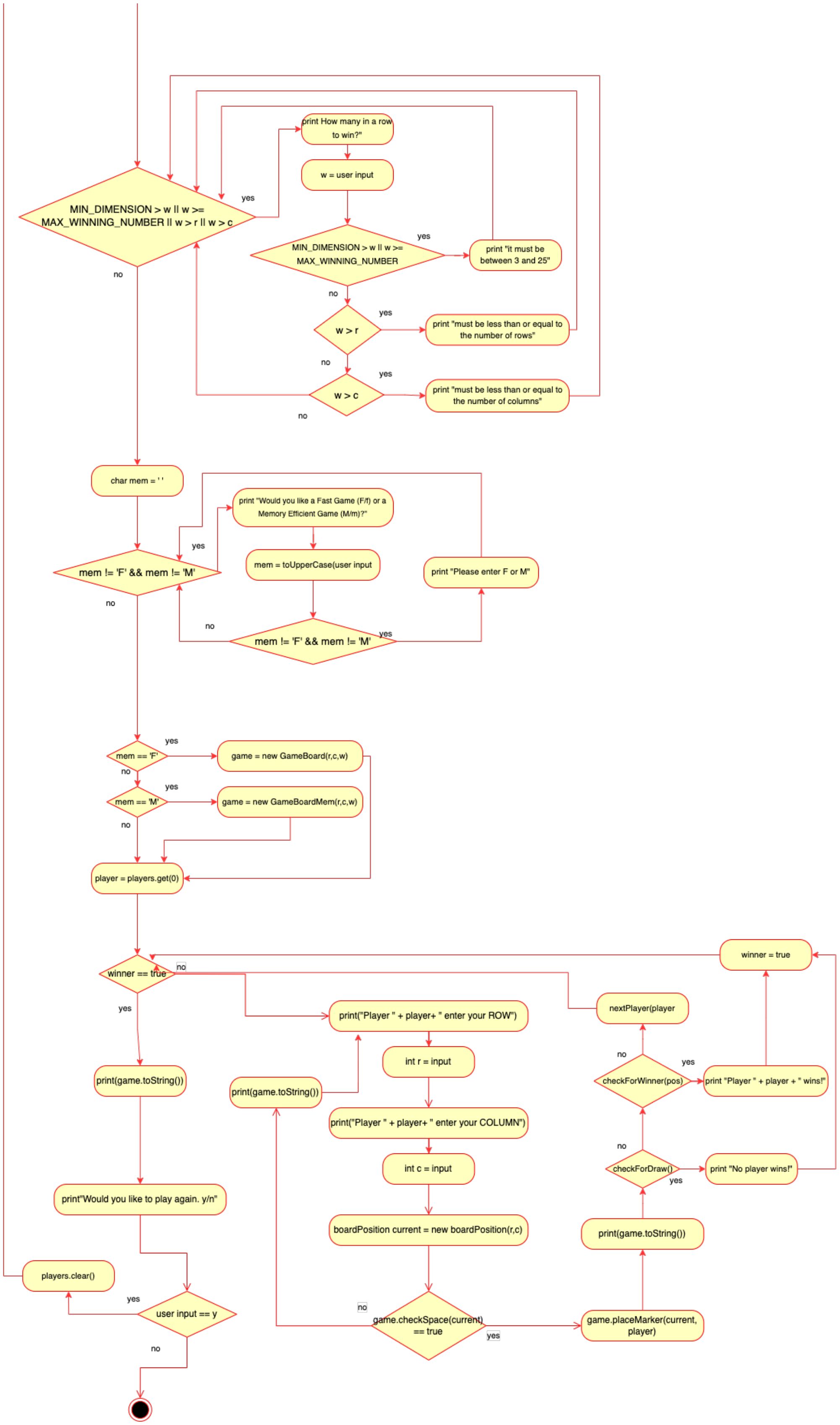


Activity diagrams


main method
playGame()

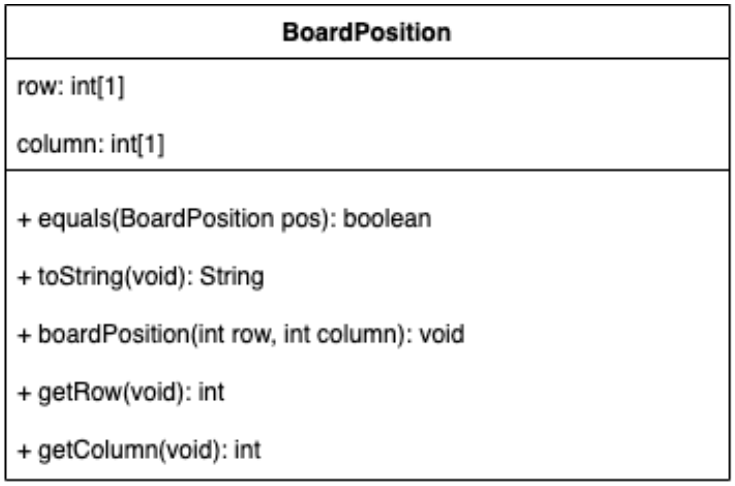

nextPlayer(char p)
int i = players.indexOf(p)
i == players.size()-1
yes
i = 0
i++
player = players.get(i)


playGame()
Boolean flag = false
Boolean winner = false
int r = 0
int c = 0
int w = 0
int num_of_players = 0
IGameBoard game
MIN_PLAYERS > num_of_players || num_of_players > MAX_PLAYERS
no
print "How many players"
num_of_players = user input
num_of_players > MAX_PLAYERS
yes
print "Must be 10 players or fewer"
no
num_of_players < MIN_PLAYERS
no
yes
print "Must be at least 2 players"
yes
int i = 0
print "Enter the character to represent player "+(i+1)
i++
i < num_of_players
yes
player = Character.toUpperCase(scan.next().charAt(0))
no
yes
i == 0 || players.contain(player)
players.add(player)
no
print player + " is already taken as a player token!"
print "How many Rows?"
print "Rows must be between 3 and 100"
r = user input
MIN_DIMENSION > r || r >= MAX_DIMENSION
yes
no
no
MIN_DIMENSION > r || r >= MAX_DIMENSION
yes
print "How many Columns?"
print "Columns must be between 3 and 100"
c = user input
MIN_DIMENSION > c || c >= MAX_DIMENSION
yes
no
no
MIN_DIMENSION > c || c >= MAX_DIMENSION
yes

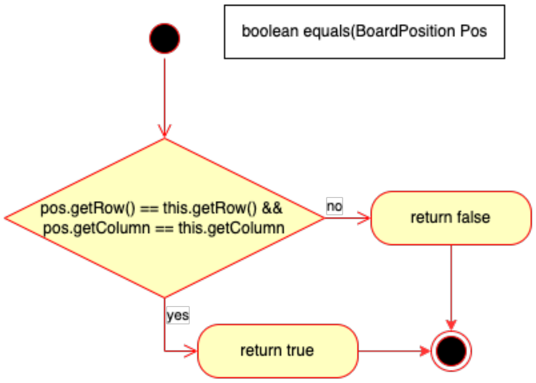

MIN_DIMENSION > w || w >= MAX_WINNING_NUMBER || w > r || w > c
yes
print How many in a row to win?"
w = user input
MIN_DIMENSION > w || w >= MAX_WINNING_NUMBER
yes
print "it must be between 3 and 25"
no
w > r
yes
print "must be less than or equal to the number of rows"
no
w > c
yes
print "must be less than or equal to the number of columns"
no
no
char mem = ' '
print "Would you like a Fast Game (F/f) or a Memory Efficient Game (M/m)?"
mem != 'F' && mem != 'M'
yes
mem = toUpperCase(user input
print "Please enter F or M"
no
no
mem != 'F' && mem != 'M'
yes
mem == 'F'
yes
game = new GameBoard(r,c,w)
no
mem == 'M'
yes
game = new GameBoardMem(r,c,w)
no
player = players.get(0)
winner = true
winner == true
no
nextPlayer(player
print("Player " + player+ " enter your ROW")
yes
int r = input
no
yes
print(game.toString())
print(game.toString())
checkForWinner(pos)
print "Player " + player + " wins!"
print("Player " + player+ " enter your COLUMN")
no
int c = input
checkForDraw()
print "No player wins!"
yes
print"Would you like to play again. y/n"
boardPosition current = new boardPosition(r,c)
print(game.toString())
players.clear()
yes
user input == y
no
game.checkSpace(current) == true
yes
game.placeMarker(current, player)
no

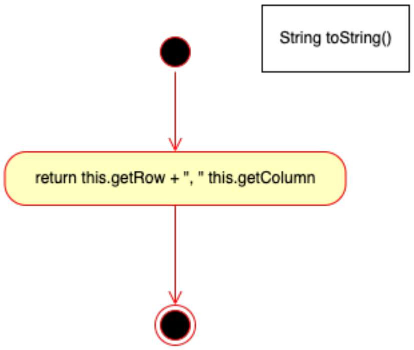
Class 2: BoardPosition

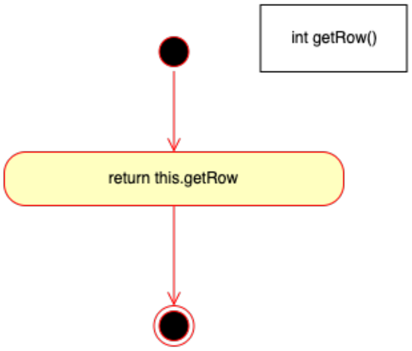
Class diagram

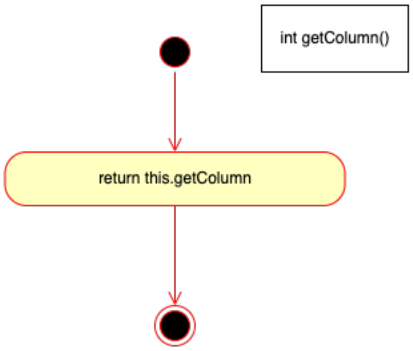


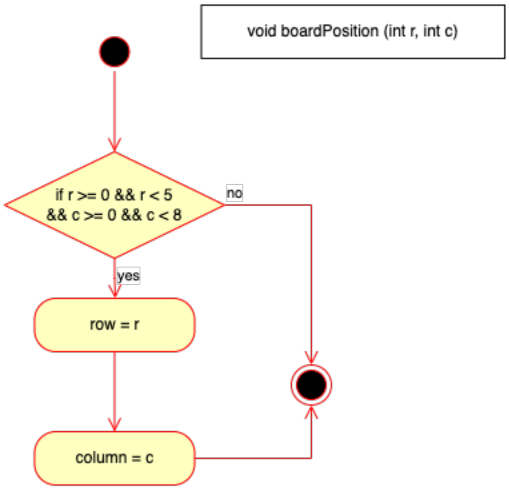
Activity diagrams


boolean equals(BoardPosition Pos
pos.getRow() == this.getRow() &&
pos.getColumn == this.getColumn
no
return false
yes
return true


