Grant Duffey
CPSC 2150
Kevin Plis

Requirements Analysis

Functional Requirements:

1. As a player, I can see the board because I need to know what the board looks like to play.
2. As a player, I can choose a column because that is how I place my marker.
3. As a player, I can choose an invalid location because I need to be told that column is unavailable.
4. As a player, I can choose the total number of players so I can play with my desired number of players.
5. As a player, I can choose an invalid total number of players so I can know that number of players is invalid.
6. As a player, I can choose my player character so I can have a token to play the game with.
7. As a player, I can choose an invalid player character so I can know that character is invalid.
8. As a player, I can choose the number of rows so the board will have the number of rows I want.
9. As a player, I can choose an invalid number of rows so I can know that number of rows is invalid.
10. As a player, I can choose the number of columns so the board will have the number of columns I want.
11. As a player, I can choose an invalid number of columns so I can know that number of columns is invalid.
12. As a player, I can choose the number of tokens needed in a row to win so the win condition can be the number I desire.
13. As a player, I can choose an invalid number of tokens needed in a row to win so I can know that number is invalid.

14. As a player, I can choose a Fast Game so I can have a game board that runs quickly.
15. As a player, I can choose a Memory Efficient Game so I can have a game board that is memory efficient.
16. As a player, I can make a move after my opponents do if they did not win so that we can continue to play the game.
17. As a player, I can meet the win condition number of tokens in a row horizontally so I can win the game.
18. As a player, I can meet the win condition number of tokens in a row vertically so I can win the game.
19. As a player, I can meet the win condition number of tokens in a row diagonally so I can win the game.
20. As a player, I can see that I won the game so I can know that I won the game.
21. As a player, I can see that I tied the game so I can know that the game ended in a tie.
22. As a player, upon finishing the game, I can choose whether or not I want to play again because I may want to play another game.

Non-Functional Requirements:

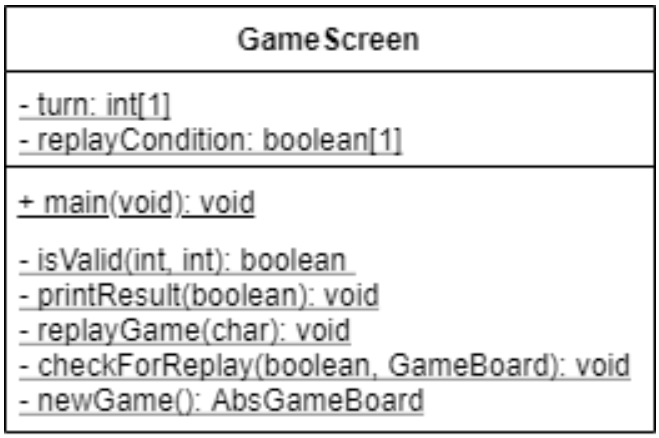
1. The program must be coded in Java
2. The program must run on UNIX
3. There must be a fast implementation of the game board
4. There must be a memory efficient implementation of game board
5. 0,0 is the bottom left of the board
6. The maximum number of players is 10
7. The minimum number of players is 2
8. The maximum row and column size is 100
9. The minimum row and column size is 3
10. The maximum number to win is 25
11. The minimum number to win is 3

12. Players cannot have the same token character
13. Player 1 must always go first
14. The number to win must be less than or equal to the number of rows or the number of columns

Design:

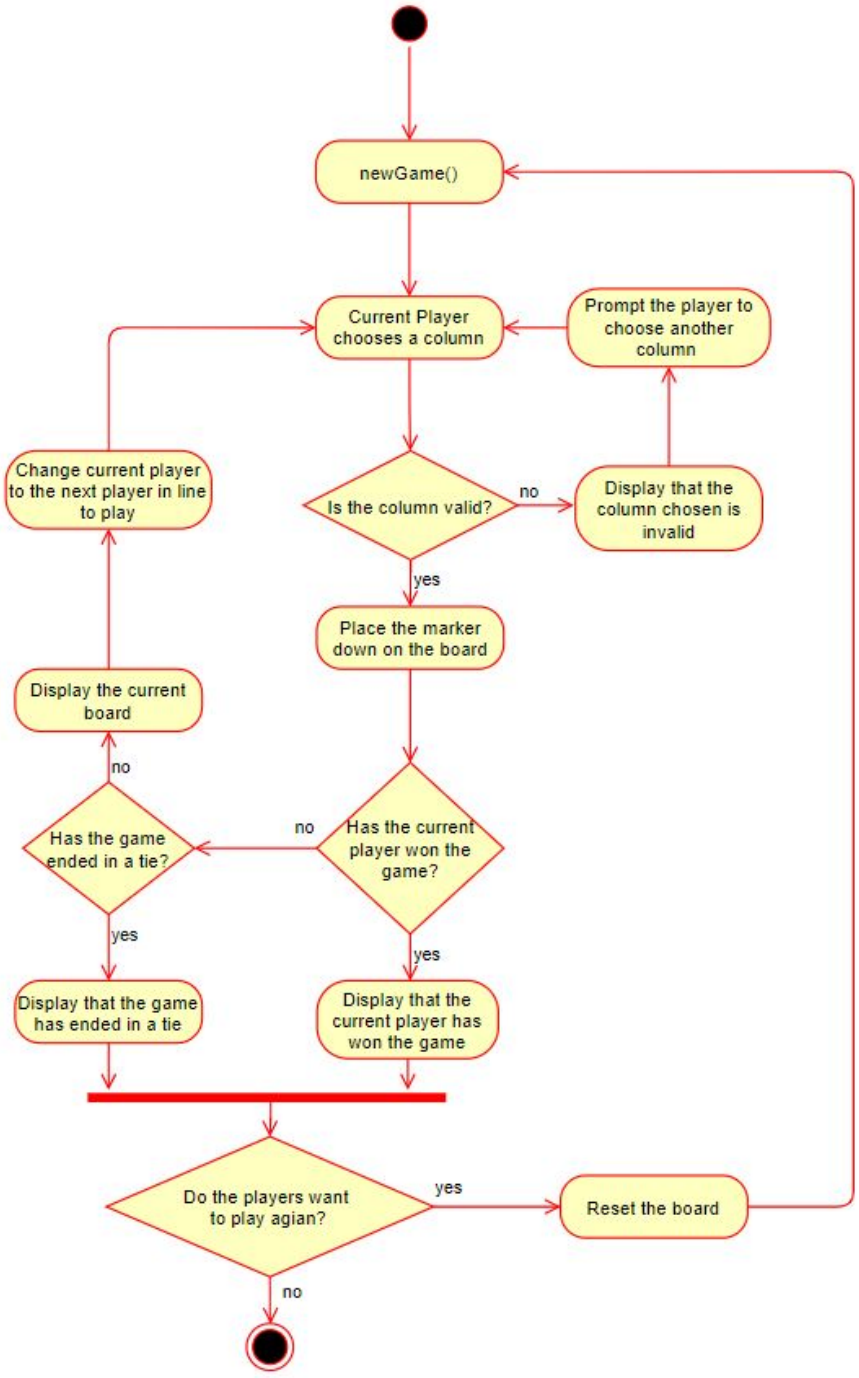
GameScreen.java

Class Diagram:

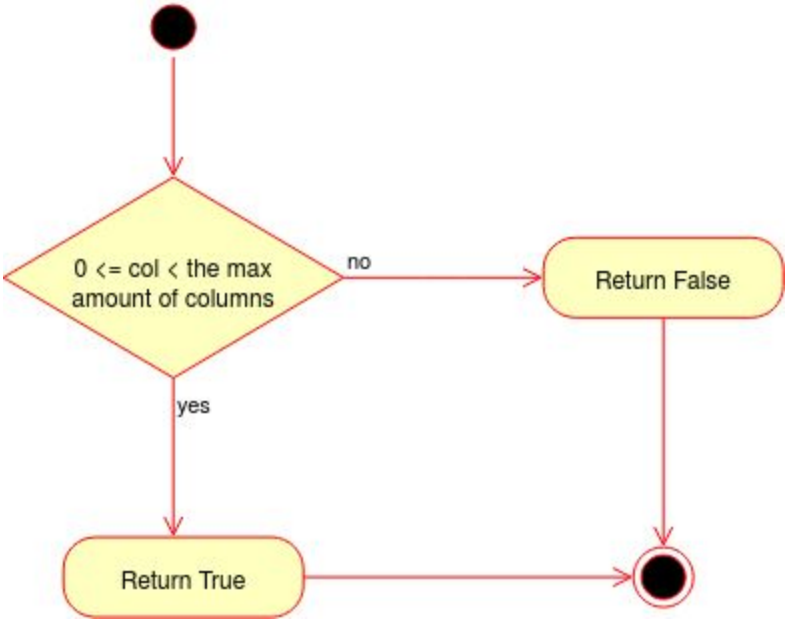


Activity Diagrams:

