CPSC 2150 Project Report

Sumanth Pandiri

Requirements Analysis

Functional Requirements:

1. As a user, I need to be able to start the game so that I can play it.
2. As a user, I need to be able to know who's turn it is so that I know what to do.
3. As a user, I need to know what my token is so I know which one I am placing.
4. As a user, I need to be able to select a spot on the board so I can put the character down.
5. As a user, I need to be able to view where the other player places their characters so that I can make a decision.
6. As a user, I need to be able to see the entire board so I can make a decision.
7. As a user, I need to be able to see who won so I know what the result was.
8. As a user, I need to be able to see the winning board when I win so I can remember what I did.
9. As a user, I need to be able to see the board that draws so I can see why we drew.
10. As a user, I need to be able to see the board that I lost on so I can analyze my mistakes.
11. As a user, I need the system to tell me if I can't place my character somewhere to prevent messing up the game.
12. As a user, I need the game to ask me if I want to play again so I can choose whether I want to play again or not.
13. As a user, I need the game to let me choose to play again so that I can play another round.
14. As a user, I need the game to let me quit after the round so I don't have to play another round.
15. As a user, I need to know how many in a row I have so I can strategize.
16. As a user, I need to know how many in a row the computer has so I can strategize.
17. As a user, I need to be able to see where there are empty spaces so I can make a decision.
18. As a user, I need to know what buttons I need to press to play the game.
19. As a user, I need to know what buttons I need to press to place a token.
20. As a user, I need to know the instructions for the game before I start playing.
21. As a user, I need to know the rules for the game before I start playing.
22. As a user, I need to know what the end goal of the game is before I start playing.
23. As a user, I need the inputting methods to be simple so that it is easy to figure out how to play.
24. As a user, I need to know if the board is full so I know if the situation is a draw.
25. As a user, I need to know if I have vertical win so that I know if I won.
26. As a user, I need to know if I have horizontal win so that I know if I won.
27. As a user, I need to know if I have diagonal win so that I know if I won.
28. As a user, I need to know if an opponent has vertical win so I know if I lost.
29. As a user, I need to know if an opponent has horizontal win so I know if I lost.
30. As a user, I need to know if an opponent has diagonal win so I know if I lost.
31. As a user, I should be able to set the size of the board within the boundaries provided so that I can control the game board size to my desire.

32. As a user, I should be able to set the number of tokens needed to win within the boundaries provided so I can control how difficult it is to win.
33. As a user, I should be able to select a character that nobody else has picked.
34. As a user, the character I select should be unique so that nobody can pick it.
35. As a user, I should be able to choose what kind of implementation I want, fast or slow, so that my system can handle the game.
36. As a user, I should be able to make decisions on if I want to play again so that I can play again.
37. As a user, I should be able to select how many users will play the game.

Non-Functional Requirements

1. Must compile on all machines without errors
2. Must run on all machines without mistakes
3. Must be written in java
4. Must have information hiding so the user can't access the wrong variables
5. Must be able to be quitted by the user
6. Must run efficiently and quickly
7. Board is of size 5x8
8. X always goes first
9. (0, 0) is at the top left of the board
10. All outputs to the screen must be accurate and easy to understand
11. The game must run according to the rules.
12. The GUI must display and be legible
13. The buttons should be clickable and visible
14. The message printed should be visible and readable
15. The program must properly pass input through input validation and to the controller

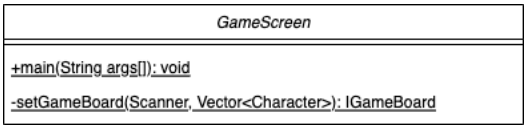
Deployment Instructions

Details in Projects 2-5.