String toString()
return this.getRow + ", " this.getColumn

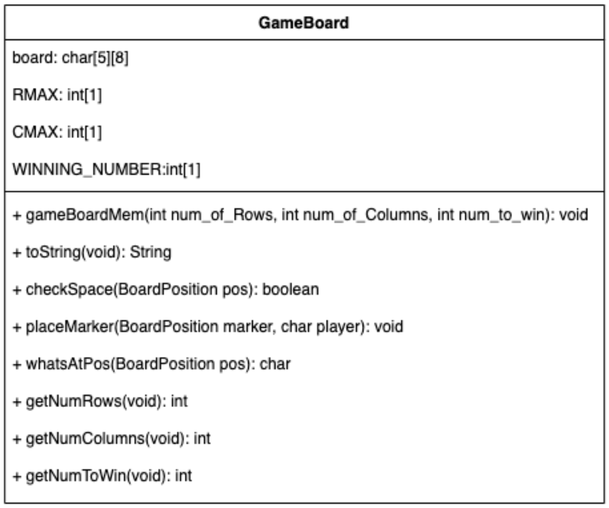

int getRow()
return this.getRow


int getColumn()
return this.getColumn


void boardPosition (int r, int c)
if r >= 0 && r < 5
&& c >= 0 && c < 8
no
yes
row = r
column = c

Class 3: GameBoard

Class diagram



GameBoard

board: char[5][8]

RMAX: int[1]

CMAX: int[1]

WINNING_NUMBER:int[1]