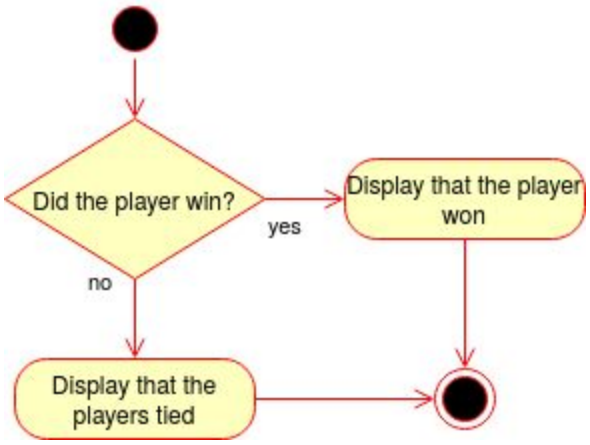
Main



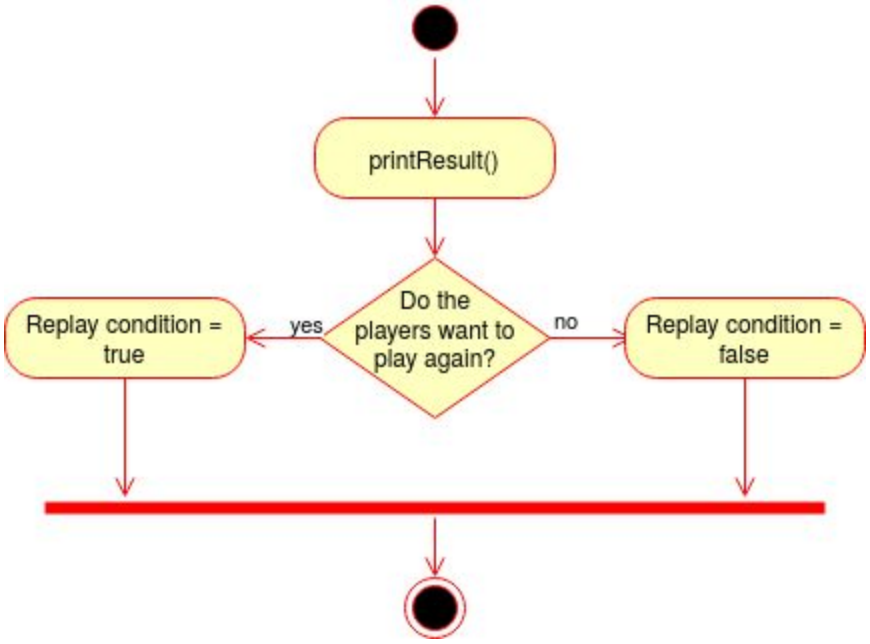
isValid



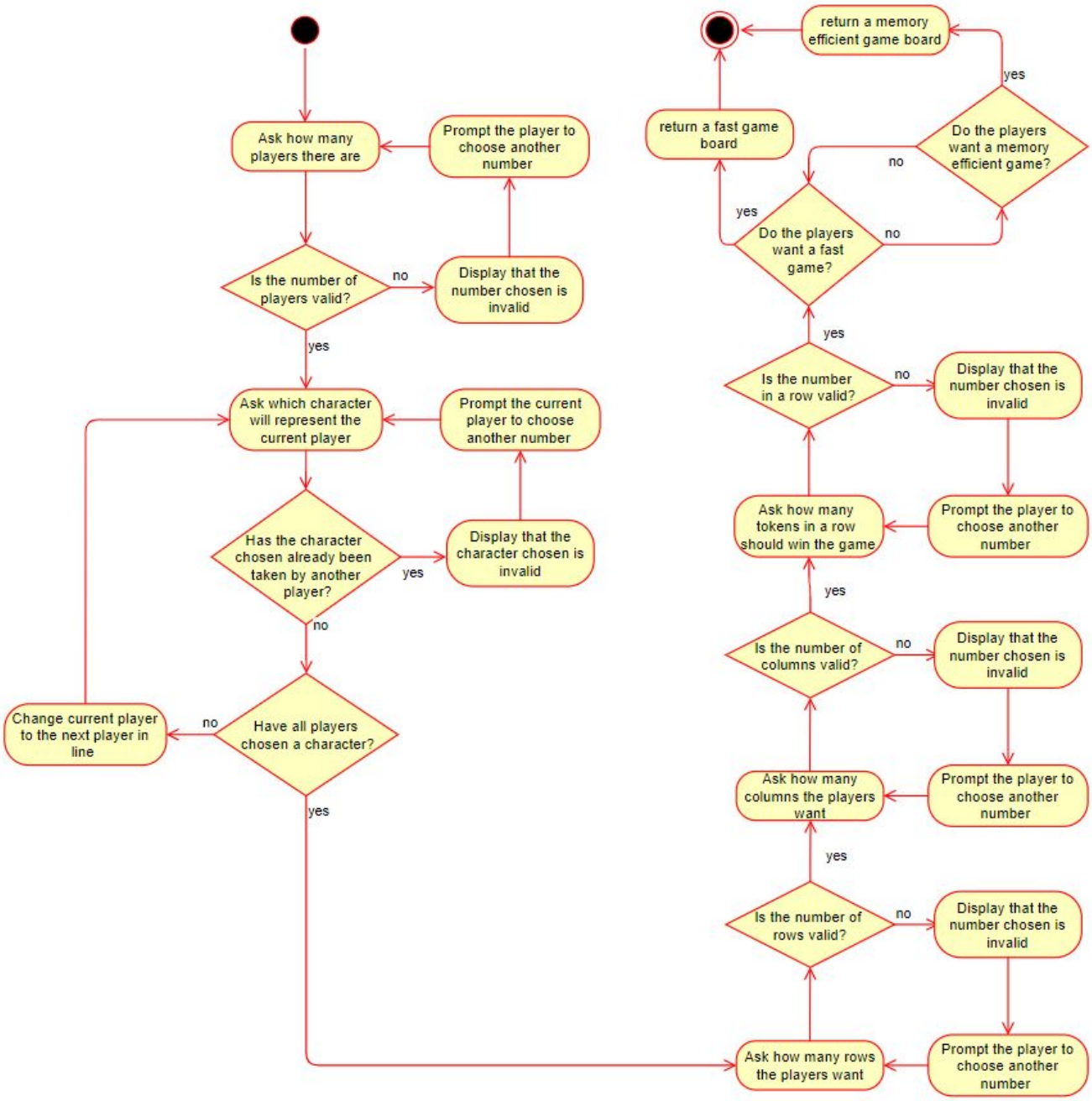
printResult



checkForReplay

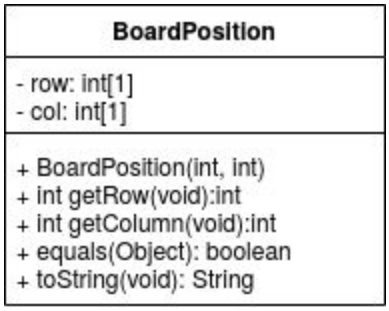


newGame


Ask how many players there are
Prompt the player to choose another number
Is the number of players valid?
no
Display that the number chosen is invalid
yes
Ask which character will represent the current player
Prompt the current player to choose another number
Has the character chosen already been taken by another player?
yes
Display that the character chosen is invalid
no
Have all players chosen a character?
no
Change current player to the next player in line
yes
return a memory efficient game board
yes
Do the players want a memory efficient game?
no
return a fast game board
yes
Do the players want a fast game?
no
yes
Is the number in a row valid?
no
Display that the number chosen is invalid
Ask how many tokens in a row should win the game
Prompt the player to choose another number
yes
Is the number of columns valid?
no
Display that the number chosen is invalid
Ask how many columns the players want
Prompt the player to choose another number
yes
Is the number of rows valid?
no
Display that the number chosen is invalid
Ask how many rows the players want
Prompt the player to choose another number

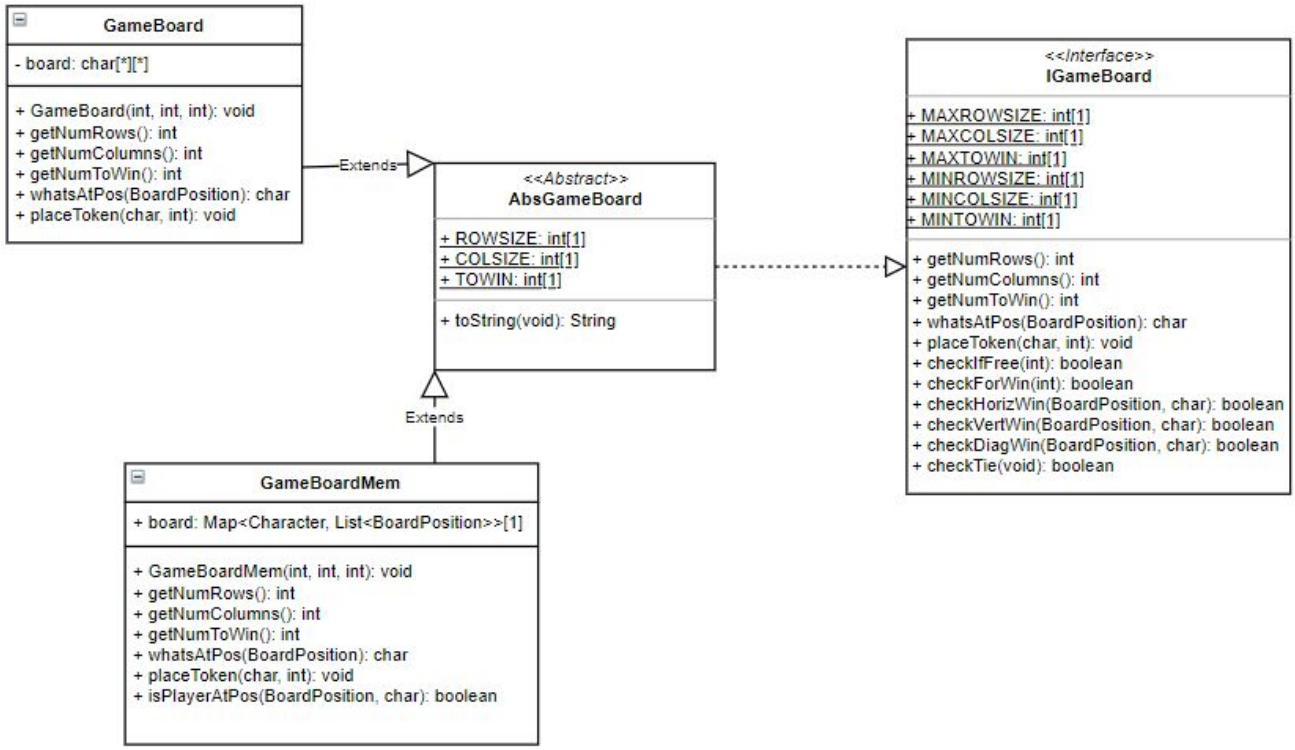
BoardPosition.java

Class Diagram:



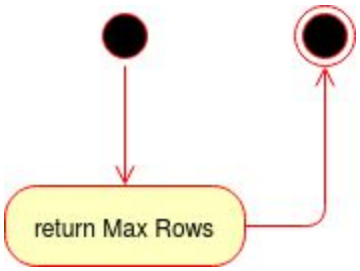
IGameBoard.java

Class Diagram:

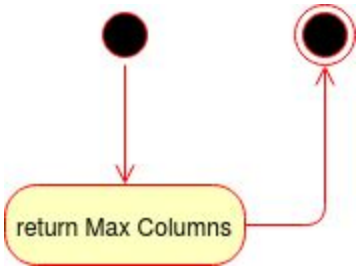


Activity Diagrams:

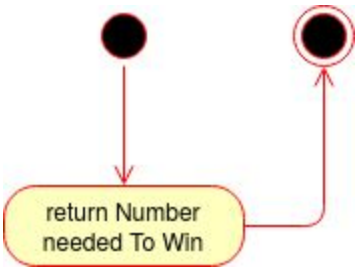
getNumRows



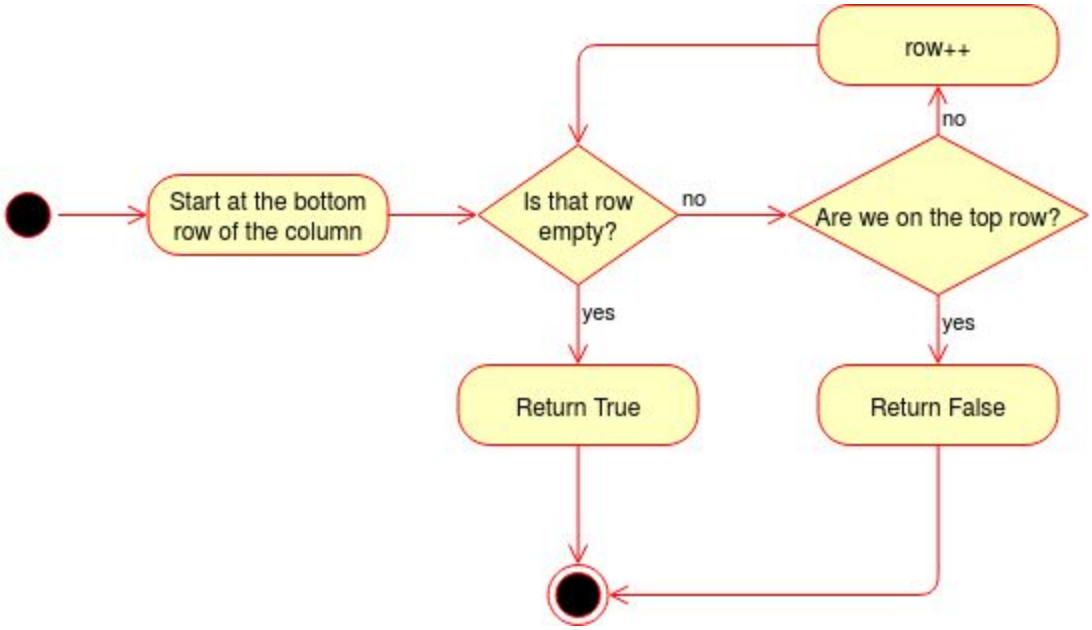
getNumColumns