System Design

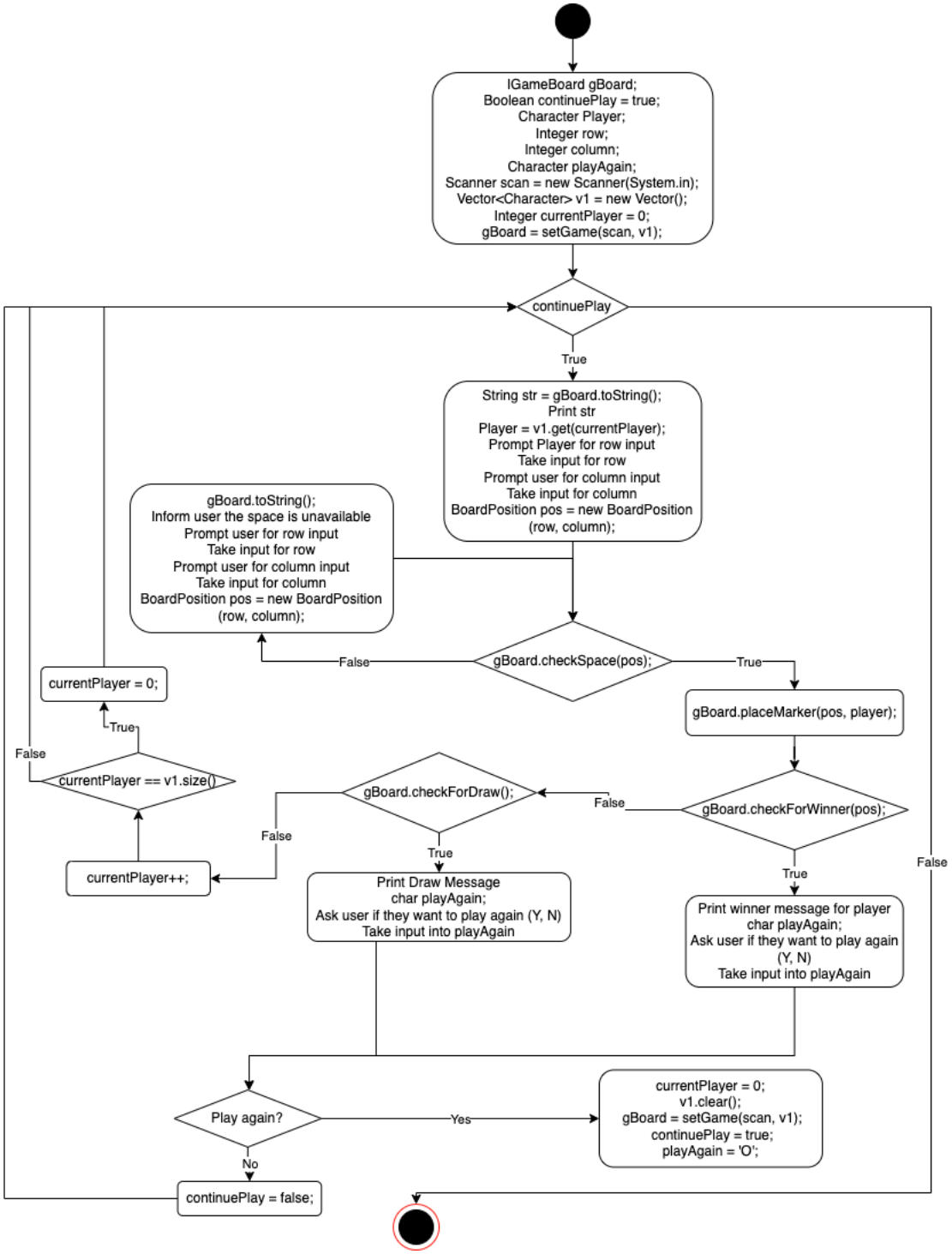
Class 1: GameScreen

Class diagram

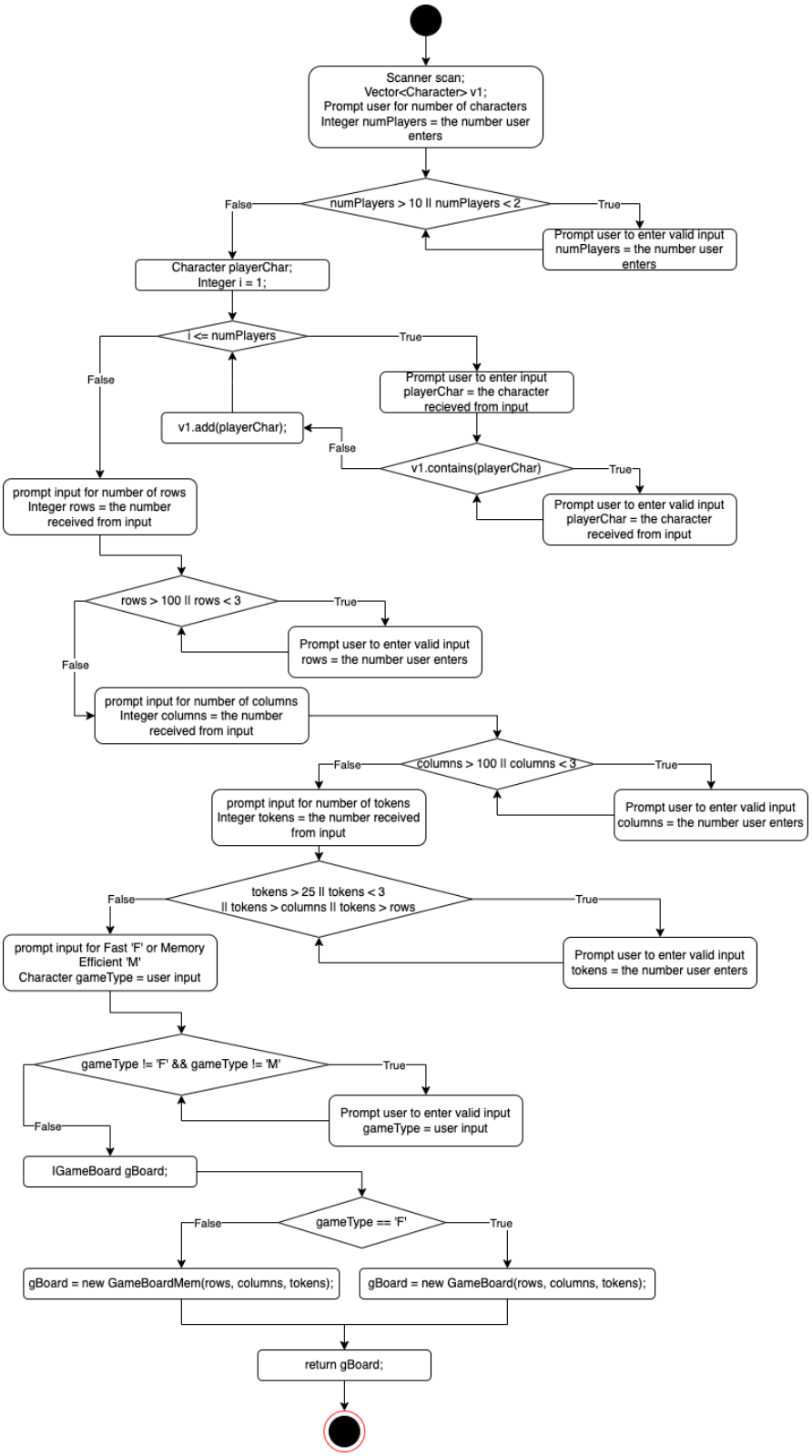


Activity diagrams

main(String[] args):

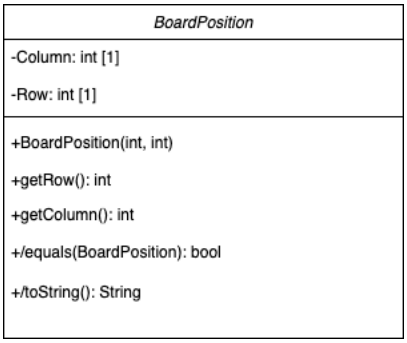


setGame(Scanner scan, Vector<Character> v1):


Scanner scan;
Vector<Character> v1;
Prompt user for number of characters
Integer numPlayers = the number user enters
numPlayers > 10 || numPlayers < 2
False
True
Prompt user to enter valid input
numPlayers = the number user enters
Character playerChar;
Integer i = 1;
i <= numPlayers
True
False
Prompt user to enter input
playerChar = the character recieved from input
v1.add(playerChar);
False
v1.contains(playerChar)
True
Prompt user to enter valid input
playerChar = the character received from input
prompt input for number of rows
Integer rows = the number received from input
rows > 100 || rows < 3
True
Prompt user to enter valid input
rows = the number user enters
False
prompt input for number of columns
Integer columns = the number received from input
False
columns > 100 || columns < 3
True
Prompt user to enter valid input
columns = the number user enters
prompt input for number of tokens
Integer tokens = the number received from input
False
tokens > 25 || tokens < 3
|| tokens > columns || tokens > rows
True
Prompt user to enter valid input
tokens = the number user enters
prompt input for Fast 'F' or Memory Efficient 'M'
Character gameType = user input
gameType != 'F' && gameType != 'M'
True
Prompt user to enter valid input
gameType = user input
False
IGameBoard gBoard;
False
gameType == 'F'
True
gBoard = new GameBoardMem(rows, columns, tokens);
gBoard = new GameBoard(rows, columns, tokens);
return gBoard;