+ gameBoardMem(int num_of_Rows, int num_of_Columns, int num_to_win): void

+ toString(void): String

+ checkSpace(BoardPosition pos): boolean

+ placeMarker(BoardPosition marker, char player): void

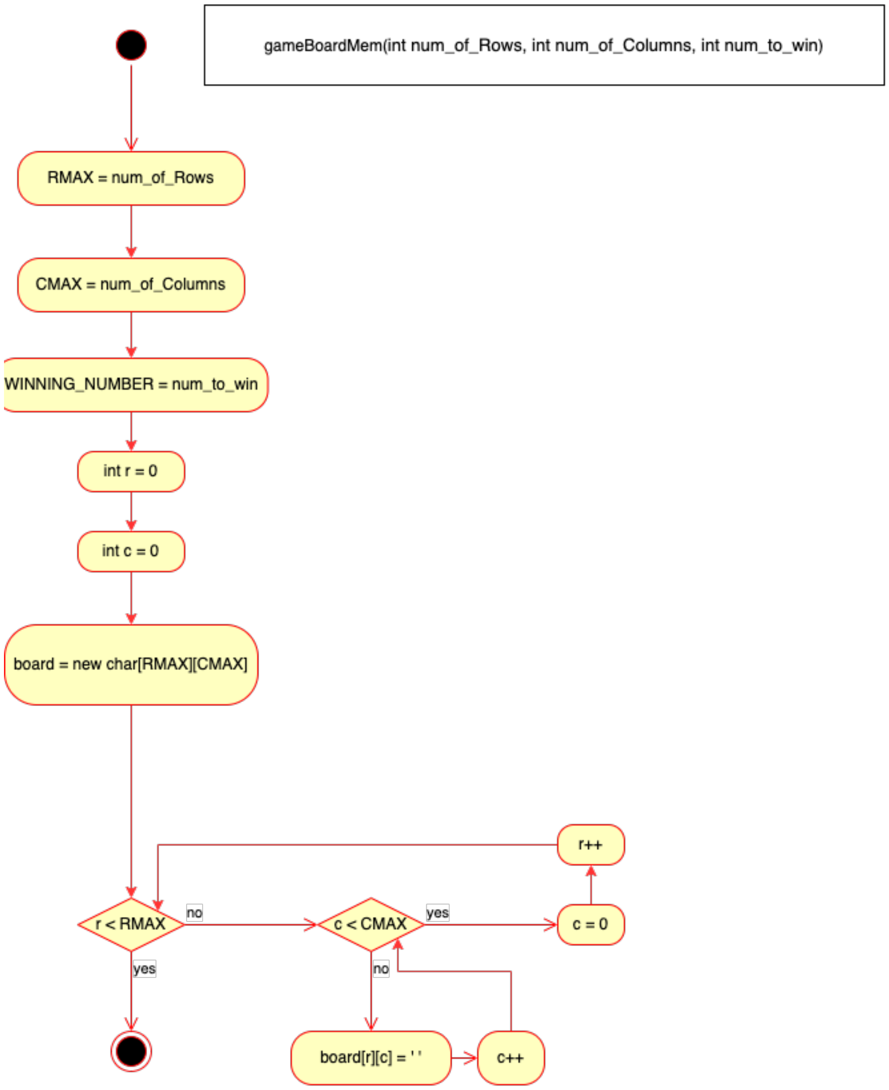
+ whatsAtPos(BoardPosition pos): char

+ getNumRows(void): int

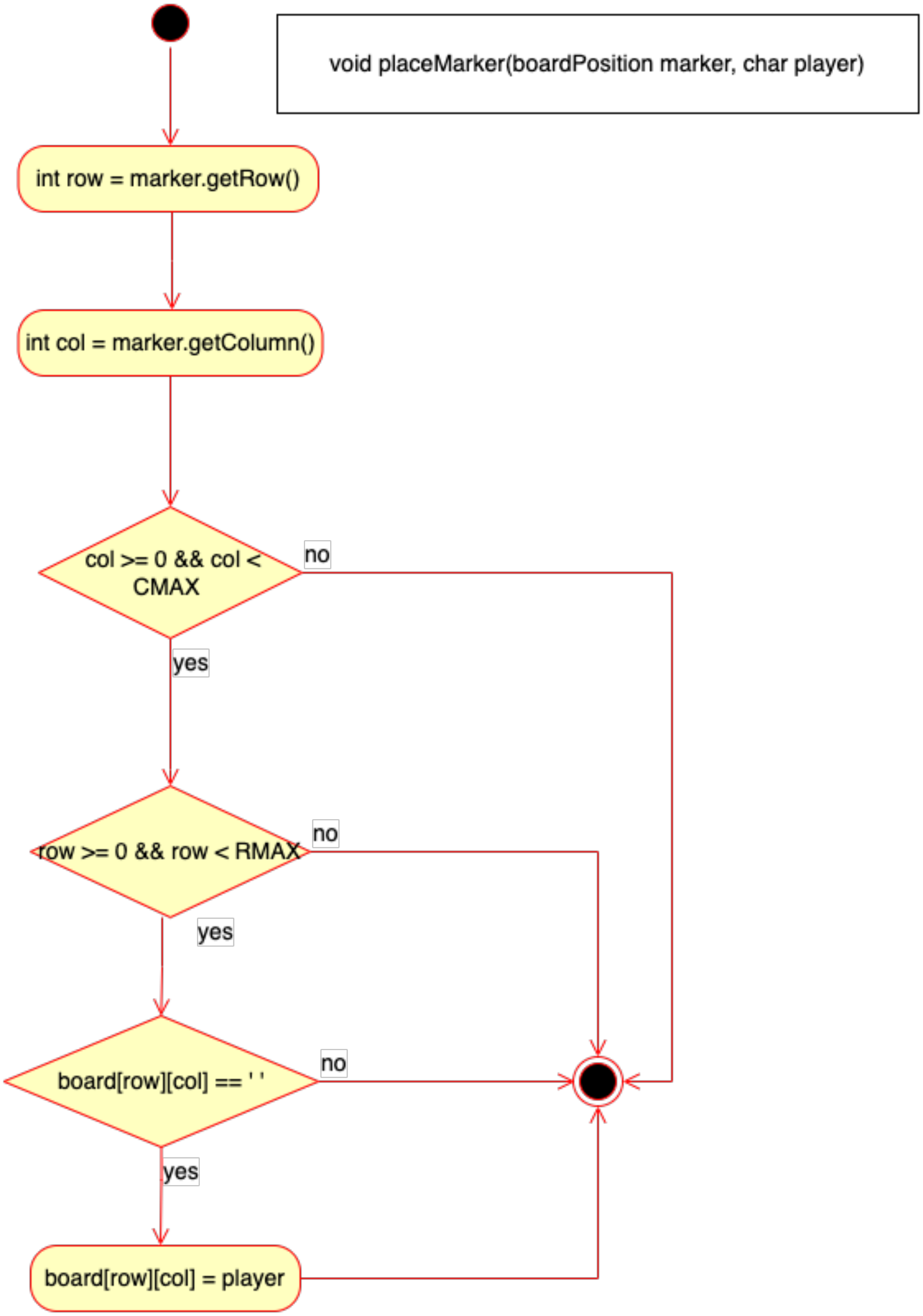
+ getNumColumns(void): int

+ getNumToWin(void): int

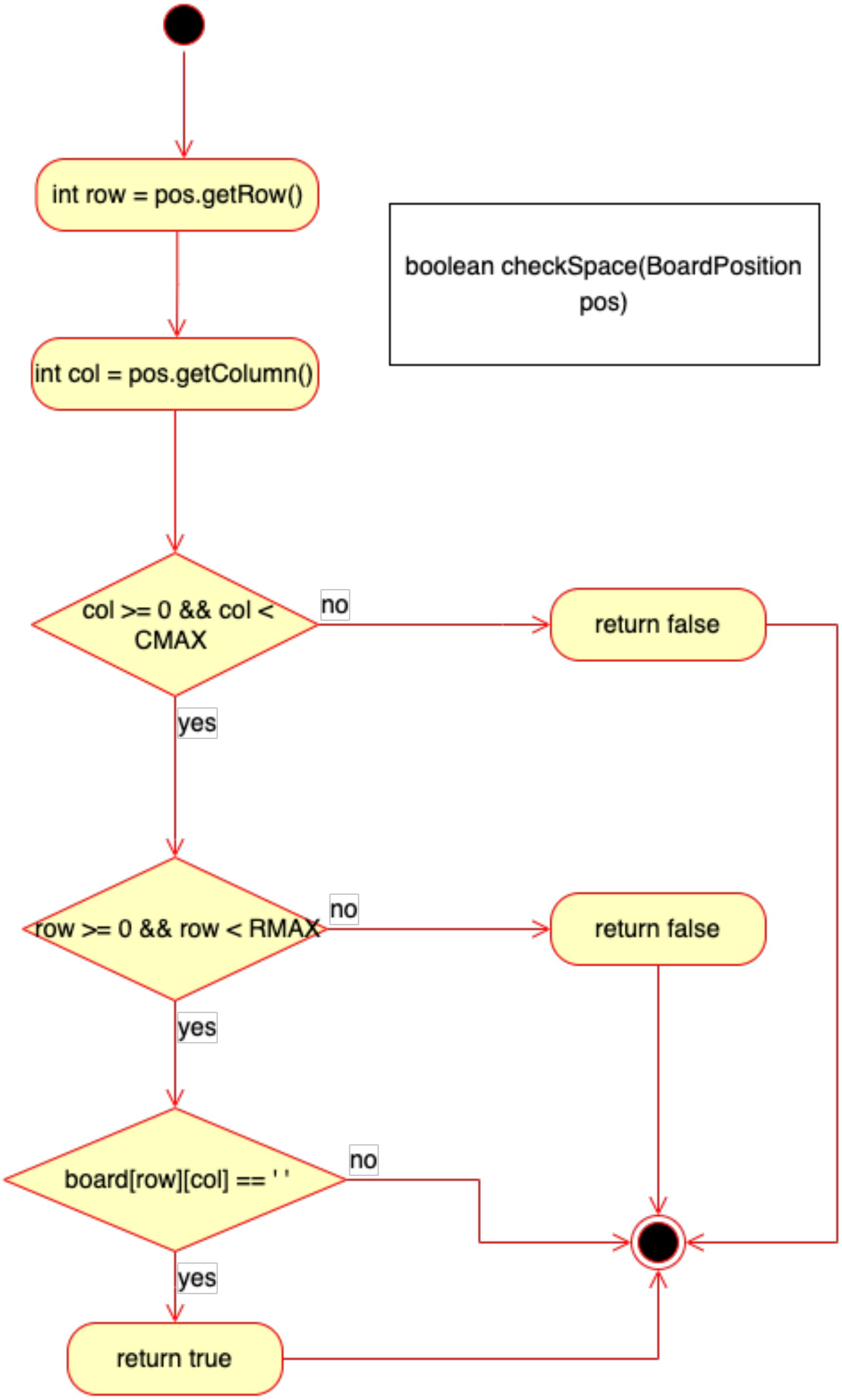
GameBoard Activity diagrams

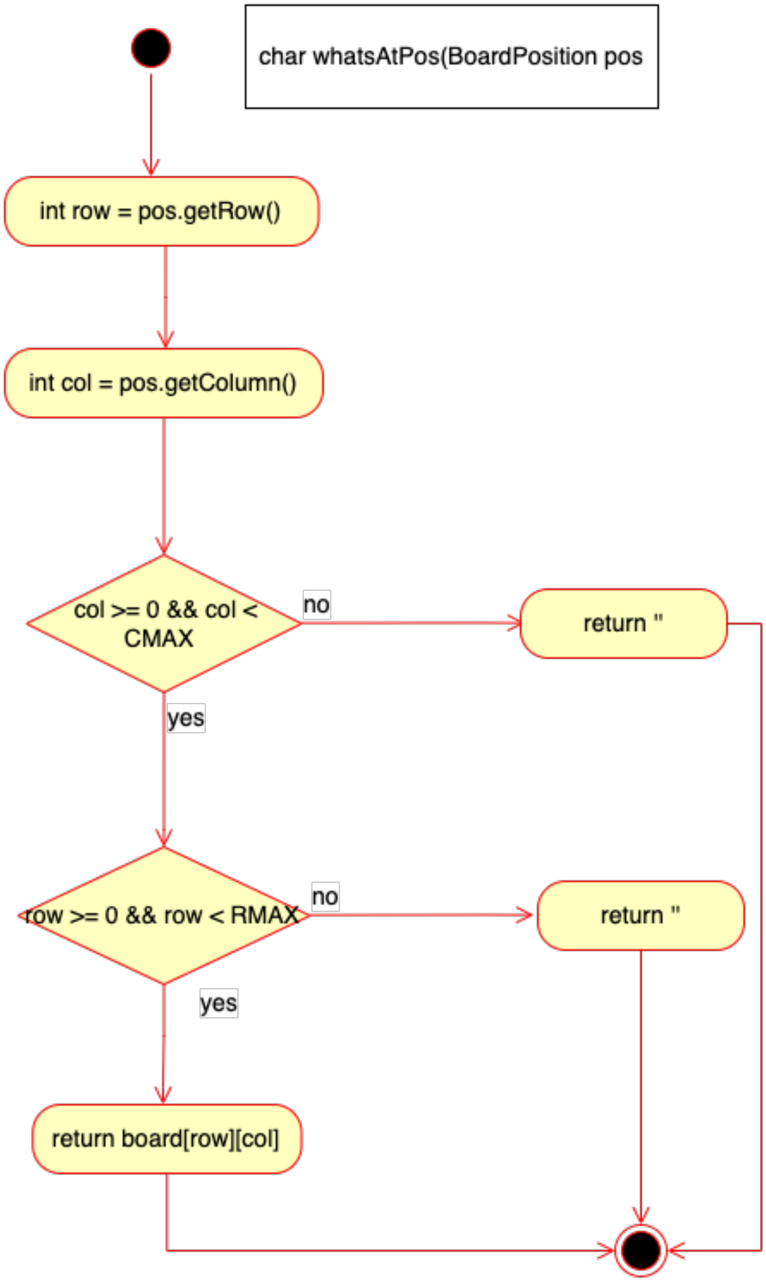


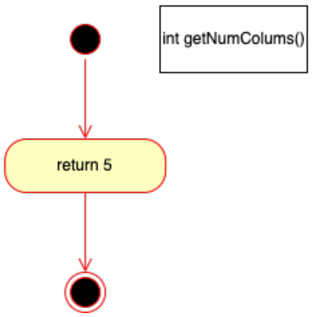
gameBoardMem(int num_of_Rows, int num_of_Columns, int num_to_win)

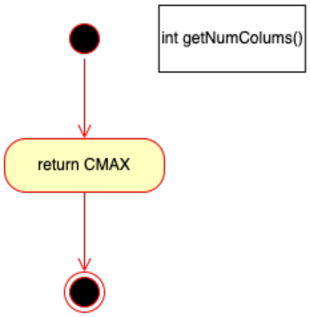


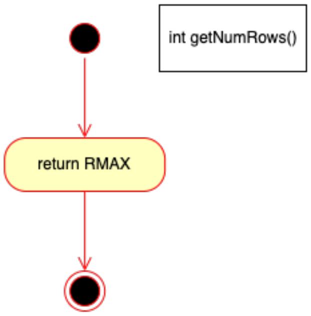
void placeMarker(boardPosition marker, char player)


boolean checkSpace(BoardPosition pos)
int row = pos.getRow()
int col = pos.getColumn()
col >= 0 && col < CMAX
no
return false
yes
row >= 0 && row < RMAX
no
return false
yes
board[row][col] == ' '
no
yes
return true


char whatsAtPos(BoardPosition pos
int row = pos.getRow()
int col = pos.getColumn()
col >= 0 && col < CMAX
no
return ''
yes
row >= 0 && row < RMAX
no
return ''
yes
return board[row][col]