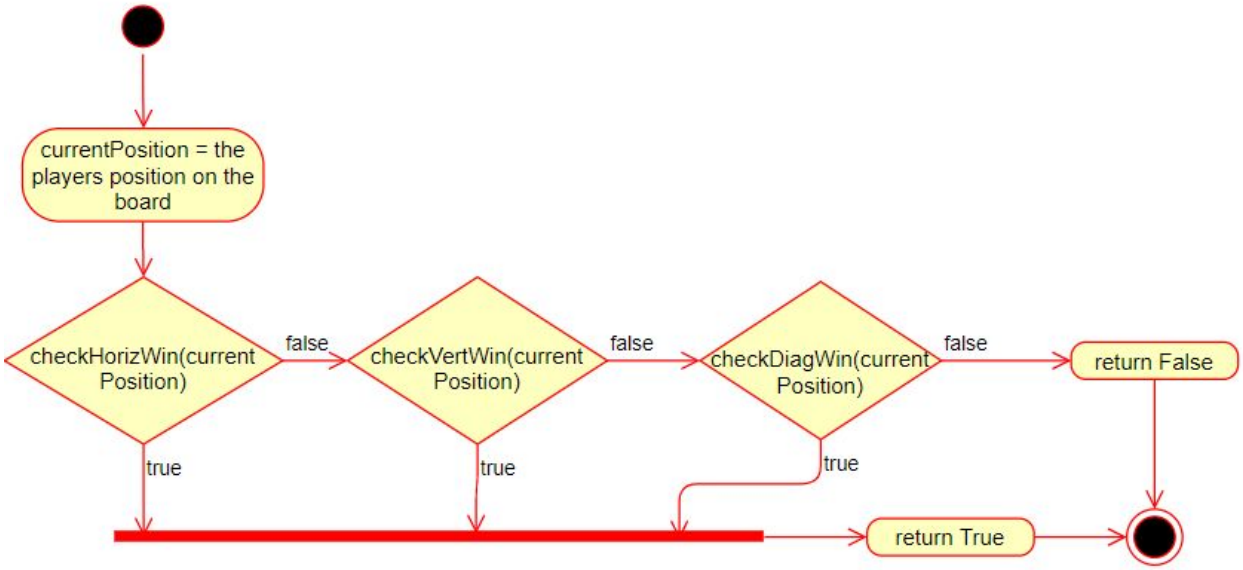
getNumToWin



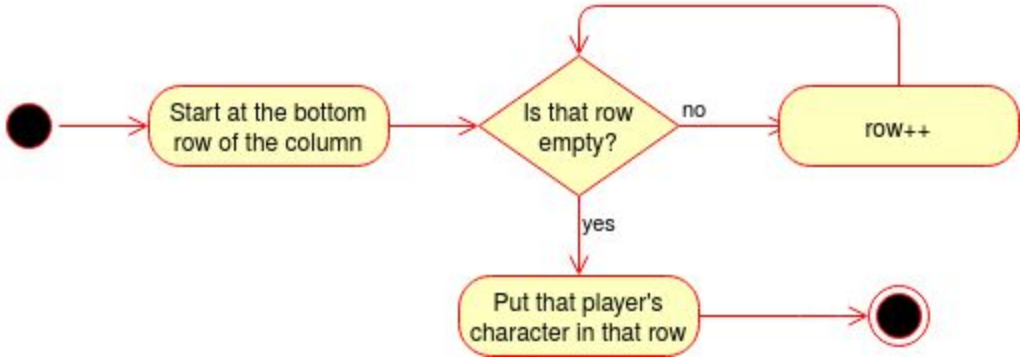
checkIfFree



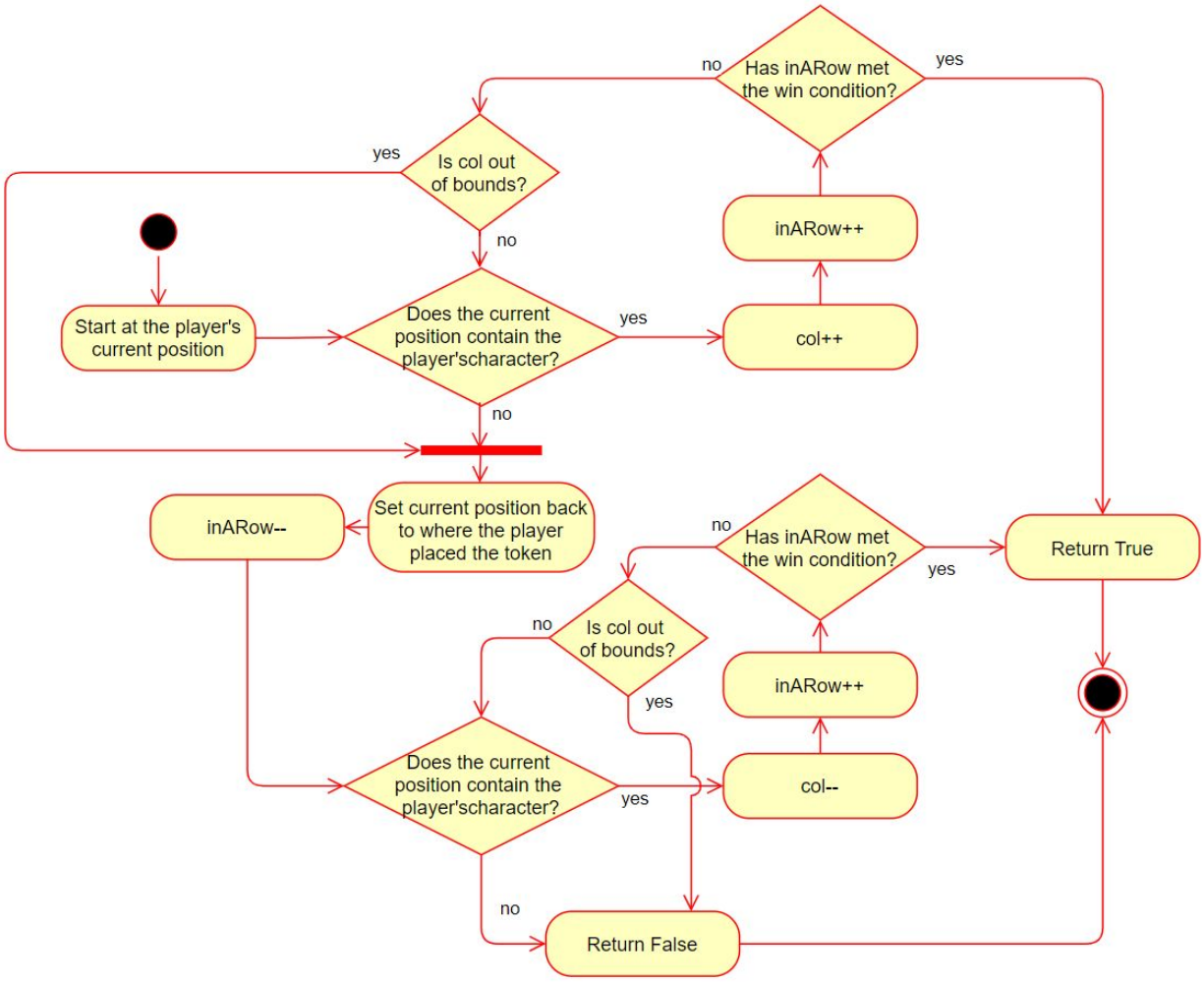
checkForWin



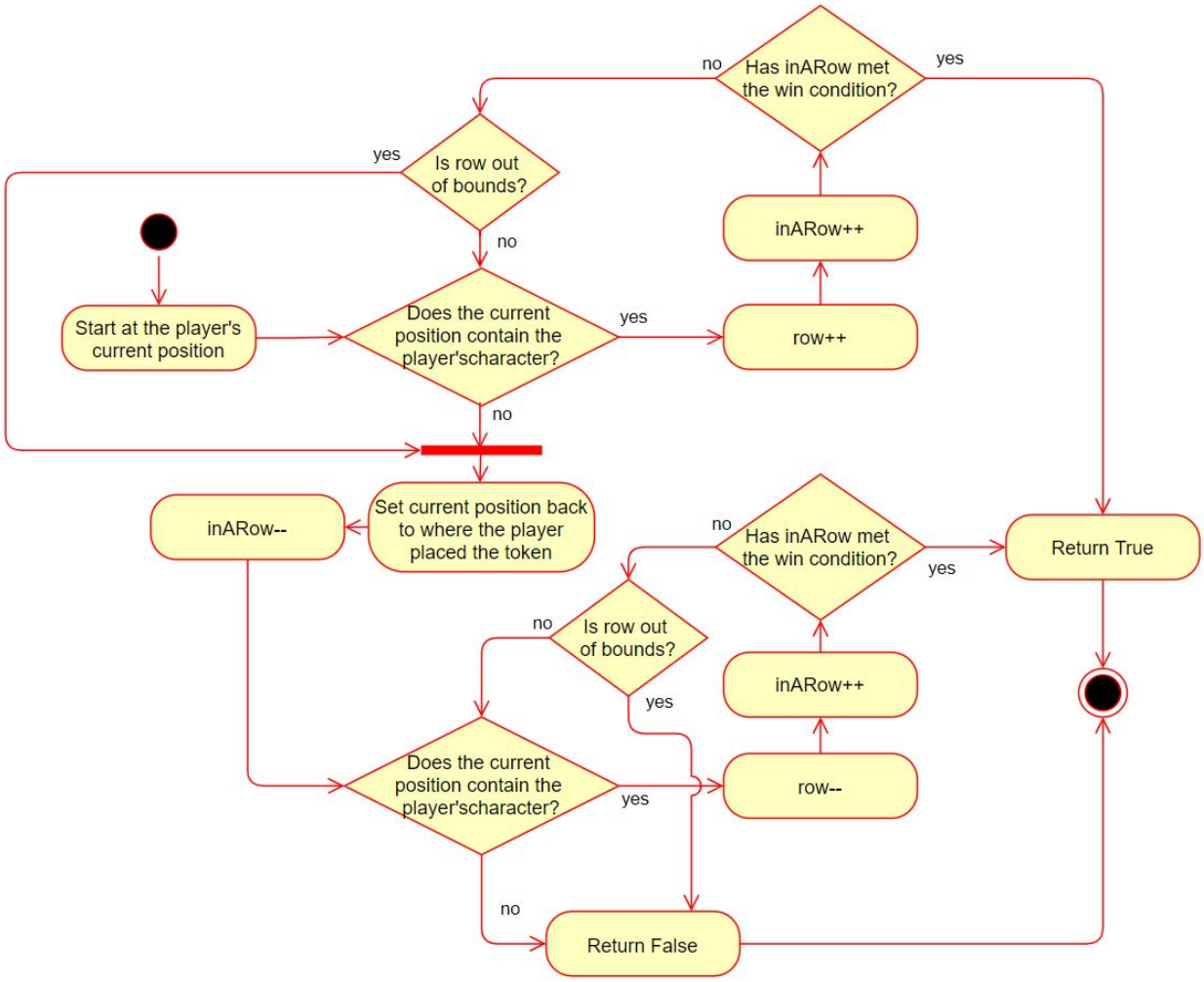
placeToken



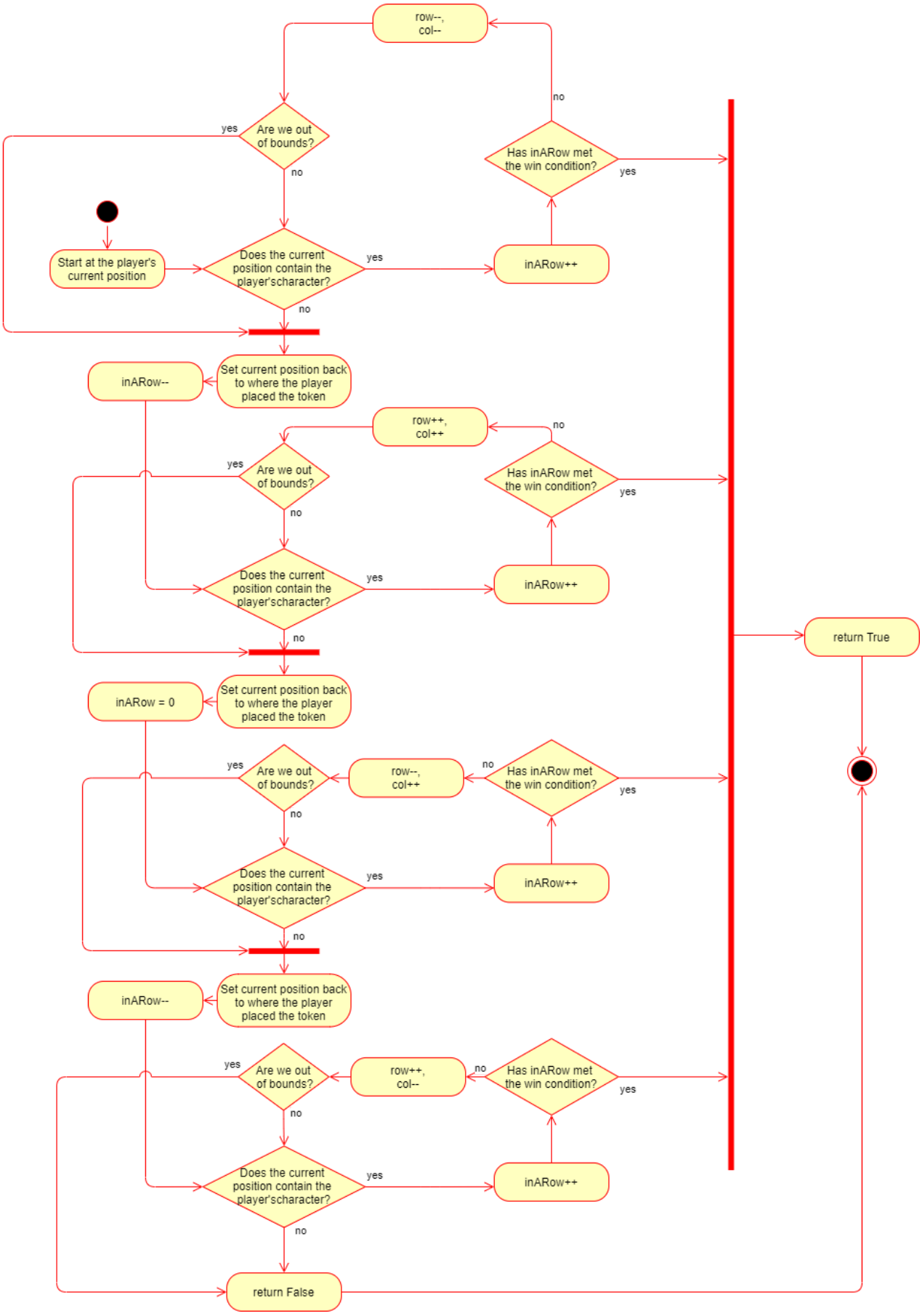
checkHorizWin