Class 2: BoardPosition

Class diagram



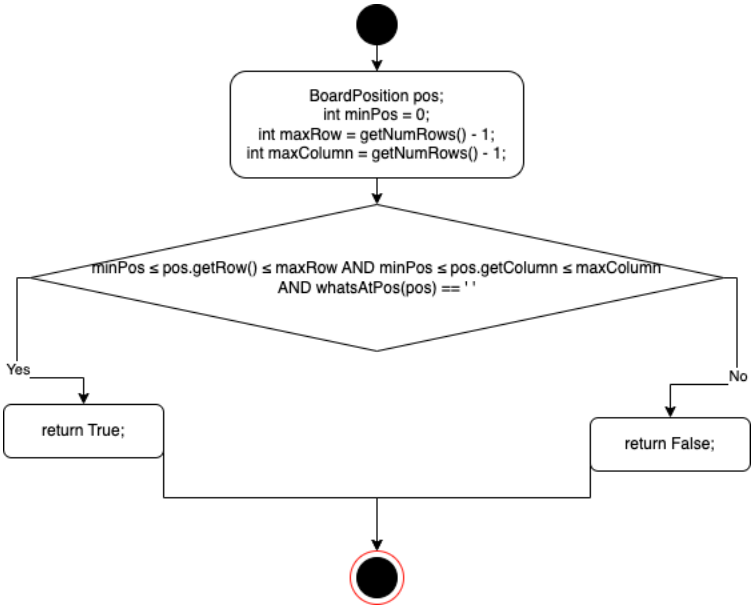
Activity diagrams

Class 3: IGameBoard

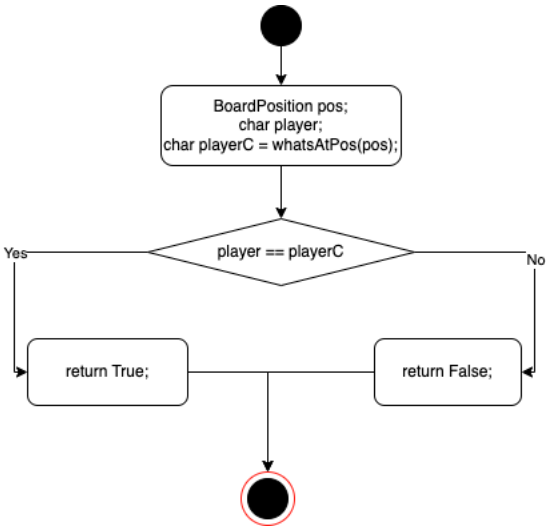
Class diagram

Activity diagrams

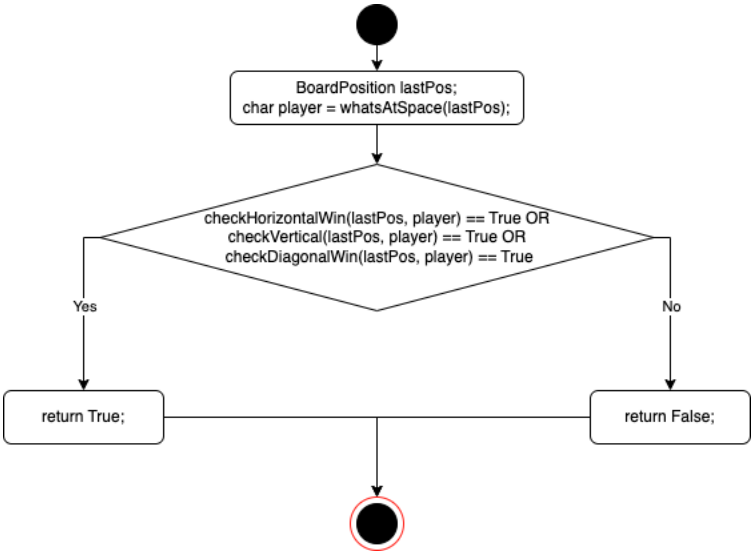
checkSpace(BoardPosition pos):



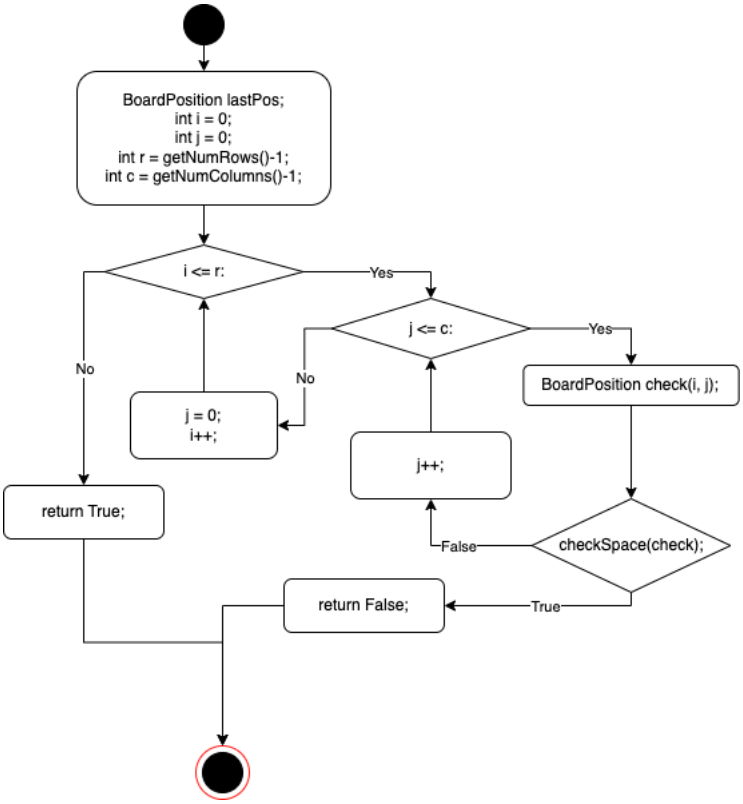
isPlayerAtPos(BoardPosition pos, char player):



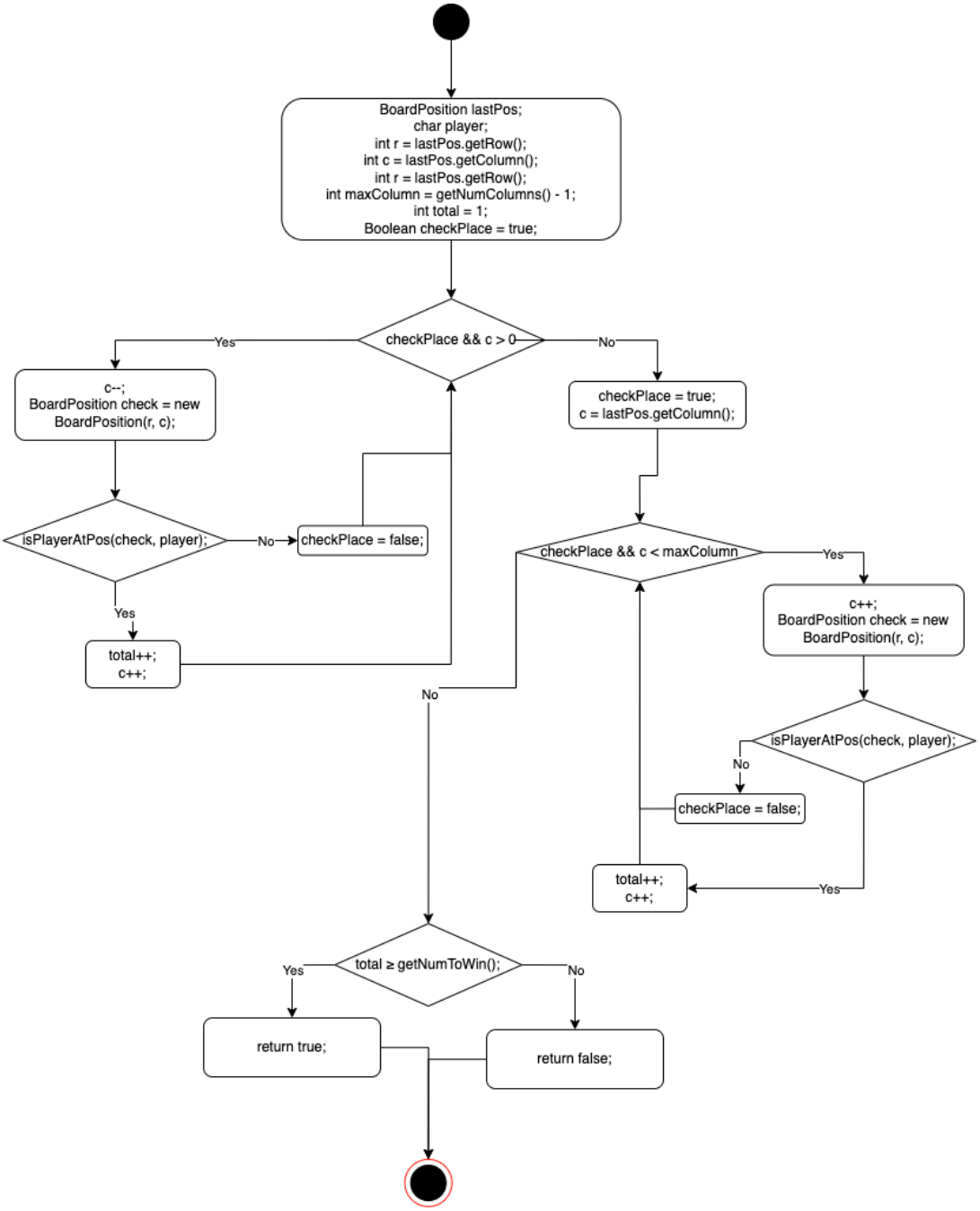
checkForWinner(BoardPosition lastPos):