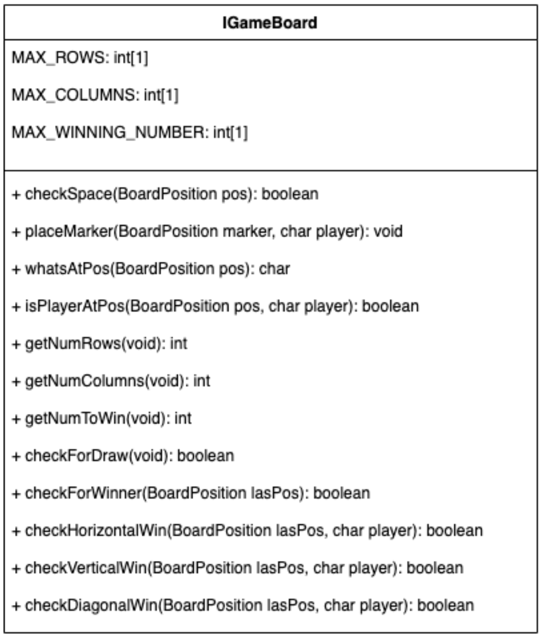

int getNumColums()
return 5


int getNumColums()
return CMAX

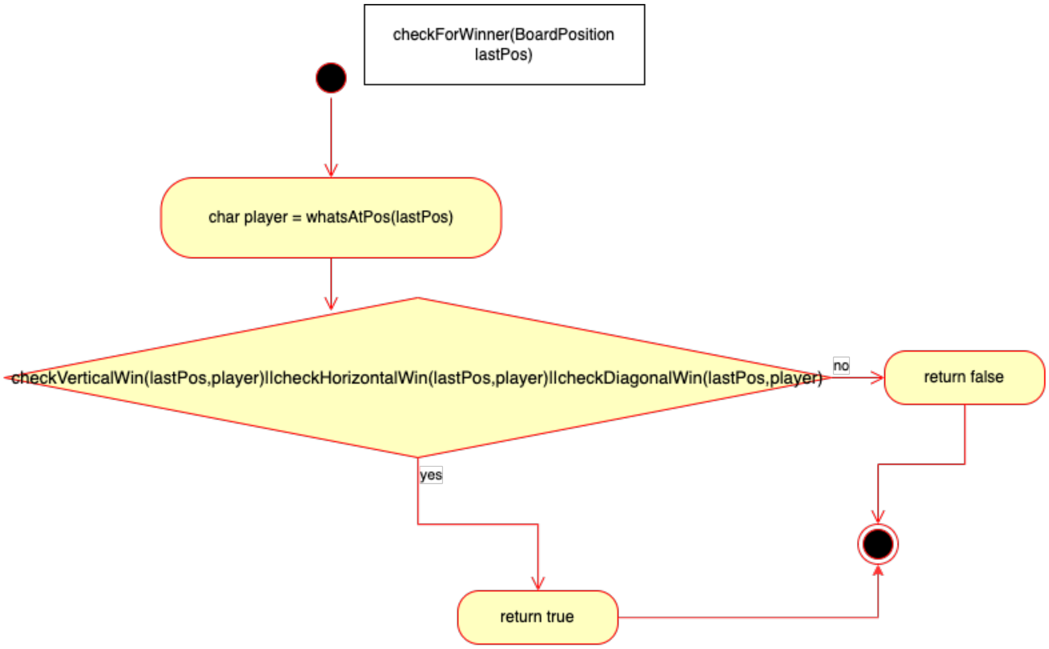

int getNumRows()
return RMAX

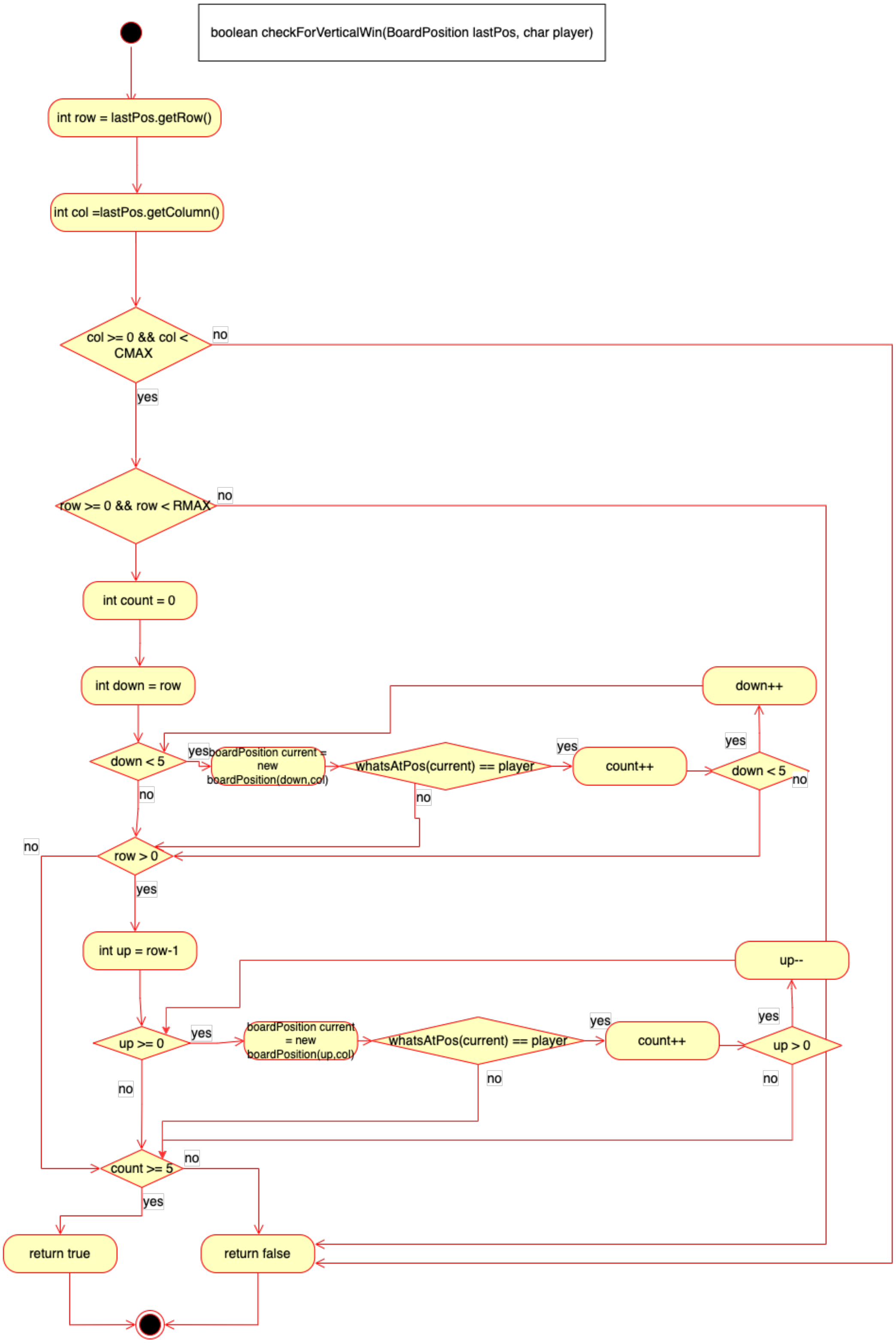
Class 4: IGameBoard

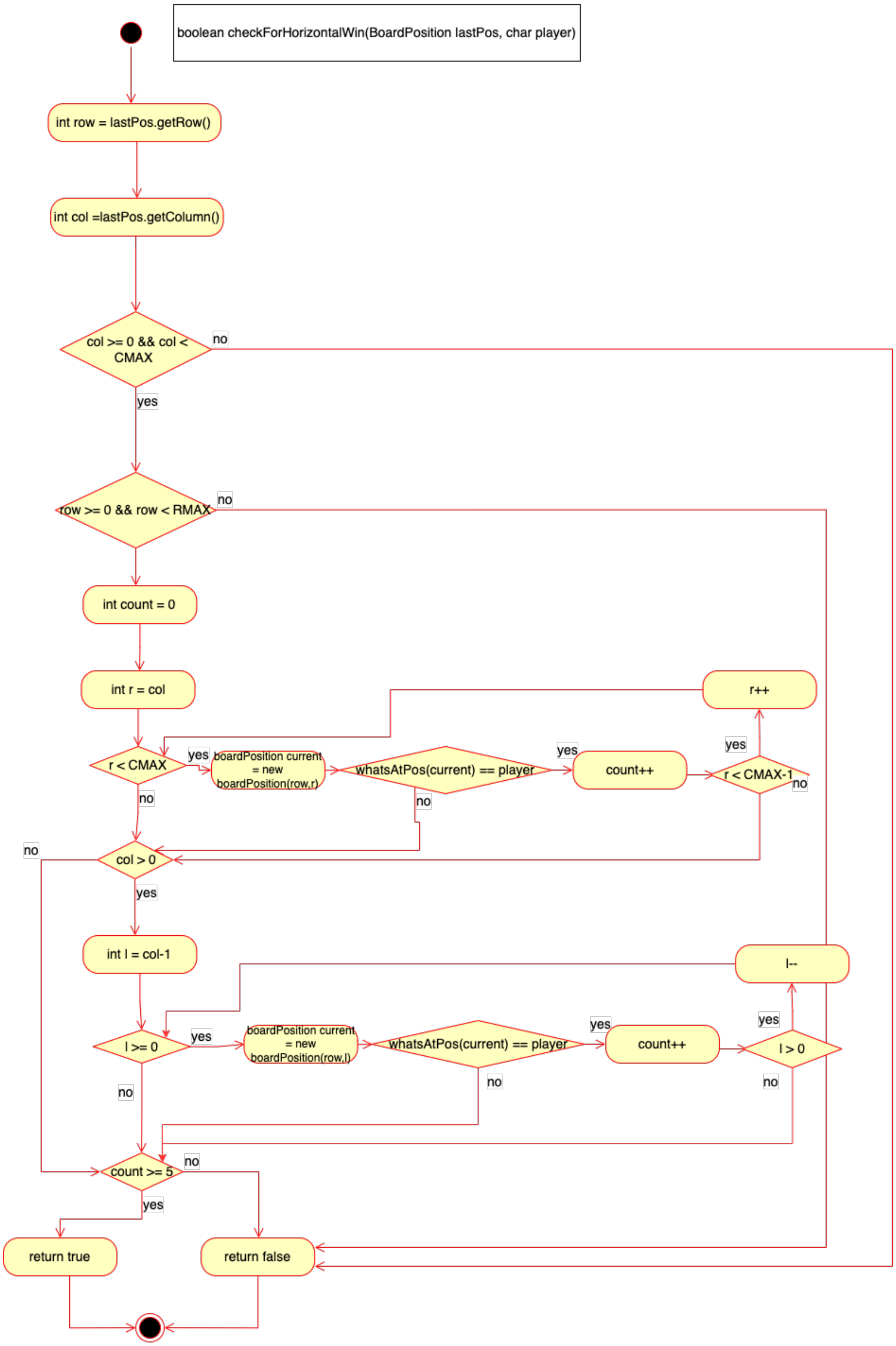
Class Diagram

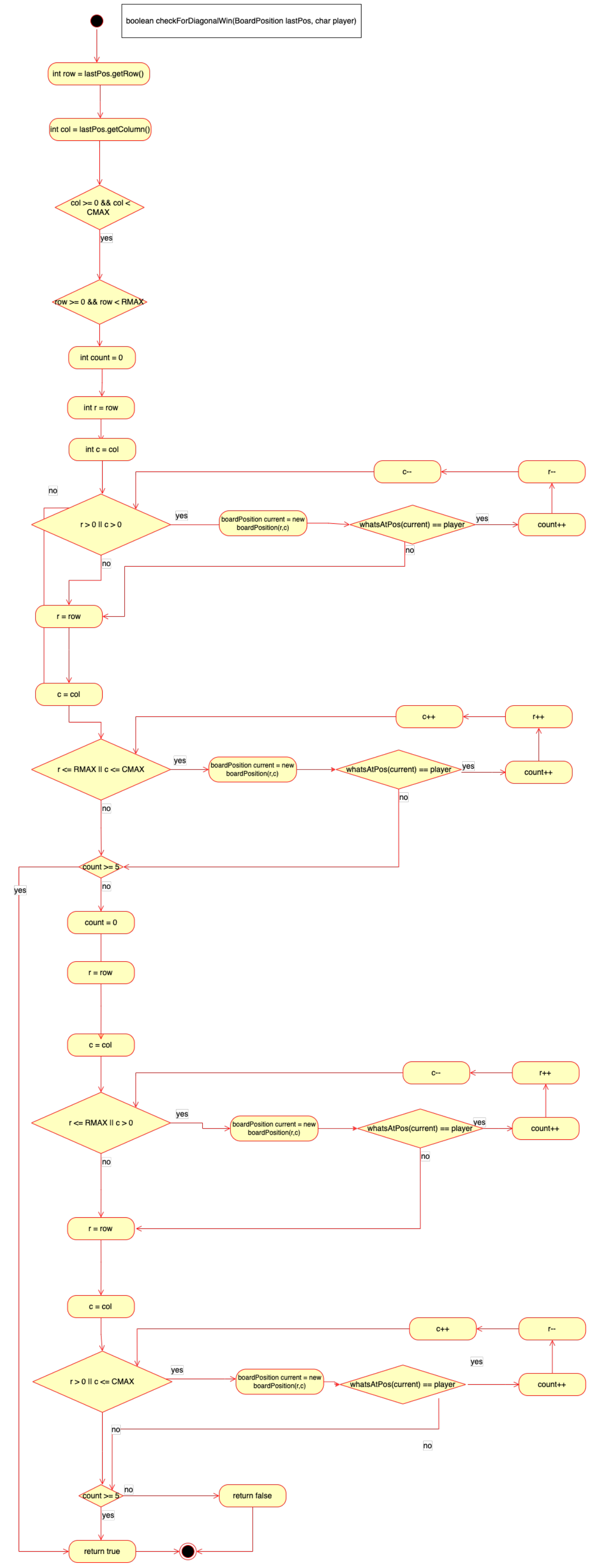

IGameBoard
MAX_ROWS: int[1]
MAX_COLUMNS: int[1]
MAX_WINNING_NUMBER: int[1]
+ checkSpace(BoardPosition pos): boolean
+ placeMarker(BoardPosition marker, char player): void
+ whatsAtPos(BoardPosition pos): char
+ isPlayerAtPos(BoardPosition pos, char player): boolean
+ getNumRows(void): int
+ getNumColumns(void): int
+ getNumToWin(void): int
+ checkForDraw(void): boolean
+ checkForWinner(BoardPosition lasPos): boolean
+ checkHorizontalWin(BoardPosition lasPos, char player): boolean
+ checkVerticalWin(BoardPosition lasPos, char player): boolean
+ checkDiagonalWin(BoardPosition lasPos, char player): boolean

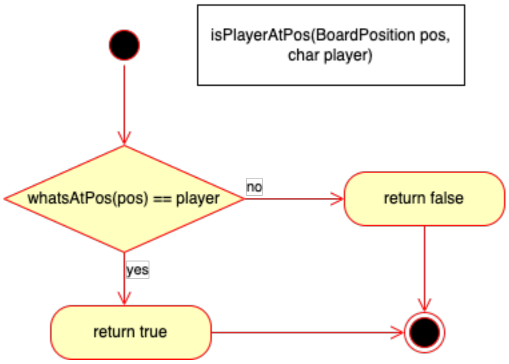
Activity Diagram

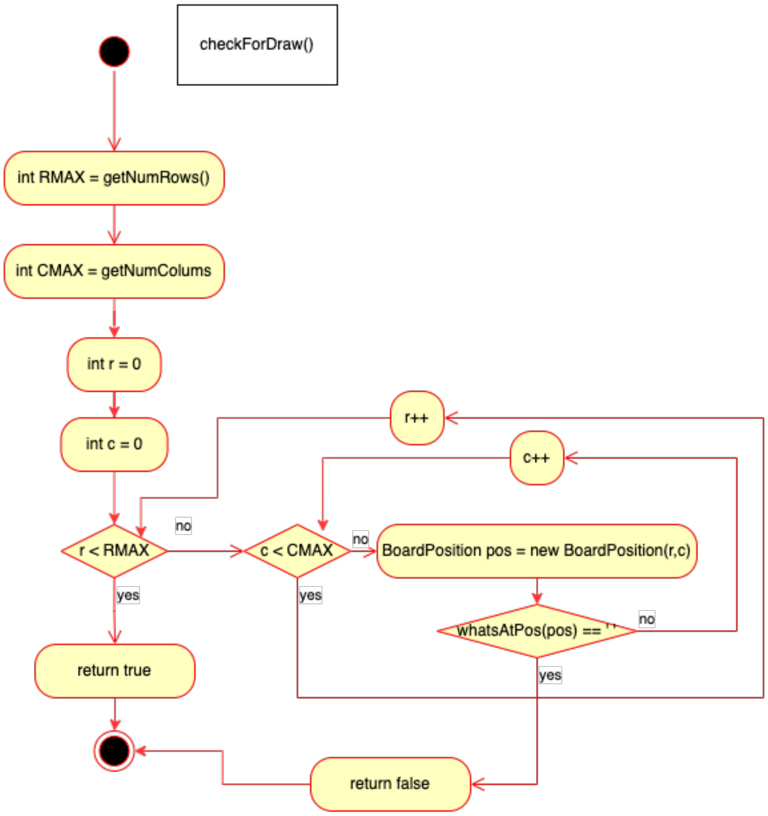

checkForWinner(BoardPosition lastPos)
char player = whatsAtPos(lastPos)
checkVerticalWin(lastPos,player)||checkHorizontalWin(lastPos,player)||checkDiagonalWin(lastPos,player)
no
return false
yes
return true

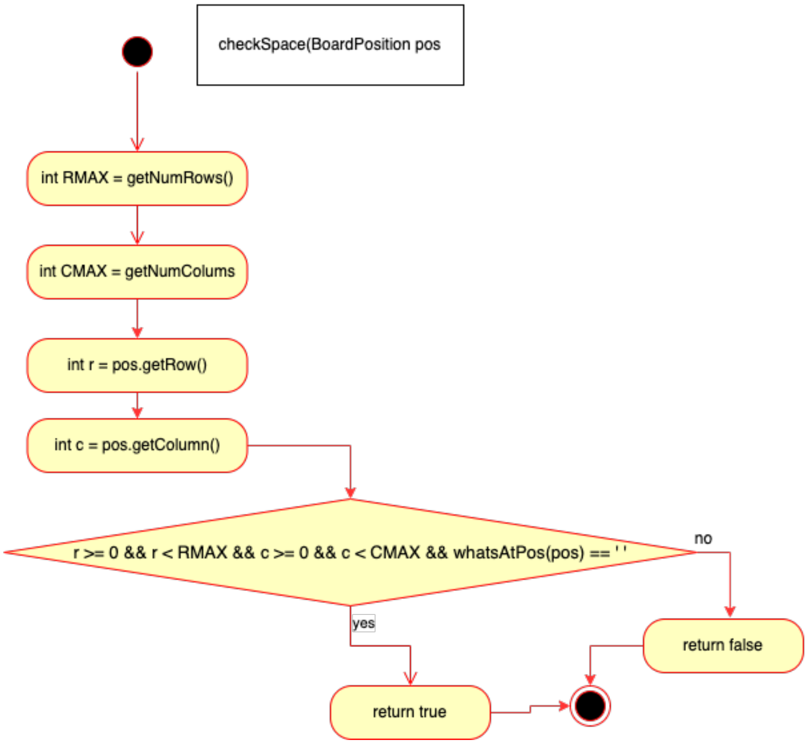

boolean checkForVerticalWin(BoardPosition lastPos, char player)
int row = lastPos.getRow()
int col =lastPos.getColumn()
col >= 0 && col < CMAX
no
yes
row >= 0 && row < RMAX
no
int count = 0
int down = row
down++
down < 5
yes
boardPosition current = new boardPosition(down,col)
whatsAtPos(current) == player