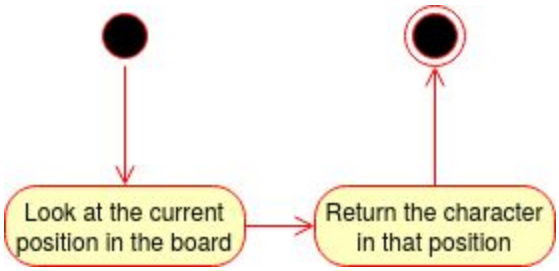
checkVertWin



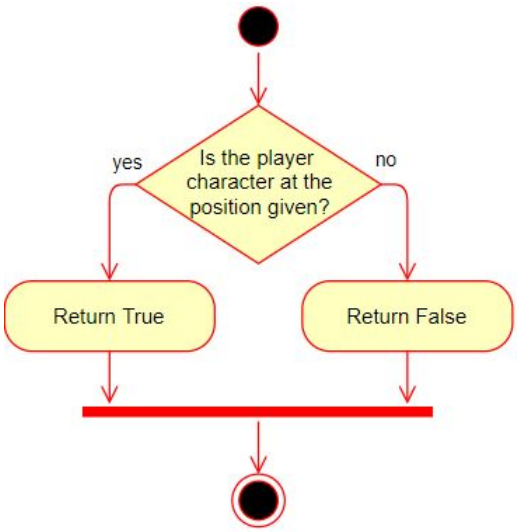
checkDiagWin



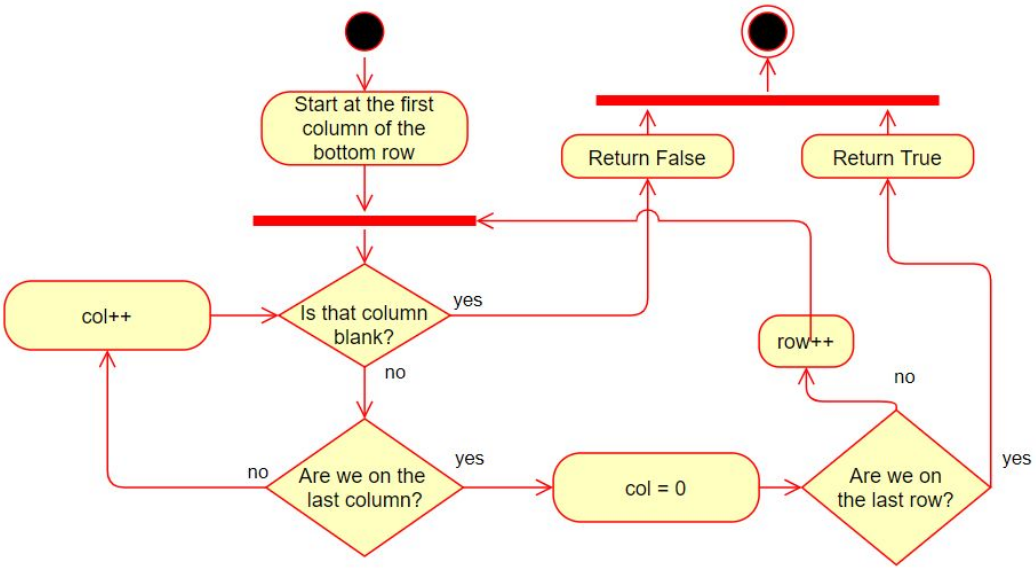
whatsAtPos



isPlayerAtPos

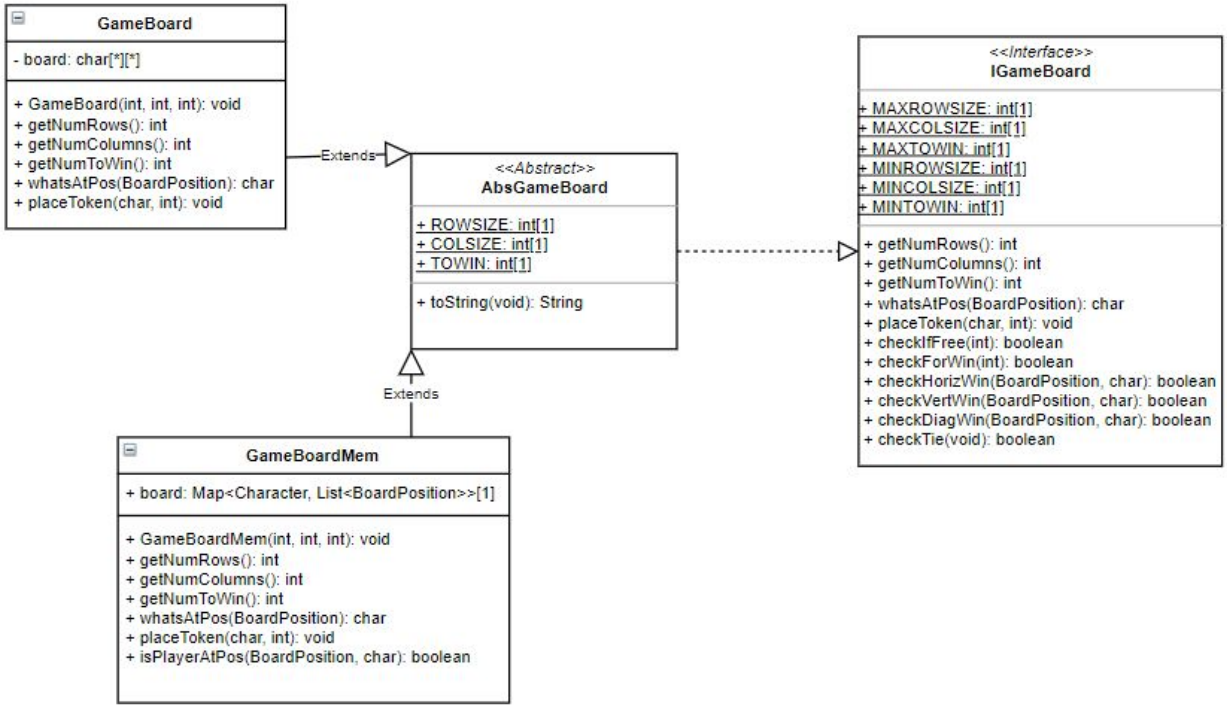


checkTie



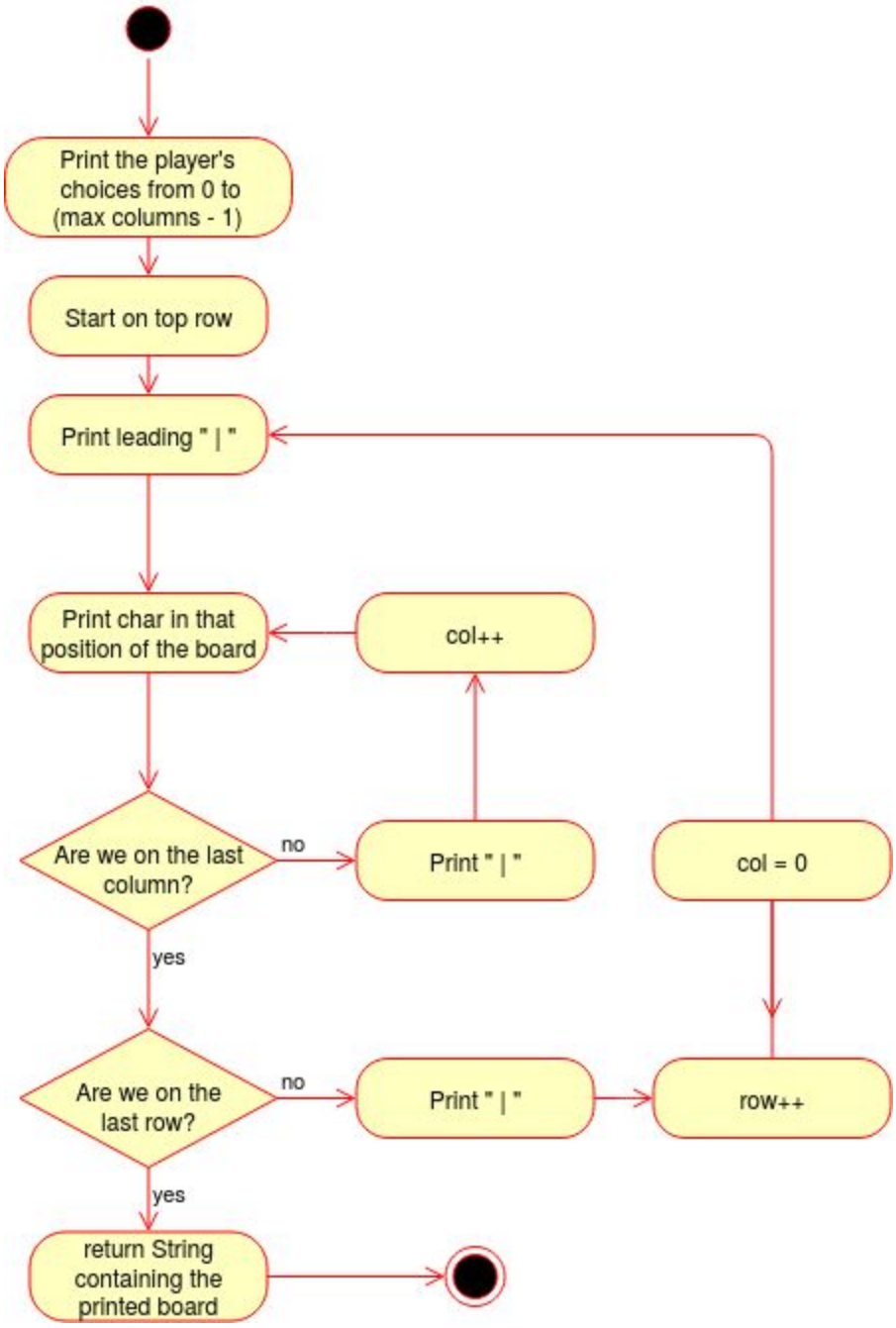