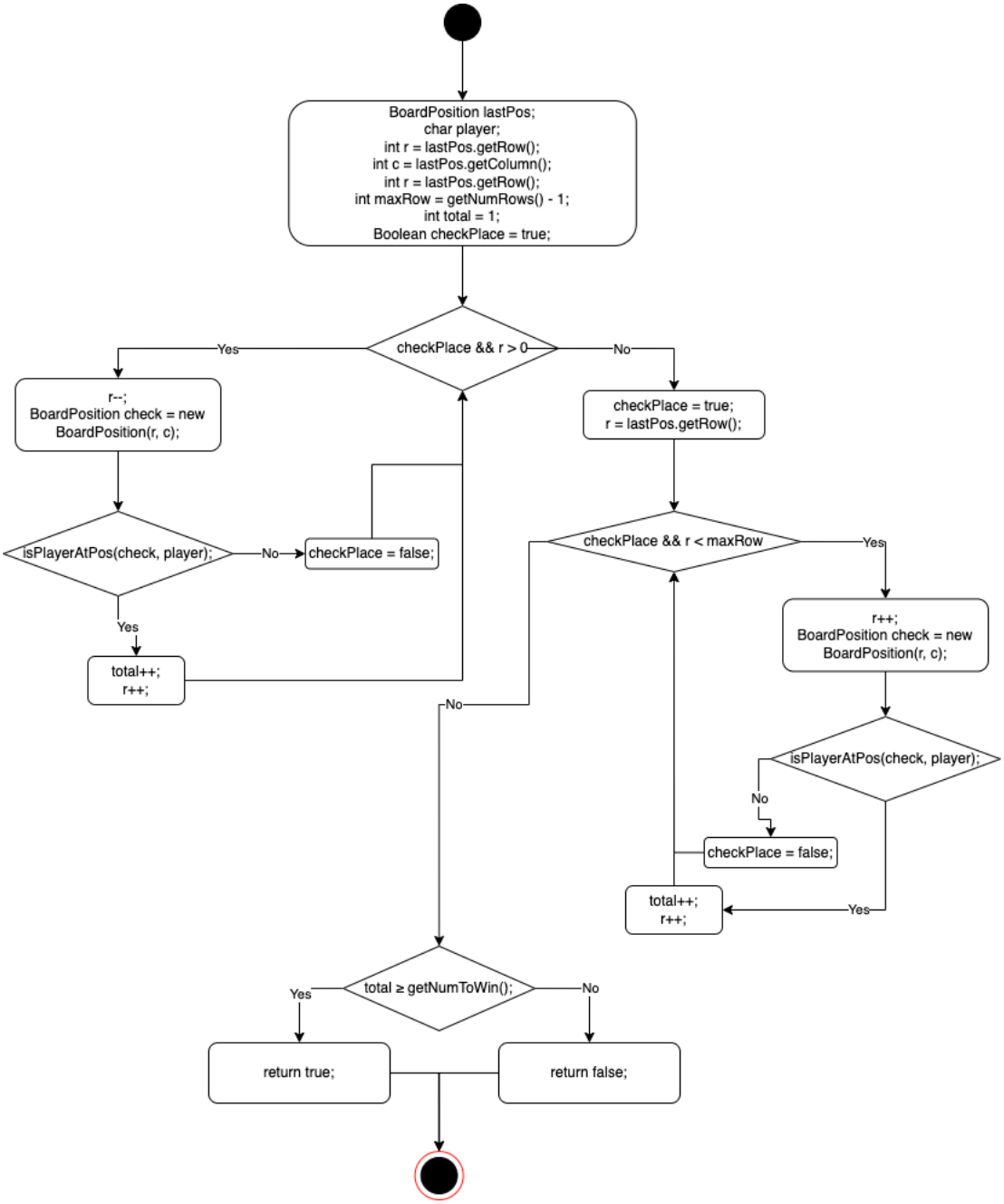
checkForDraw():



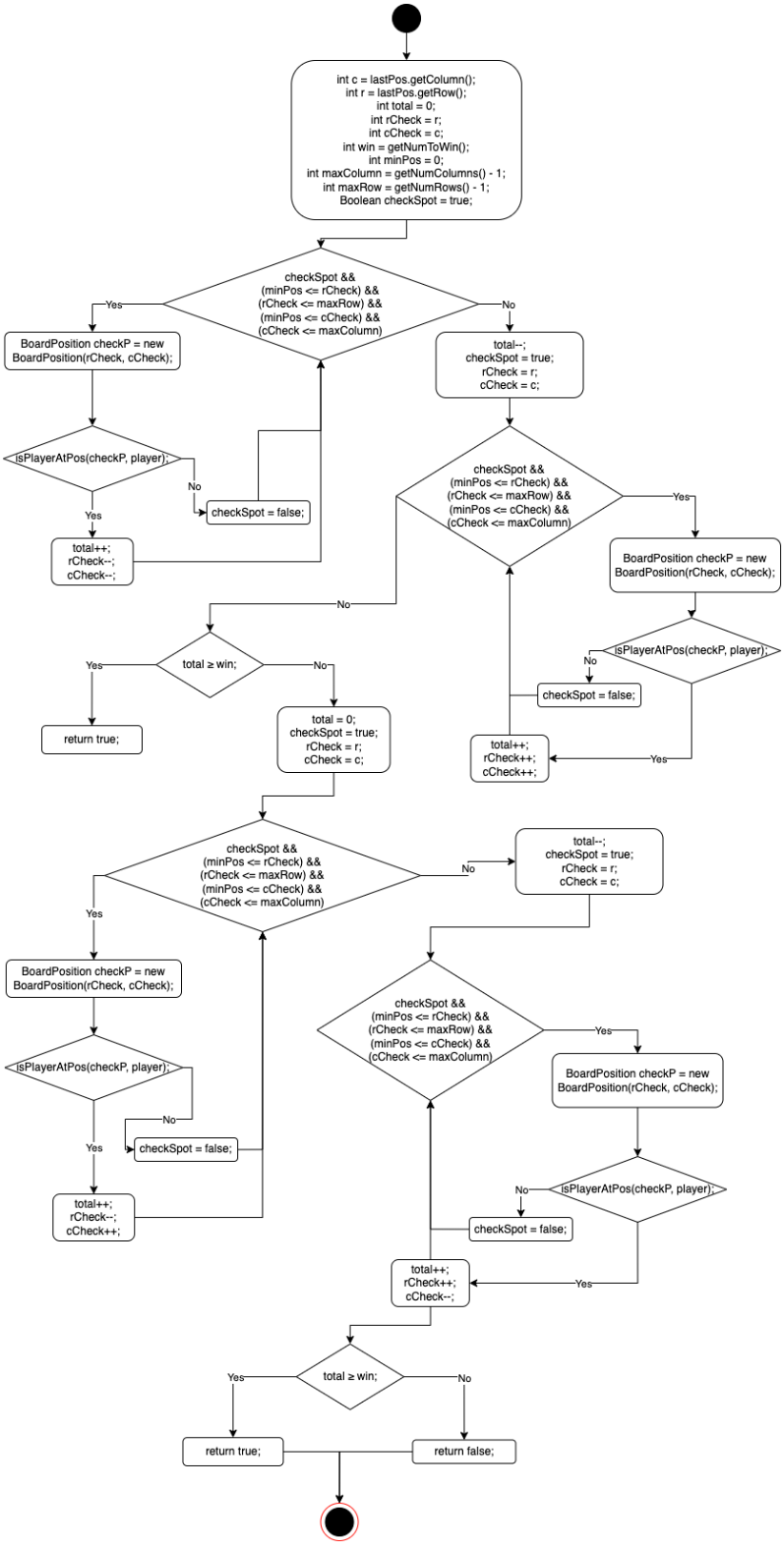
checkHorizontalWin(BoardPosition lastPos, char player):



checkVerticalWin(BoardPosition lastPos, char player):