yes
count++
yes
down < 5
no
no
no
no
row > 0
yes
int up = row-1
up--
up >= 0
yes
boardPosition current = new boardPosition(up,col)
whatsAtPos(current) == player
yes
count++
yes
up > 0
no
no
no
count >= 5
no
yes
return true
return false


boolean checkForHorizontalWin(BoardPosition lastPos, char player)
int row = lastPos.getRow()
int col =lastPos.getColumn()
col >= 0 && col < CMAX
no
yes
row >= 0 && row < RMAX
no
int count = 0
int r = col
r++
r < CMAX
yes
boardPosition current = new boardPosition(row,r)
whatsAtPos(current) == player
yes
count++
yes
r < CMAX-1
no
no
no
no
col > 0
yes
int l = col-1
l--
l >= 0
yes
boardPosition current = new boardPosition(row,l)
whatsAtPos(current) == player
yes
count++
yes
l > 0
no
no
no
count >= 5
no
yes
return true
return false


boolean checkForDiagonalWin(BoardPosition lastPos, char player)
int row = lastPos.getRow()
int col = lastPos.getColumn()
col >= 0 && col < CMAX
yes
row >= 0 && row < RMAX
int count = 0
int r = row
int c = col
c--
r--
no
r > 0 || c > 0
yes
boardPosition current = new boardPosition(r,c)
whatsAtPos(current) == player
yes
count++
no
no
r = row
c = col
c++
r++
r <= RMAX || c <= CMAX
yes
boardPosition current = new boardPosition(r,c)
whatsAtPos(current) == player
yes
count++
no
no
count >= 5
yes
no
count = 0
r = row
c = col
c--
r++
r <= RMAX || c > 0
yes
boardPosition current = new boardPosition(r,c)
whatsAtPos(current) == player
yes
count++
no
no
r = row
c = col
c++
r--
r > 0 || c <= CMAX
yes
boardPosition current = new boardPosition(r,c)
whatsAtPos(current) == player
yes
count++
no
no
count >= 5
no
return false
yes
return true