AbsGameBoard.java

Class Diagram:



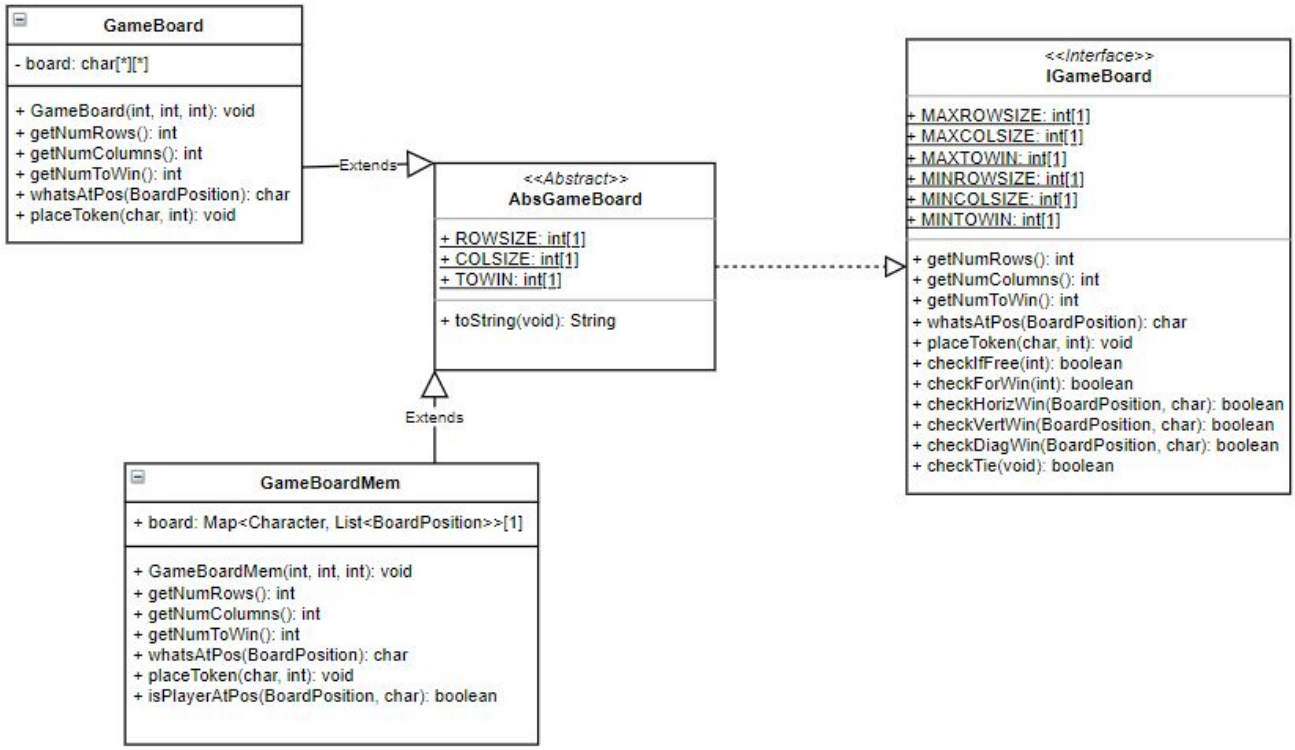
Activity Diagrams:

toString



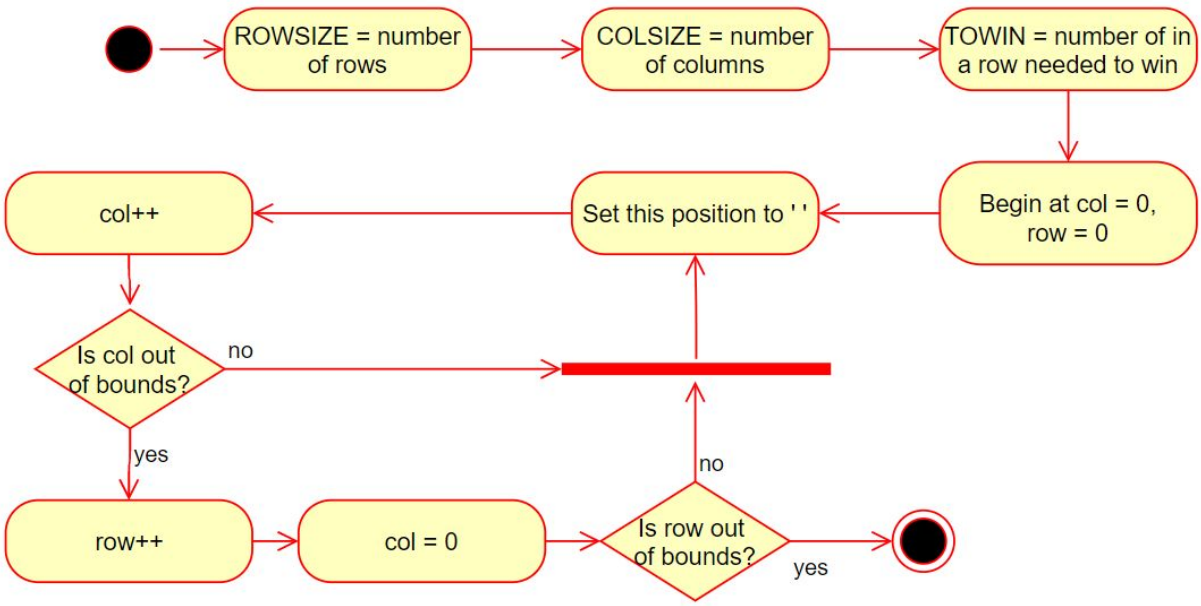
GameBoard.java

Class Diagram:



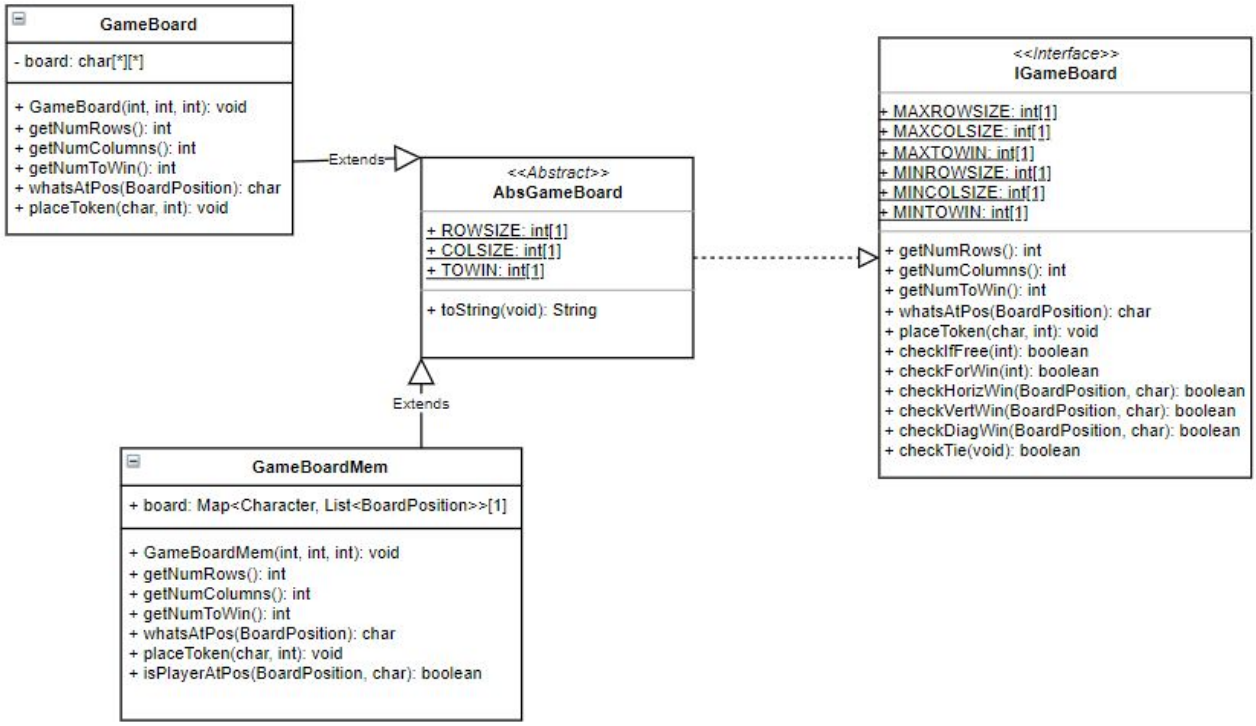
Activity Diagrams:

GameBoard



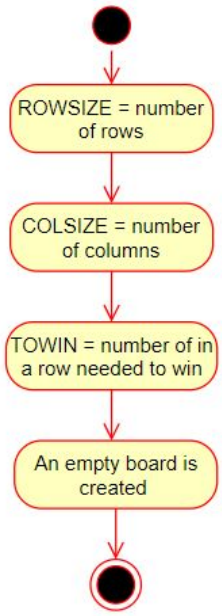
GameBoardMem.java

Class Diagram:



Activity Diagrams:

GameBoardMem



Testing:

GameBoard(int rowNum, int colNum, int winNum)

<table>
<tr>
<td>Input

State:
rowNum= 100;
colNum= 100;
winNum= 25;</td>
<td>Output

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>...</td><td>99</td></tr>
<tr><td>99</td><td></td><td></td><td></td><td></td></tr>
<tr><td>98</td><td></td><td></td><td></td><td></td></tr>
<tr><td>...</td><td></td><td></td><td></td><td></td></tr>
<tr><td>0</td><td></td><td></td><td></td><td></td></tr>
</table>
</td>
<td>Reason:
This test case is distinct because this creates the max sized board possible and the max number of tokens to win in a row

Function:
test_Constructor_Max</td>
</tr>
</table>

<table>
<tr>
<td>Input

State:
rowNum= 3;
colNum= 3;
winNum= 3;</td>
<td>Output

State:

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td></tr>
<tr><td>2</td><td></td><td></td><td></td></tr>
<tr><td>1</td><td></td><td></td><td></td></tr>
<tr><td>0</td><td></td><td></td><td></td></tr>
</table>
</td>
<td>Reason:
This test case is distinct because this creates the min sized board possible and the min number of tokens to win in a row

Function:
test_Constructor_Max</td>
</tr>
</table>

<table>
<tr>
<td>Input

State:
rowNum= 4;
colNum= 3;
winNum= 3;</td>
<td>Output

State:
<table><tr><td></td><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td></td><td></td><td></td></tr><tr><td>2</td><td></td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td></tr><tr><td>0</td><td></td><td></td><td></td></tr></table></td>
<td>Reason:
This test case is distinct because this creates a rectangular board instead of a square board

Function:
test_Constructor_Rect</td>
</tr>
</table>

boolean checkIfFree(int c)

<table>
<tr>
<td>Input
c = 1

State:
<table><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></td>
<td>Output

checkIfFree = true

State of the board is unchanged</td>
<td>Reason:
This test case is distinct because the board is completely empty

Function:
test_CheckIfFree_Empty</td>
</tr>
</table>

Input c = 4 State: 						Output checkIfFree = false State of the board is unchanged	Reason: This test case is distinct because the column being checked is completely full, and it is the maximum column we can check Function: test_CheckIfFree_Full
				X			
				O			
				X			
				O			
				X			

Input c = 0 State: 						Output checkIfFree = true State of the board is unchanged	Reason: This test case is distinct because the column being checked has tokens in it, but is not completely full. It also is checking the minimum column. Function: test_CheckIfFree_Partially_Full
O							
X							

boolean checkHorizWin(BoardPosition pos, char p)

Input	Output	Reason:
pos = <0,2> char = X State: (number to win = 4) | | | | | | | | | | | | | | | | | | | | | | | | | | X | | |	checkHorizWin = false State of the board is unchanged	This test case is distinct because one token is the minimum amount of tokens on the board that this function can be called for. Function: test_checkHorizWin_One_Token

Input	Output	Reason:
pos = <0,3> char = X State: (number to win = 4) | | | | | | | | | | | | | | | | | | | O | O | O | | | | X | X | X | X | |	checkHorizWin = true State of the board is unchanged	This test case is distinct because the last X was placed at the end of the string of 4 consecutive X's, so the function only needs to count X's on the left Function: test_checkHorizWin_Win_Last_Token_End

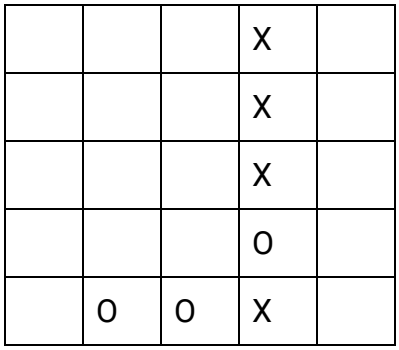
Input	Output	Reason:
pos = <0,1> char = X State: (number to win = 4) | | | | | | | | | | | | | | | | | | | O | | O | O | | | X | X | X | X | |	checkHorizWin = true State of the board is unchanged	This test case is distinct because the last X was placed in the middle of the string of 4 consecutive X's, so the function needs to count X's on the right and left Function: test_checkHorizWin_Win_Last_Token_Middle

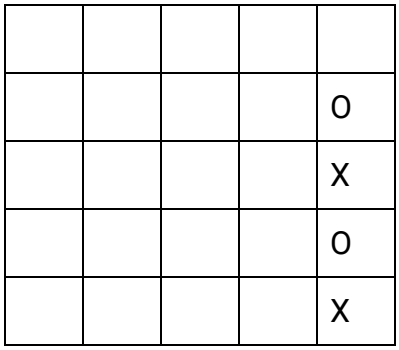
Input	Output	Reason:
pos = <1,2> char = X State: (number to win = 4) | | | | | | | | | | | | | | | | | O | | X | X | X | O | X | | X | O | O | O | X |	checkHorizWin = false State of the board is unchanged	This test case is distinct because there are four X's on the row it is checking, but not sequentially, so it should not detect a win at this position. Function: test_checkHorizWin_Four_Not_Sequential

boolean checkVertWin(BoardPosition pos, char p)

Input	Output	Reason:
pos = <0,2> char = X State: (number to win = 4) | | | | | | | | | | | | | | | | | | | | | | | | | | | X | | |	checkVertWin= false State of the board is unchanged	This test case is distinct because one token is the minimum amount of tokens on the board that this function can be called for. Function: test_checkVertWin_One_Token

Input	Output	Reason:
pos = <3,0> char = X State: (number to win = 4) | | | | | | | X | | | | | | X | O | | | | | X | O | | | | | X | O | | | |	checkVertWin= true State of the board is unchanged	This test case is distinct because the last X was placed at the top of the string of 4 consecutive X's, so this is a situation where the function should return true. Function: test_checkVertWin_Win_Last_Token_Top

Input pos = <4,3> char = X State: (number to win = 4) 	Output checkVertWin= false State of the board is unchanged	Reason: This test case is distinct because there are four X's on the column it is checking, but not sequentially, so it should not detect a win at this position. Function: test_checkVertWin_Four_Not_Sequential

Input pos = <3,4> char = X State: (number to win = 4) 	Output checkVertWin= false State of the board is unchanged	Reason: This test case is distinct because there are four tokens in this column, but they are not the same player character, so it should not detect a win at this position. Function: test_checkVertWin_Four_Not_Same_Char

boolean checkDiagWin(BoardPosition pos, char p)

Input	Output	Reason:
pos = <0,2> char = X State: (number to win = 4) | | | | | | | | | | | | | | | | | | | | | | | | | | | X | | |	checkDiagWin= false State of the board is unchanged	This test case is distinct because one token is the minimum amount of tokens on the board that this function can be called for. Function: test_checkVertWin_One_Token

Input	Output	Reason:
pos = <0,3> char = X State: (number to win = 4) | | | | | | | X | | | | | | X | X | | | O | | O | X | X | | O | | O | O | X | X | O |	checkDiagWin = true State of the board is unchanged	This test case is distinct because the last X was placed at the end of the 4 consecutive X's going down and to the right, so the function needs to count X's up and to the left. Function: test_checkDiagWin_Win_Last_Token_Bottom_Right

Input	Output	Reason:
pos = <3,0> char = X State: (number to win = 4) | | | | | | | |---|---|---|---|---| | X | | | | | | X | X | | | | | O | X | X | | | | O | O | O | X | O |	checkDiagWin = true State of the board is unchanged	This test case is distinct because the last X was placed at the beginning of the 4 consecutive X's going down and to the right, so the function needs to count X's down and to the right. Function: test_checkDiagWin_Win_Last_Token_Top_Left

Input	Output	Reason:
pos = <1,2> char = X State: (number to win = 4) | | | | | | | |---|---|---|---|---| | X | | | | | | X | X | | | | | O | X | X | | | | O | O | O | X | O |	checkDiagWin = true State of the board is unchanged	This test case is distinct because the last X was placed in the middle of the string of 4 consecutive X's, so the function needs to count X's on down and to the right and up and to the left. Function: test_checkDiagWin_Win_Token_In_Middle_Down_Right_Diag