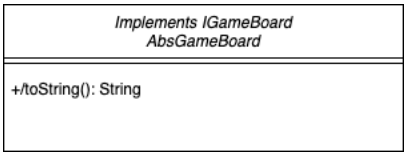


checkDiagonalWin(BoardPosition lastPos, char player):



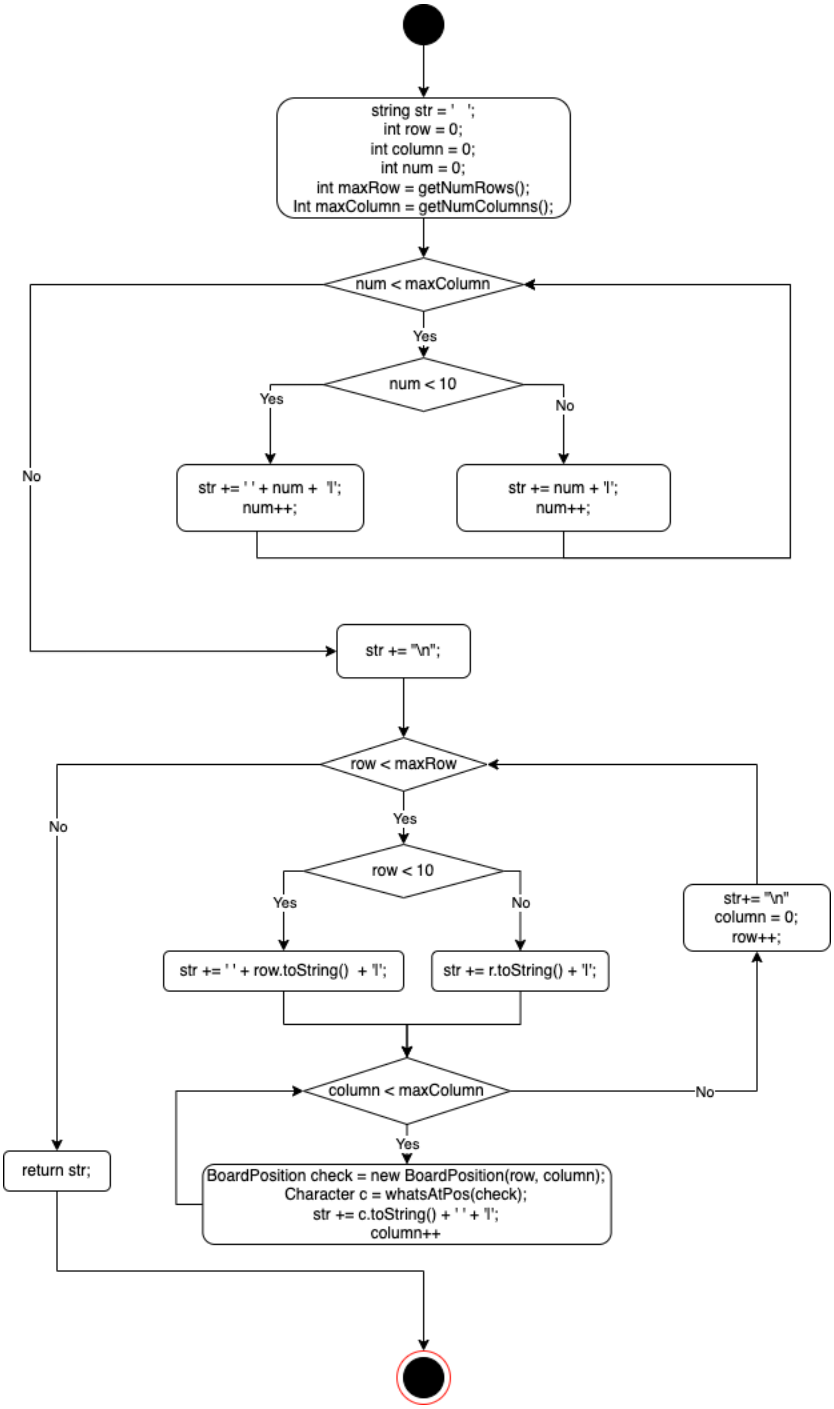
Class 4: AbsGameBoard

Class diagram



Activity diagrams

toString():


string str = ' ';
int row = 0;
int column = 0;
int num = 0;
int maxRow = getNumRows();
Int maxColumn = getNumColumns();
num < maxColumn
Yes
No
num < 10
Yes
No
str += ' ' + num + 'l';
num++;
str += num + 'l';
num++;
str += "\n";
row < maxRow
Yes
No
row < 10
Yes
No
str += ' ' + row.toString() + 'l';
str += r.toString() + 'l';
str+= "\n"
column = 0;
row++;
column < maxColumn
Yes
No
BoardPosition check = new BoardPosition(row, column);
Character c = whatsAtPos(check);
str += c.toString() + ' ' + 'l';
column++
return str;

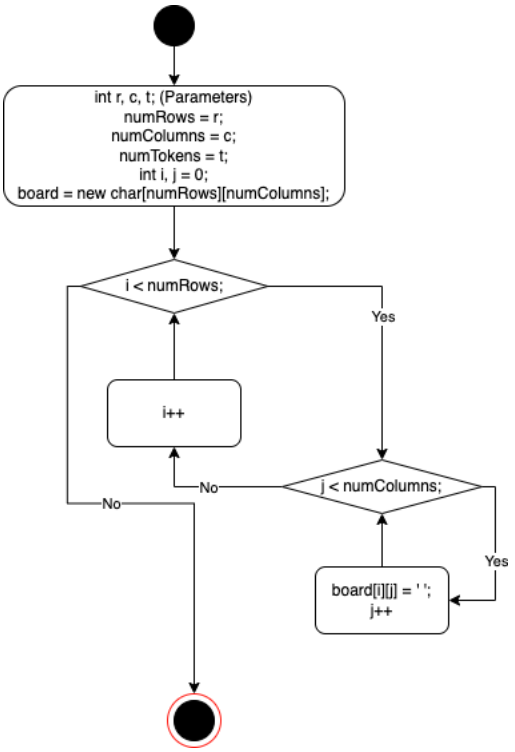
Class 5: GameBoard

Class diagram

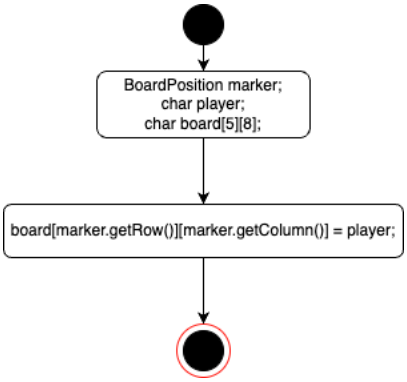
Extends AbsGameBoard Implements IGameBoard *GameBoard*
-board: char [*] [*] -numRows: int [1] -numColumns: int [1] -numTokens: int [1]
+GameBoard(int, int, int) +placeMarker(BoardPosition, char): void +whatsAtPos(BoardPosition): char +getNumRows(): int +getNumColumns(): int +getNumToWin(): int

Activity diagrams

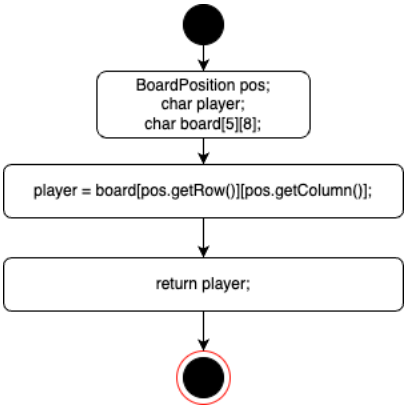
GameBoard(int numRows, int numColumns, int numTokens):