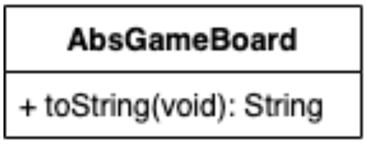

isPlayerAtPos(BoardPosition pos, char player)
whatsAtPos(pos) == player
no
return false
yes
return true


checkForDraw()
int RMAX = getNumRows()
int CMAX = getNumColums
int r = 0
int c = 0
r++
c++
r < RMAX
no
c < CMAX
no
BoardPosition pos = new BoardPosition(r,c)
yes
yes
whatsAtPos(pos) == ' '
no
yes
return true
return false

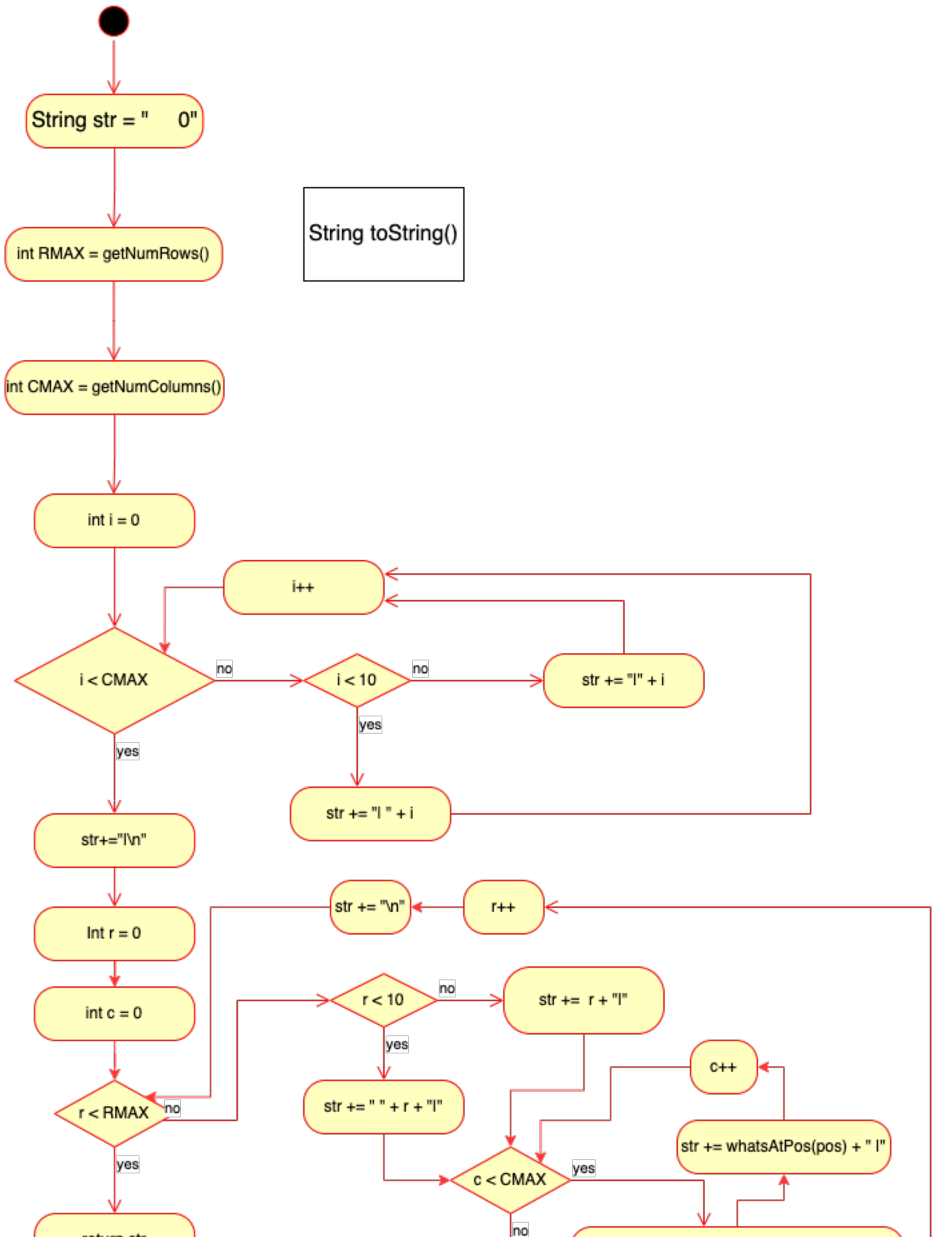

checkSpace(BoardPosition pos
int RMAX = getNumRows()
int CMAX = getNumColums
int r = pos.getRow()
int c = pos.getColumn()
r >= 0 && r < RMAX && c >= 0 && c < CMAX && whatsAtPos(pos) == ' '
no
yes
return false
return true

Class 5: AbsGameBoard

Class Diagram

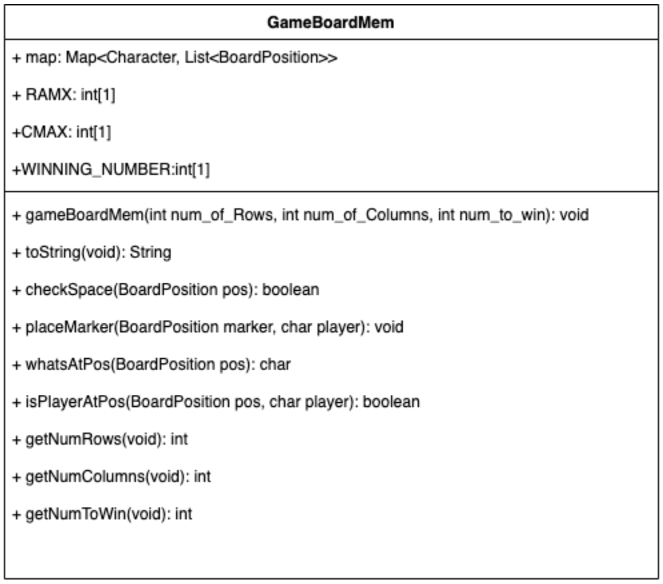


Activity Diagram



Class 6: GameBoardMem

Class Diagrams



GameBoardMem

+ map: Map<Character, List<BoardPosition>>

+ RAMX: int[1]

+CMAX: int[1]

+WINNING_NUMBER:int[1]

+ gameBoardMem(int num_of_Rows, int num_of_Columns, int num_to_win): void

+ toString(void): String

+ checkSpace(BoardPosition pos): boolean

+ placeMarker(BoardPosition marker, char player): void

+ whatsAtPos(BoardPosition pos): char

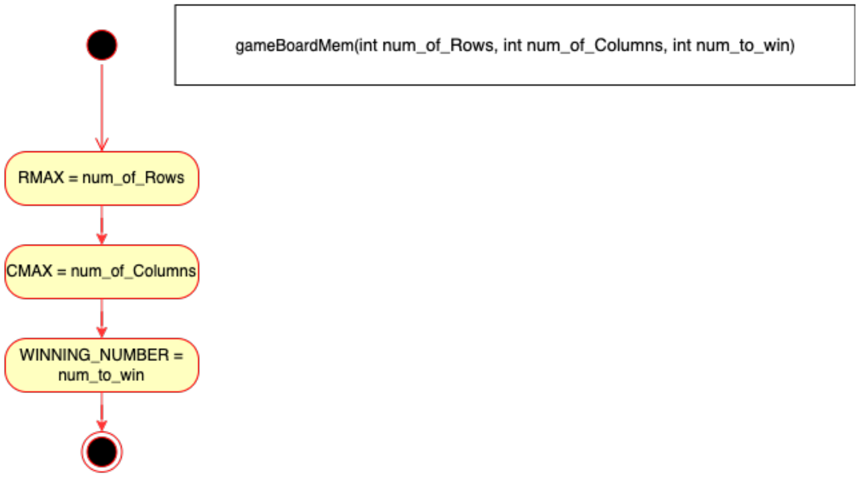
+ isPlayerAtPos(BoardPosition pos, char player): boolean