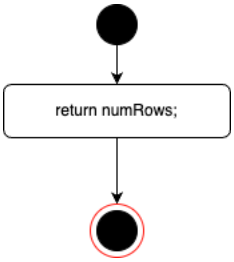
placeMarker(BoardPosition marker, char player):



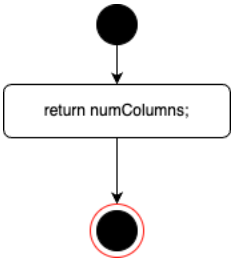
whatsAtPos(BoardPosition pos):



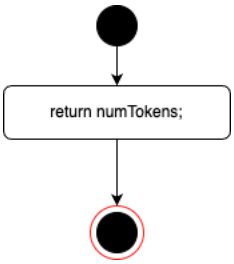
getNumRows():



getNumColumns():



getNumToWin():



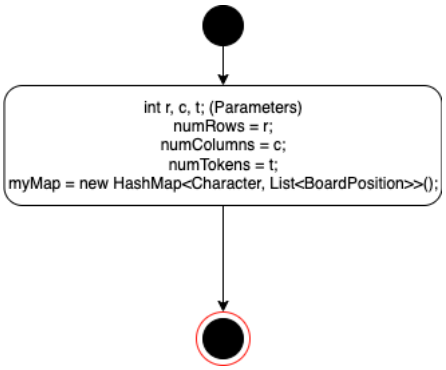
Class 6: GameBoardMem

Class diagram

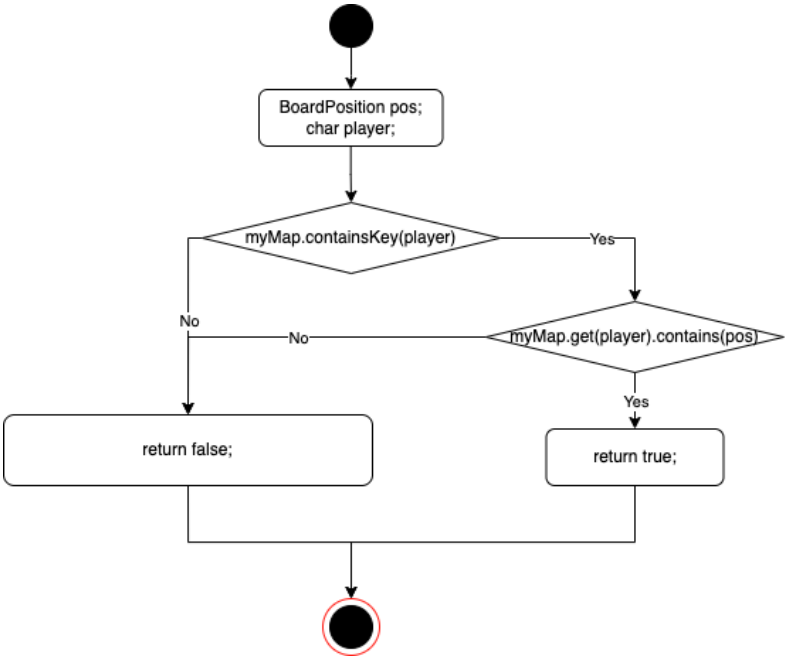
Extends AbsGameBoard Implements IGameBoard *GameBoardMem*
-myMap: HashMap<Character, List<BoardPosition>> [1] -numRows: int [1] -numColumns: int [1] -numTokens: int [1]
+GameBoardMem(int, int, int) +placeMarker(BoardPosition, char): void +whatsAtPos(BoardPosition): char +getNumRows(): int +getNumColumns(): int +getNumToWin(): int +isPlayerAtPos(BoardPosition, char): bool

Activity diagrams

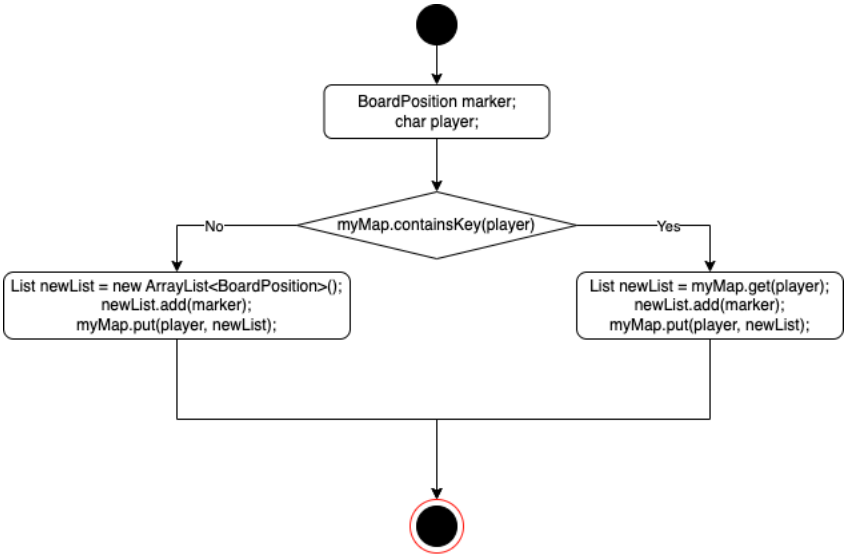
GameBoardMem(int numRows, int numColumns, int numTokens):



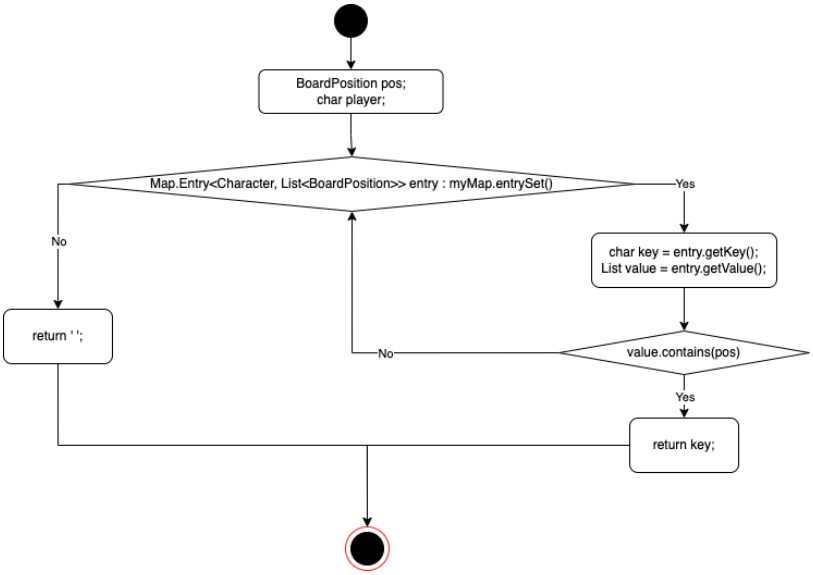
isPlayerAtPos(BoardPosition pos, char player):



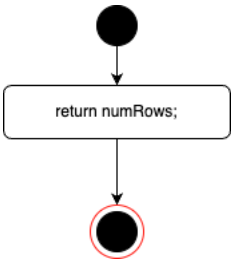
placeMarker(BoardPosition marker, char player):



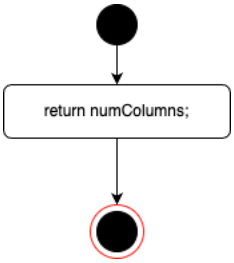
whatsAtPos(BoardPosition pos):



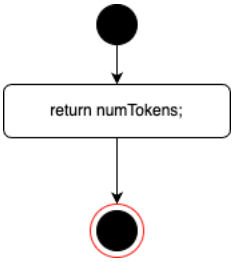
getNumRows():



getNumColumns():



getNumToWin():



Test Cases

Details in Project 4.

Constructor(numRows, numColumns, numTokens)

<table>
<tr>
<td>Input:
numRows = 3
numColumns = 3
numTokens = 3</td>
<td>Output:
GameBoard/GameBoardMem has size initialized to 3x3 and numTokens equaling 3. All elements are ‘ ‘.

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></td>
<td>Reason: This test case is unique because it ensures that the minimum size and number of tokens needed to win can be used.

Function Name: 3</td>
</tr>
<tr>
<td>Input:
numRows = 100
numColumns = 100
numTokens = 25</td>
<td>Output:
GameBoard/GameBoardMem has size initialized to 100x100 and numTokens equaling 25. All elements are ‘ ‘.</td>
<td>Reason: This test case is unique because it ensures that the maximum size and maximum number of tokens needed to win can be used.

Function Name:
testConstructor_maximum_size</td>
</tr>
<tr>
<td>Input:
numRows = 28
numColumns = 20
numTokens = 5</td>
<td>Output:
GameBoard/GameBoardMem with size initialized to 28x20 and numTokens equaling 5. All elements are ‘ ‘.</td>
<td>Reason: This test case is unique because it ensures that the game board can be initialized to a size within the size range, instead of only the boundaries.

Function Name:
testConstructor_reg_size</td>
</tr>
</table>

boolean checkSpace(BoardPosition pos)

<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></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>X</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>pos.getRow = 2
pos.getCol = 2</td>
<td>Output:
False

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it ensures that checkSpace correctly identifies a taken space and returns false.

Function Name:
testCheckSpace_unavailable_position</td>
</tr>
<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></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>X</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>pos.getRow = 2
pos.getCol = 1</td>
<td>Output:
True

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it ensures that checkSpace correctly identifies an empty space and returns true.

Function Name:
testCheckSpace_available_position</td>
</tr>
<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></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>X</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>pos.getRow = 5
pos.getCol = 5</td>
<td>Output:
False

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it tests to see if checkSpace correctly identifies a space that is out of bounds and returns false.

Function Name:
testCheckSpace_out_of_bounds_position</td>
</tr>
</table>

checkHorizontalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr>
<tr><td>0</td><td>O</td><td>O</td><td>O</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></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow = 0
pos.getCol = 0
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the last O of the victory was placed in the top left corner of the horizontal line, testing to see if checkHorizontalWin checks the minimum boundaries of the board for tokens.

Function Name:
testCheckHorizontalWin_last_marker_minimum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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>X</td><td></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>O</td><td>O</td><td>O</td></tr>
</table>
pos.getRow = 3
pos.getCol = 3
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the last O of the victory was placed in the right bottom of the horizontal line, testing to see if checkHorizontalWin checks the maximum boundaries.

Function Name:
testCheckHorizontalWin_last_marker_maximum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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>X</td><td></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>O</td><td>O</td></tr>
</table>
pos.getRow = 3
pos.getCol = 3
player = ‘O’</td>
<td>Output:
false

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it is a case where there is no win situation, and it tests to see if checkHorizontalWin correctly returns false.

Function Name:
testCheckHorizontalWin_last_marker_no_win</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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></td><td>X</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>X</td><td>O</td><td>O</td><td>O</td><td>X</td></tr>
</table>
pos.getRow = 3
pos.getCol = 2
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the O is placed in the center of the row, which makes the checkHorizontal win check to the left and the right, instead of down one row.

Function Name:
testCheckHorizontalWin_last_marker_middle_marker</td>
</tr>
</table>

checkVerticalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>O</td><td></td><td></td><td></td></tr><tr><td>1</td><td>O</td><td>X</td><td></td><td></td></tr><tr><td>2</td><td>O</td><td></td><td>X</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>pos.getRow = 0
pos.getCol = 0
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the last O of the victory was placed in the top left corner of the vertical line, testing to see if checkVerticalWin checks the minimum boundaries of the board for tokens.

Function Name:
testCheckVerticalWin_last_marker_minimum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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>X</td><td></td><td>O</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>O</td></tr></table>pos.getRow = 3
pos.getCol = 3
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the last O of the victory was placed in the right bottom of the vertical line, testing to see if checkVerticalWin checks the maximum boundaries.

Function Name:
testCheckVerticalWin_last_marker_maximum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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>X</td><td></td><td>O</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>pos.getRow = 2
pos.getCol = 3
player = ‘O’</td>
<td>Output:
false

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it is a case where there is no win situation, and it tests to see if checkVerticalWin correctly returns false.

Function Name:
testCheckVerticalWin_last_marker_no_win</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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></td><td></td><td></td><td>O</td><td></td></tr><tr><td>2</td><td></td><td></td><td>X</td><td>O</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td>X</td><td></td></tr></table>pos.getRow = 1
pos.getCol = 3
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the O is placed in the middle of the winning column, which makes the checkVerticalWin check to the top and the bottom, instead of only up or only down the column.

Function Name:
testCheckVerticalWin_last_marker_middle_marker</td>
</tr>
</table>

checkDiagonalWin(BoardPosition lastPos, char player)

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<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>O</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td>O</td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td>X</td><td>O</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td>X</td><td></td></tr></table>pos.getRow = 2
pos.getCol = 3
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case checks if the checkDiagonalWin can detect a diagonal going from top left to bottom right.

Function Name:
testCheckDiagonalWin_top_left_diagonal</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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></td><td></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></td><td></td><td>X</td><td></td></tr></table>pos.getRow = 0
pos.getCol = 3
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks if the checkDiagonalWin can detect a diagonal going from top right to bottom left (the opposite diagonal).

Function Name:
testCheckDiagonalWin_top_right_diagonal</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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></td><td></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></td><td></td><td>X</td><td></td></tr></table>pos.getRow = 1
pos.getCol = 2
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks if the checkDiagonalWin can detect a diagonal going from top right to bottom left with the last marker placed in the middle, which means it has to check both directions of the diagonal.

Function Name:
testCheckDiagonalWin_top_right_diagonal_middle_marker</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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>O</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td>O</td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td>X</td><td>O</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td>X</td><td></td></tr></table>pos.getRow = 1
pos.getCol = 2
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks if the checkDiagonalWin can detect a diagonal going from top right to bottom left (the opposite diagonal) when the last marker placed is in the middle, which means it must check both directions of the diagonal.

Function Name:
testCheckDiagonalWin_top_left_diagonal_middle_marker</td>
</tr>
</table>

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<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>O</td><td>X</td><td></td><td></td></tr>
<tr><td>2</td><td>O</td><td></td><td>X</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow = 0
pos.getCol = 0
player = ‘X’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the last X of the victory was placed in the top left corner of the diagonal line, testing to see if checkDiagonalWin checks the minimum boundaries of the board for tokens.

Function Name:
testCheckDiagonalWin_last_marker_minimum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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>X</td><td></td><td>O</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>X</td></tr>
</table>
pos.getRow = 3
pos.getCol = 3
player = ‘X’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the last X of the victory was placed in the right bottom of the diagonal line, testing to see if checkDiagonalWin checks the maximum boundaries.

Function Name:
testCheckDiagonalWin_last_marker_maximum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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>X</td><td></td><td>O</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></td><td>O</td></tr>
</table>
pos.getRow = 3
pos.getCol = 3
player = ‘O’</td>
<td>Output:
false

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it is a case where there is no win situation, and it tests to see if checkDiagonalWin correctly returns false.