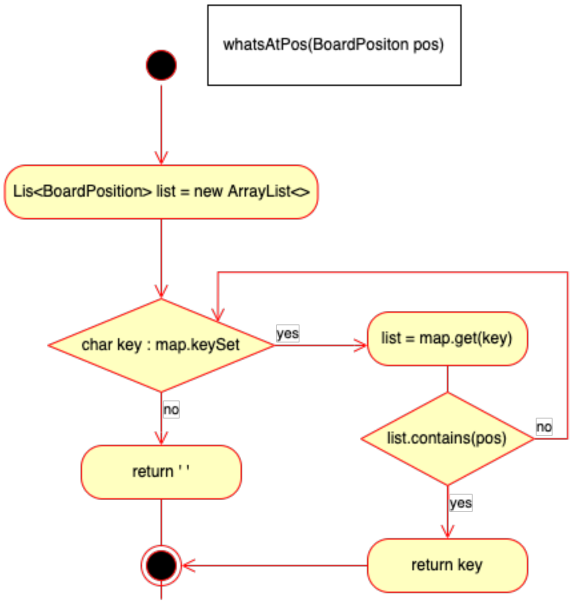
+ getNumRows(void): int

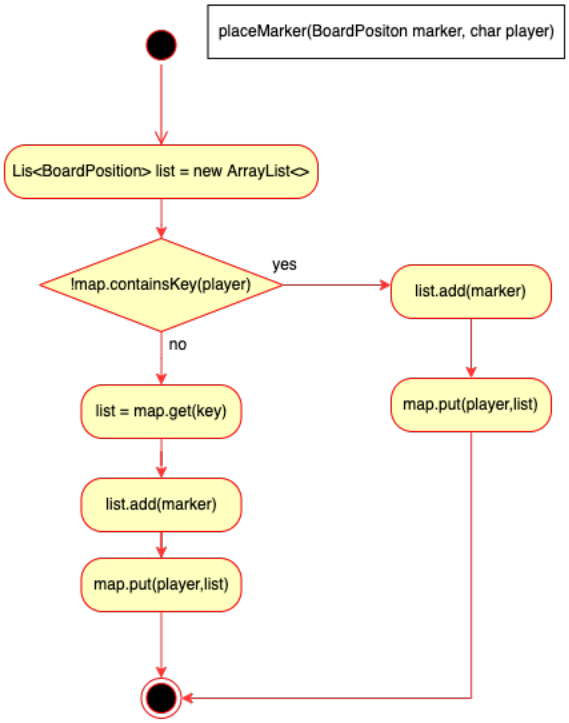
+ getNumColumns(void): int

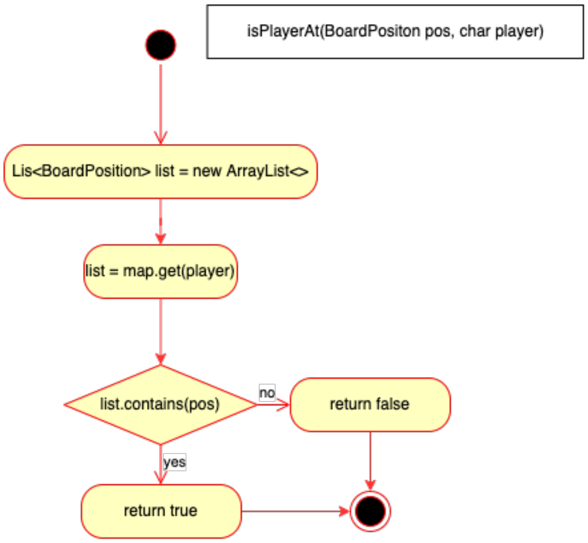
+ getNumToWin(void): int

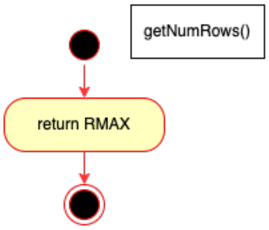
Activity Diagrams

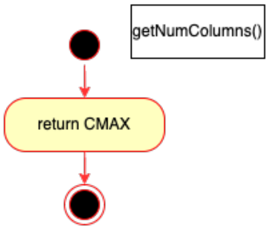

gameBoardMem(int num_of_Rows, int num_of_Columns, int num_to_win)
RMAX = num_of_Rows
CMAX = num_of_Columns
WINNING_NUMBER = num_to_win

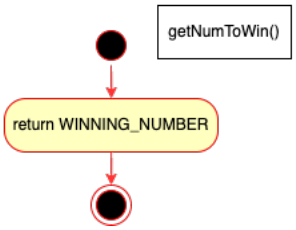

whatsAtPos(BoardPositon pos)
Lis<BoardPosition> list = new ArrayList<>
char key : map.keySet
yes
list = map.get(key)
no
list.contains(pos)
no
return ' '
yes
return key


placeMarker(BoardPositon marker, char player)
Lis<BoardPosition> list = new ArrayList<>
!map.containsKey(player)
yes
list.add(marker)
no
list = map.get(key)
map.put(player,list)
list.add(marker)
map.put(player,list)


isPlayerAt(BoardPositon pos, char player)
Lis<BoardPosition> list = new ArrayList<>
list = map.get(player)
list.contains(pos)
no
return false
yes
return true


getNumRows()
return RMAX


getNumColumns()
return CMAX


getNumToWin()
return WINNING_NUMBER

Testing Plan

Constructor

GameBoard(int r, int c, int n)

<table>
<tr>
<td><b>Input:</b>
r = 3
c = 3
n = 3</td>
<td><b>Output:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></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>
State of the board is created</td>
<td><b>Reason:</b>
This test case is unique and distinct because the gameboard created dimensions are the minimum allowed (3x3)

<b>Function Name:</b>
Constructor1</td>
</tr>
</table>

GameBoard(int r, int c, int n)

<table>
<tr>
<td><b>Input:</b>
r = 12
c = 15
n = 5</td>
<td><b>Output:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>…</td><td>13</td><td>14</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>…</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>11</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>12</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

State of the board is created
Board is shortened to fit, board dimensions are 13x15</td>
<td><b>Reason:</b>
This test case is unique and distinct because the gameboard created dimensions are the within the specified dimensions

<b>Function Name:</b>
Constructor2</td>
</tr>
</table>

GameBoard(int r, int c, int n)

<table>
<tr>
<td><b>Input:</b>
r = 100
c = 100
n = 25</td>
<td><b>Output:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>…</td><td>98</td><td>99</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>…</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>98</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>99</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

State of the board is created
Board is shortened to fit, board dimensions are 100x100</td>
<td><b>Reason:</b>
This test case is unique and distinct because the gameboard created dimensions are the maximum allowed (100x100)

<b>Function Name:</b>
Constructor3</td>
</tr>
</table>

checkSpace

boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
States:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td><td></td><td>O</td></tr>
<tr><td>2</td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow() = 3
pos.getColumn() = 2</td>
<td><b>Output:</b>
State of the board is unchanged

checkSpace = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because it checks a space that is empty within the bounds and returns true

<b>Function Name:</b>
checkSpace1</td>
</tr>
</table>

boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
States:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td><td></td><td>O</td></tr>
<tr><td>2</td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow() = 1
pos.getColumn() = 1</td>
<td><b>Output:</b>
State of the board is unchanged

checkSpace = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because it checks a space that is taken by a player that is within the bounds

<b>Function Name:</b>
checkSpace2</td>
</tr>
</table>

boolean checkSpace(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
States:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td><td></td><td>O</td></tr>
<tr><td>2</td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow() = 6
pos.getColumn() = 2</td>
<td><b>Output:</b>
State of the board is unchanged

checkSpace = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because it checks a space that is outside the bounds of the board and return false

<b>Function Name:</b>
checkSpace3</td>
</tr>
</table>

checkHorizontalWin

boolean checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td>O</td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow() = 1
lastPos.getColumn() = 2
player = ‘X’</td>
<td><b>Output:</b>
State of the board is unchanged

checkHorizontalWin = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because lastPos was placed in the middle of 4 consecutive X pieces, so checkHorizontalWin must check both to the left and to the right.

<b>Function Name:</b>
checkHorizontalWin1</td>
</tr>
</table>

boolean checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>**Input:**
States: (number to win = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>X</td><td>X</td><td></td><td>O</td></tr><tr><td>2</td><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td>O</td><td>O</td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow() = 1
lastPos.getColumn() = 2
player = ‘X’</td>
<td>**Output:**
State of the board is unchanged
checkHorizontalWin = false</td>
<td>**Reason:**
This test case is unique and distinct because lastPos was placed to the right of two pieces resulting in only 3 consecutive horizontal piece so there is no winner
Function Name:
checkHorizontalWin2</td>
</tr>
</table>

boolean checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>**Input:**
States: (number to win = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>O</td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td>O</td><td>O</td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow() = 1
lastPos.getColumn() = 0
player = ‘X’</td>
<td>**Output:**
State of the board is unchanged
checkHorizontalWin = true</td>
<td>**Reason:**
This test case is unique and distinct because lastPos was placed to the left of 3 consecutive player pieces, so checkHorizontalWin must check to the right of the piece.
Function Name:
checkHorizontalWin3</td>
</tr>
</table>

boolean checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>**Input:**
States: (number to win = 4)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td>X</td><td>X</td><td>X</td><td>X</td><td>O</td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td></td><td></td><td>O</td><td>O</td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr></table>lastPos.getRow() = 1
lastPos.getColumn() = 3
player = ‘X’</td>
<td>**Output:**
State of the board is unchanged
checkHorizontalWin = true</td>
<td>**Reason:**
This test case is unique and distinct because lastPos was placed to the right of 3 consecutive player pieces, so checkHorizontalWin must check to the left of the piece.
Function Name:
checkHorizontalWin4</td>
</tr>
</table>

checkVerticalWin

boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td>X</td><td></td></tr>
</table>
lastPos.getRow() = 2
lastPos.getColumn() = 3
player = ‘O’</td>
<td><b>Output:</b>
State of the board is unchanged

checkVerticalWin= true</td>
<td><b>Reason:</b>
This test case is unique and distinct because lastPos was placed in the middle of 4 consecutive player pieces, so checkVerticalWin must check both above and below lastPos.

<b>Function Name:</b>
checkVerticalWin1</td>
</tr>
</table>

boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td>X</td><td></td></tr>
</table>
lastPos.getRow() = 1
lastPos.getColumn() = 3
player = ‘O’</td>
<td><b>Output:</b>
State of the board is unchanged

checkVerticalWin= false</td>
<td><b>Reason:</b>
This test case is unique and distinct because lastPos was placed only next to one piece, so checkVerticalWin will return false

<b>Function Name:</b>
checkVerticalWin2</td>
</tr>
</table>

boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td>X</td><td></td></tr>
</table>
lastPos.getRow() = 0
lastPos.getColumn() = 3
player = ‘O’</td>
<td><b>Output:</b>
State of the board is unchanged

checkVerticalWin= true</td>
<td><b>Reason:</b>
This test case is unique and distinct because lastPos was placed above 3 consecutive player pieces, so checkVerticalWin must check both below lastPos.

<b>Function Name:</b>
checkVerticalWin3</td>
</tr>
</table>

boolean checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>X</td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td>X</td><td></td></tr>
</table>
lastPos.getRow() = 3
lastPos.getColumn() = 3
player = ‘O’</td>
<td><b>Output:</b>
State of the board is unchanged

checkVerticalWin= true</td>
<td><b>Reason:</b>
This test case is unique and distinct because lastPos was placed below 3 consecutive player pieces, so checkVerticalWin must check both above lastPos.

<b>Function Name:</b>
checkVerticalWin4</td>
</tr>
</table>

checkDiagonalWin

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td>O</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td>X</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow() = 1
lastPos.getColumn() = 1
player = ‘X’</td>
<td><b>Output:</b>
State of the board is unchanged

checkDiagonalWin = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because the player does not have 4 in a row so checkDiagonalWin will be false

<b>Function Name:</b>
checkDiagonalWin1</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td>O</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>O</td><td></td><td>X</td><td></td></tr>
<tr><td>4</td><td></td><td>O</td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow() = 1
lastPos.getColumn() = 1
player = ‘X’</td>
<td><b>Output:</b>
State of the board is unchanged

checkDiagonalWin = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because the marker pos was placed in the middle of 4 consecutive diagonal markers in the northwest, southeast direction, so checkDiagonalWin has to check both the northwest, and southeast directions

<b>Function Name:</b>
checkDiagonalWin2</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td>O</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>O</td><td></td><td>X</td><td></td></tr>
<tr><td>4</td><td></td><td>O</td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow() = 0
lastPos.getColumn() = 0
player = ‘X’</td>
<td><b>Output:</b>
State of the board is unchanged

checkDiagonalWin = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because the marker lastPos has been placed at the northwest edge of 3 consecutive markers so checkDiagonalWin must check in the southeast direction

<b>Function Name:</b>
checkDiagonalWin3</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td>O</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td>O</td><td></td><td>X</td><td></td></tr>
<tr><td>4</td><td></td><td>O</td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow() = 3
lastPos.getColumn() = 3
player = ‘X’</td>
<td><b>Output:</b>
State of the board is unchanged

checkDiagonalWin = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because the marker lastPos has been placed at the southeast edge of 3 consecutive markers so checkDiagonalWin must check in the northwest direction

<b>Function Name:</b>
checkDiagonalWin4</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>O</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow() = 2
lastPos.getColumn() = 1
player = ‘O’</td>
<td><b>Output:</b>
State of the board is unchanged

checkDiagonalWin = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because the marker pos was placed in the middle of 4 consecutive diagonal markers in the northeast, southwest directions, so checkDiagonalWin has to check both the northeast, and southwest directions

<b>Function Name:</b>
checkDiagonalWin5</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>O</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow() = 0
lastPos.getColumn() = 3
player = ‘O’</td>
<td><b>Output:</b>
State of the board is unchanged

checkDiagonalWin = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because the marker lastPos has been placed at the northeast edge of 3 consecutive markers so checkDiagonalWin must check in the southwest direction

<b>Function Name:</b>
checkDiagonalWin6</td>
</tr>
</table>

boolean checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td>X</td><td></td><td></td><td>O</td><td></td></tr>
<tr><td>1</td><td>X</td><td>X</td><td>O</td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>O</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
lastPos.getRow() = 3
lastPos.getColumn() = 0
player = ‘O’</td>
<td><b>Output:</b>
State of the board is unchanged

checkDiagonalWin = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because the marker lastPos has been placed at the southwest edge of 3 consecutive markers so checkDiagonalWin must check in the northeast direction

<b>Function Name:</b>
checkDiagonalWin7</td>
</tr>
</table>

checkForDraw

boolean checkForDraw()

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>X</td><td>X</td></tr>
</table>
</td>
<td><b>Output:</b>
State is unchanged

checkForDraw = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because there is no winner and ever space is taken on the board

<b>Function Name:</b>
checkForDraw1</td>
</tr>
</table>

boolean checkForDraw()

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td>X</td></tr>
<tr><td>1</td><td>X</td><td>O</td><td>O</td></tr>
<tr><td>2</td><td>O</td><td>X</td><td></td></tr>
</table>
</td>
<td><b>Output:</b>
State is unchanged

checkForDraw = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because there is no winner but there is a single empty space

<b>Function Name:</b>
checkForDraw2</td>
</tr>
</table>

boolean checkForDraw()

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td></td><td>X</td></tr>
<tr><td>1</td><td></td><td>O</td><td>O</td></tr>
<tr><td>2</td><td></td><td></td><td>X</td></tr>
</table>
</td>
<td><b>Output:</b>
State is unchanged

checkForDraw = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because although because there are multiple players on the board, and multiple empty spaces.

<b>Function Name:</b>
checkForDraw3</td>
</tr>
</table>

boolean checkForDraw()

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></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>
</td>
<td><b>Output:</b>
State is unchanged

checkForDraw = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because there are no players on the board

<b>Function Name:</b>
checkForDraw4</td>
</tr>
</table>

whatsAtPos

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>1</td><td></td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table>
pos.getRow() = 0
pos.getColumn() = 1</td>
<td><b>Output:</b>
State is unchanged

whatsAtPos = ‘X’</td>
<td><b>Reason:</b>
This test case is unique and distinct because there Is a player at the marker pos

<b>Function Name:</b>
whatsAtPos1</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>1</td><td></td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table>
pos.getRow() = 2
pos.getColumn() = 1</td>
<td><b>Output:</b>
State is unchanged

whatsAtPos = ‘ ‘</td>
<td><b>Reason:</b>
This test case is unique and distinct because there is no player at the marker pos

<b>Function Name:</b>
whatsAtPos2</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></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>
pos.getRow() = 1
pos.getColumn() = 1</td>
<td><b>Output:</b>
State is unchanged

whatsAtPos = ‘ ‘</td>
<td><b>Reason:</b>
This test case is unique and distinct because there are no players on the board

<b>Function Name:</b>
whatsAtPos3</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>1</td><td></td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table>
pos.getRow() = 0
pos.getColumn() = 0</td>
<td><b>Output:</b>
State is unchanged

whatsAtPos = ‘X’</td>
<td><b>Reason:</b>
This test case is unique and distinct because it is testing the left and upper boundaries of the board

<b>Function Name:</b>
whatsAtPos4</td>
</tr>
</table>

char whatsAtPos(BoardPosition pos)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>X</td><td>O</td></tr>
<tr><td>1</td><td></td><td>O</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
</table>
pos.getRow() = 2
pos.getColumn() = 2</td>
<td><b>Output:</b>
State is unchanged

whatsAtPos = ‘ ‘</td>
<td><b>Reason:</b>
This test case is unique and distinct because it is testing the right and lower boundaries of the board

<b>Function Name:</b>
whatsAtPos5</td>
</tr>
</table>

isPlayerAtPos

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>O</td></tr>
</table>
pos.getRow() = 1
pos.getColumn() = 1
player = ‘X’</td>
<td><b>Output:</b>
State is unchanged

isPlayerAtPos = true</td>
<td><b>Reason:</b>
This test case is unique and distinct because the correct player is at the marker pos

<b>Function Name:</b>
isPlayerAtPos1</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>O</td></tr>
</table>
pos.getRow() = 1
pos.getColumn() = 2
player = ‘O’</td>
<td><b>Output:</b>
State is unchanged

isPlayerAtPos = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because the marker pos is an empty spot

<b>Function Name:</b>
isPlayerAtPos2</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>O</td></tr>
</table>
pos.getRow() = 1
pos.getColumn() = 1
player = ‘O’</td>
<td><b>Output:</b>
State is unchanged

isPlayerAtPos = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because the marker pos is contains a different player than player ‘O’

<b>Function Name:</b>
isPlayerAtPos3</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>O</td></tr>
</table>
pos.getRow() = 1
pos.getColumn() = 1
player = ‘A’</td>
<td><b>Output:</b>
State is unchanged

isPlayerAtPos = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because player is not one of the current players playing in the game

<b>Function Name:</b>
isPlayerAtPos4</td>
</tr>
</table>

boolean isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>0</td><td>X</td><td>O</td><td></td></tr>
<tr><td>1</td><td></td><td>X</td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>O</td></tr>
</table>
pos.getRow() = 3
pos.getColumn() = 0
player = ‘X’</td>
<td><b>Output:</b>
State is unchanged

isPlayerAtPos = false</td>
<td><b>Reason:</b>
This test case is unique and distinct because the marker pos is outside of the bounds of the board

<b>Function Name:</b>
isPlayerAtPos5</td>
</tr>
</table>

placeMarker

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
marker.getRow() = 3
marker.getColumn() = 3
player = ‘A’</td>
<td><b>Output:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td></td><td></td><td>A</td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
This test case is unique and distinct because I am placing a marker representing a player who has not been placed on this board before.

<b>Function Name:</b>
placeMarker1</td>
</tr>
</table>

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
marker.getRow() = 2
marker.getColumn() = 1
player = ‘A’</td>
<td><b>Output:</b>
The state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because I am placing a marker in a pos that is already taken by another marker

<b>Function Name:</b>
placeMarker2</td>
</tr>
</table>

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td>O</td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>X</td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
marker.getRow() = 0
marker.getColumn() = 5
player = ‘A’</td>
<td><b>Output:</b>
The state of the board is unchanged</td>
<td><b>Reason:</b>
This test case is unique and distinct because I am placing a marker in a pos that is invalid

<b>Function Name:</b>
placeMarker3</td>
</tr>
</table>

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 4)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table>
marker.getRow() = 2
marker.getColumn() = 2
player = ‘X’</td>
<td><b>Output:</b>
State:
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td></td><td></td><td>X</td><td></td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td>
<td><b>Reason:</b>
This test case is unique and distinct because I am placing a marker representing a player who is the first player placed on the board

<b>Function Name:</b>
placeMarker4</td>
</tr>
</table>

void placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td><b>Input:</b>
States: (number to win = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td>O</td><td>X</td></tr><tr><td>1</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>O</td><td>X</td><td></td></tr></table>marker.getRow() = 2
marker.getColumn() = 2
player = ‘X’</td>
<td><b>Output:</b>
State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>0</td><td>X</td><td>O</td><td>X</td></tr><tr><td>1</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>O</td><td>X</td><td>X</td></tr></table></td>
<td><b>Reason:</b>
This test case is unique and distinct because I am placing a marker representing a player who will take the last remaining open spot on the board

<b>Function Name:</b>
placeMarker5</td>
</tr>
</table>

Deployment

The “make default” command will compile all of the files that is needed.

Once the java files are compiled, the “make run” command will run the files and the program will start executing.

Once the program is done running, the “make clean” command will remove any execs files created when compiling and running

The “make test” command will run after compiling via “make default” and will run the testGameBoard and testGameBoardMem junit testing files.