Function Name:
testCheckDiagonalWin_last_marker_no_win</td>
</tr>
</table>

checkForDraw()

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<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>Output:
false

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks to see if checkForDraw can understand that the board is empty, which means it is not a draw.

Function Name:
testCheckForDraw_empty_board</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>X</td><td>O</td><td>X</td><td>X</td></tr><tr><td>3</td><td>X</td><td>O</td><td>X</td><td>O</td></tr></table></td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the board is maxed out, and there is no win condition, so the board is a draw.

Function Name:
testCheckForDraw_max_board</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>O</td><td>X</td><td>O</td><td>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>X</td><td>O</td><td>X</td><td>X</td></tr><tr><td>3</td><td>X</td><td>O</td><td>X</td><td></td></tr></table></td>
<td>Output:
False

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the board is maxed out, except for the bottom right corner, which is the maximum rows and maximum columns. This tests to see if checkForDraw checks the maximum edge of the board correctly.

Function Name:
testCheckForDraw_max_board_except_bottom_right</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td>X</td><td>O</td><td>X</td></tr><tr><td>1</td><td>O</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>X</td><td>O</td><td>X</td><td>X</td></tr><tr><td>3</td><td>X</td><td>O</td><td>X</td><td>O</td></tr></table></td>
<td>Output:
false

State of the board is unchanged</td>
<td>Reason:
This test case is unique because the board is maxed out, except for the top left corner, which is the minimum rows and minimum columns. This tests to see if checkForDraw checks the minimum edge of the board correctly.

Function Name:
testCheckForDraw_max_board_except_top_left</td>
</tr>
</table>

whatsAtPos(BoardPosition pos)

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<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>pos.getRow = 2
pos.getCol = 2</td>
<td>Output:
‘ ‘

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks to see if whatsAtPos can obtain an empty square in an empty board correctly.

Function Name:
testWhatsAtPos_empty_board</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>O</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</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>pos.getRow = 0
pos.getCol = 0</td>
<td>Output:
‘O‘

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks to see if whatsAtPos can obtain the character at the minimum row and column position (testing if it can access minimum bounds).

Function Name:
testWhatsAtPos_minimum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</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>O</td></tr></table>pos.getRow = 3
pos.getCol = 3</td>
<td>Output:
‘O‘

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks to see if whatsAtPos can obtain the character at the maximum row and column position (testing if it can access maximum bounds).

Function Name:
testWhatsAtPos_maximum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>O</td><td>X</td><td>O</td><td>O</td></tr><tr><td>1</td><td>X</td><td></td><td>X</td><td>X</td></tr><tr><td>2</td><td>O</td><td>X</td><td>O</td><td>O</td></tr><tr><td>3</td><td>O</td><td>X</td><td>O</td><td>O</td></tr></table>pos.getRow = 1
pos.getCol = 1</td>
<td>Output:
‘ ‘

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it has an empty position surrounded by characters. It checks to see if whatsAtPos is checking the correct position and returning the empty space, because it is the only space that is empty.

Function Name:
testWhatsAtPos_empty_square_surrounded</td>
</tr>
</table>

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<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>O</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow = 2
pos.getCol = 2</td>
<td>Output:
‘O‘

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it has a character surrounded by empty squares. It checks to see if whatsAtPos is checking the correct position and returning the character, because it is the only space with that character.

Function Name:
testWhatsAtPos_lone_character_square</td>
</tr>
</table>

isPlayerAtPos(BoardPosition pos, char player)

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<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></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td>O</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>pos.getRow = 2
pos.getCol = 2
player = ‘X’</td>
<td>Output:
false

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks to see if isPlayerAtPos checks for the specified character and not any character, since the character in the board is a different one.

Function Name:
testIsPlayerAtPos_wrong_char</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<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></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td>O</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table>pos.getRow = 0
pos.getCol = 0
player = ‘X’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks to see if isPlayerAtPos can correctly identify the character in the spot and return true.

Function Name:
testIsPlayerAtPos_correct_char</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>O</td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</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>pos.getRow = 0
pos.getCol = 0
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks to see if isPlayerAtPos can verify the character at the minimum row and column position (testing if it can access minimum bounds).

Function Name:
testIsPlayerAtPos_minimum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td>X</td><td></td><td></td></tr><tr><td>1</td><td>X</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>O</td></tr></table>pos.getRow = 3
pos.getCol = 3
player = ‘O’</td>
<td>Output:
true

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks to see if isPlayerAtPos can obtain the character at the maximum row and column position (testing if it can access maximum bounds).

Function Name:
testIsPlayerAtPos_maximum_corner</td>
</tr>
</table>

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<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>O</td><td></td></tr>
<tr><td>3</td><td></td><td></td><td></td><td></td></tr>
</table>
pos.getRow = 2
pos.getCol = 2
player = ‘X’</td>
<td>Output:
false

State of the board is unchanged</td>
<td>Reason:
This test case is unique because it checks to see if isPlayerAtPos checks for the a character that isn’t on the board

Function Name:
testIsPlayerAtPos_not_on_board</td>
</tr>
</table>

placeMarker(BoardPosition marker, char player)

<table>
<tr>
<td>Input:
State: (numTokens = 3)
<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></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>pos.getRow = 2
pos.getCol = 2
player = ‘O’</td>
<td>Output:
State: (numTokens = 3)
<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></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td>O</td><td></td></tr><tr><td>3</td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
This test case is unique because it checks to see if placeMarker can place the marker in the specified position.

Function Name:
testPlaceMarker_regular_spot</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>O</td><td>O</td><td>X</td><td></td></tr><tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr><tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td></tr></table>pos.getRow = 0
pos.getCol = 3
player = ‘O’</td>
<td>Output:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>O</td><td>O</td><td>X</td><td>O</td></tr><tr><td>1</td><td>X</td><td>X</td><td>O</td><td>O</td></tr><tr><td>2</td><td>O</td><td>O</td><td>X</td><td>X</td></tr><tr><td>3</td><td>X</td><td>X</td><td>O</td><td>O</td></tr></table></td>
<td>Reason:
This test case is unique because it checks to see if placeMarker can place a token at the winning spot of a board, and this is to make sure that placeMarker isn’t affected by the winner or loser of the game.

Function Name:
testPlaceMarker_winning_spot</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td>O</td><td></td><td></td></tr><tr><td>1</td><td>X</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>pos.getRow = 0
pos.getCol = 0
player = ‘O’</td>
<td>Output:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td>O</td><td>O</td><td></td><td></td></tr><tr><td>1</td><td>X</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>Reason:
This test case is unique because it checks to see if placeMarker can place the character at the minimum row and column position (testing if it can access minimum bounds).

Function Name:
testPlaceMarker_minimum_corner</td>
</tr>
<tr>
<td>Input:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td>O</td><td></td><td></td></tr><tr><td>1</td><td>X</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>pos.getRow = 3
pos.getCol = 3
player = ‘O’</td>
<td>Output:
State: (numTokens = 3)
<table><tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>0</td><td></td><td>O</td><td></td><td></td></tr><tr><td>1</td><td>X</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>O</td></tr></table></td>
<td>Reason:
This test case is unique because it checks to see if placeMarker can obtain the character at the maximum row and column position (testing if it can access maximum bounds).

Function Name:
testPlaceMarker_maximum_corner</td>
</tr>
</table>

Input:

State: (numTokens = 3)

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

pos.getRow = 0
pos.getCol = 3
player = 'X'

Output:

State: (numTokens = 3)

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

Reason:

This test case is unique because it checks to see if placeMarker can place a token at the draw spot of a board, and this is to make sure that placeMarker isn't affected by the state of the game.

Function Name:

testPlaceMarker_draw_spot