Input	Output	Reason:
pos = <0,1> char = X State: (number to win = 4) | | | | | | | | | | | | | | | | X | | | | X | O | | | X | O | X | | O | X | X | O | O |	checkDiagWin = true State of the board is unchanged	This test case is distinct because the last X was placed at the end of the 4 consecutive X's going down and to the left, so the function needs to count X's up and to the right. Function: test_checkDiagWin_Win_Last_Token_Bottom_Left

Input	Output	Reason:
pos = <3,4> char = X State: (number to win = 4) | | | | | | | | | | | X | | | | | X | X | | | | X | X | O | | O | X | O | O | O |	checkDiagWin = true State of the board is unchanged	This test case is distinct because the last X was placed at the end of the 4 consecutive X's going up and to the right, so the function needs to count X's down and to the left. Function: test_checkDiagWin_Win_Last_Token_Top_Right

Input pos = <1,2> char = X State: (number to win = 4) |				
				X
			X	O
		X	O	X
O	X	O	X	O		Output checkDiagWin = true State of the board is unchanged	Reason: This test case is distinct because the last X was placed in the middle of the string of 4 consecutive X's, so the function needs to count X's on down and to the left and up and to the right. Function: test_checkDiagWin_Win_Token_In_Middle_Down_Left_Diag
---	---	---

boolean checkTie()

Input State: (number to win = 4) |				
X	X	O	X	O
O	X	O	X	X
X	O	X	O	O
O	O	X	O	O
X	X	O	X	X		Output checkTie() = true State of the board is unchanged	Reason: This test case is distinct because every position on the board is filled, and there are no wins anywhere on the board. Function: test_checkTie_Full_Board_No_Wins
---	---	---

Input	Output	Reason:
State: (number to win = 4) X X O O O X O X X X O X O O O O X O O X X O X X	checkTie() = false State of the board is unchanged	This test case is distinct because the board is almost full, but one position is still open, so the function will return false. This ensures that every position is being checked. Function: test_checkTie_One_Position_Open

Input	Output	Reason:
State: (number to win = 4) (5×5 empty board)	checkTie() = false State of the board is unchanged	This test case is distinct because the board is completely empty, so it should return false. Function: test_checkTie_Empty_Board

Input State: (number to win = 4) 					Output checkTie() = false State of the board is unchanged	Reason: This test case is distinct because the board is halfway filled, but there are still many positions open, so it returns false once it checks a blank space. Function:
X	O	X				
O	O	X	O	O		
X	X	O	X	X		

char whatsAtPos(BoardPosition pos)

Input pos = <0,0> State: 					Output whatsAtPos = ‘ ‘ State of the board is unchanged	Reason: This test case is unique because the board position chosen is empty, so it should return a ‘ ‘. Function: test_whatsAtPos_Empty

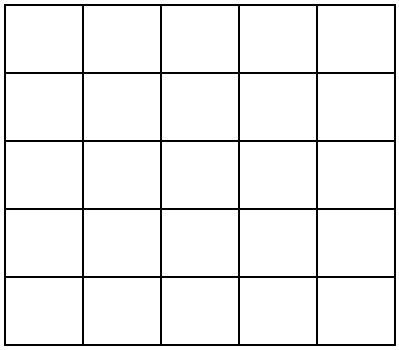
Input pos = <0,1> State: | | | | | | | | | | | | | | | | | | | | | | | | | | X | | | |	Output whatsAtPos = 'X' State of the board is unchanged	Reason: This test case is unique because the board position chosen contains a player character, so it should return that player character. Function: test_whatsAtPos_One_Token

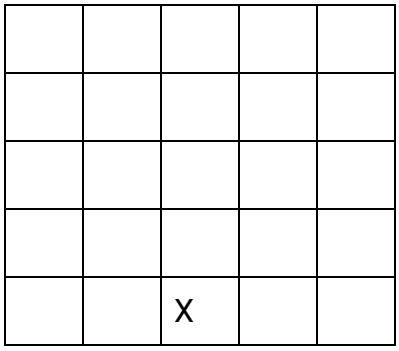
Input pos = <0,4> State: | | | | | | | | | | | | | | | | | | | | | | | | | | | O | X |	Output whatsAtPos = 'X' State of the board is unchanged	Reason: This test case is unique because the board position chosen is at the lower right bounds of the board, and it contains a player character, so it should return that character. Function: test_whatsAtPos_Bottom_Right_Bound

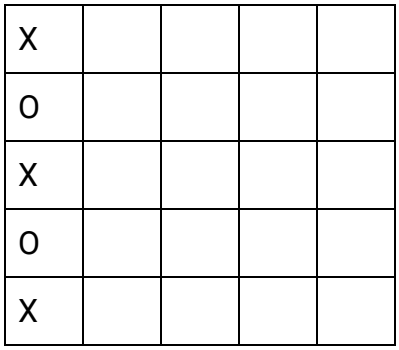
Input pos = <0,0> State: | | | | | | | | | | | | | | | | | | | | | | O | | | X | | | X | O |	Output whatsAtPos = 'X' State of the board is unchanged	Reason: This test case is unique because the board position chosen is at the lower left bounds of the board, and it contains a player character, so it should return that character. Function: test_whatsAtPos_Bottom_Left_Bound

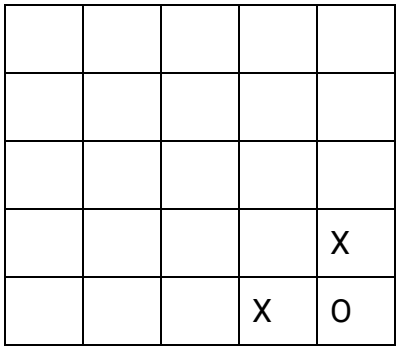
Input pos = <4,4> State: | | | | | X | | | | | | O | | | | | X | X | | | | | O | O | | X | | | X | O |	Output whatsAtPos = 'X' State of the board is unchanged	Reason: This test case is unique because the board position chosen is at the top right bounds of the board, and it contains a player character, so it should return that character. Function: test_whatsAtPos_One_Token

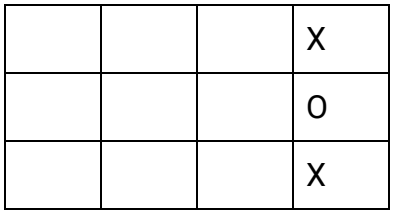
boolean isPlayerAtPos(BoardPosition pos, char player)

Input	Output	Reason:
pos = <0,0> Player: X State: 	isPlayerAtPos = false State of the board is unchanged	This test case is unique because the board position chosen is empty, so it should return false. Function: test_whatsAtPos_Empty

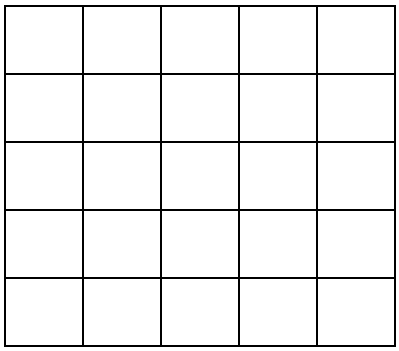
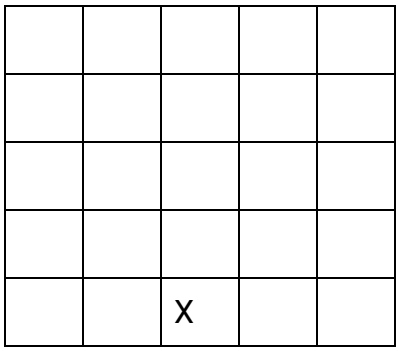
Input	Output	Reason:
pos = <0,2> Player: X State: 	isPlayerAtPos = true State of the board is unchanged	This test case is unique because the board position chosen is in the middle column of the board and is filled by the player's character, so it should return false. Function: test_whatsAtPos_Filled_Middle_Column

Input	Output	Reason:
pos = <4,0> Player: X State: 	isPlayerAtPos = true State of the board is unchanged	This test case is unique because the board position chosen is at the top left boundary of the board and is filled by the player's character, so it should return true. Function: test_whatsAtPos_Filled_Top_Left_Boundary

Input	Output	Reason:
pos = <0,4> Player: X State: 	isPlayerAtPos = false State of the board is unchanged	This test case is unique because the board position chosen is at the bottom right boundary of the board and is not filled by the player's character, so it should return false. Function: test_isPlayerAtPos_Bottom_Right_Wrong_Character

Input pos = <2,3> Player: X State: 	Output isPlayerAtPos = true State of the board is unchanged	Reason: This test case is unique because the board is rectangular, and the board position chosen is at the top right boundary of the board and is filled by the player's character, so it should return true. Function: test_isPlayerAtPos_Top_Right_Boundary_Rect

void placeToken(char p, int c)

Input pos = <0,2> Player: X State: 	Output State: 	Reason: This test case is unique because the token being placed is in an empty column on an empty board. Function: test_placeToken_Empty_Board

Input pos = <2,0> Player: X State:	Output State:	Reason: This test case is unique because I am placing my token in a column that was not empty, and also was not close to being full. Function: test_placeToken_Partially_Full

O				
X				

X				
O				
X				

Input pos = <0,0> Player: X State:	Output State:	Reason: This test case is unique because I am placing my token in the bottom left boundary of the board. Function: test_placeToken_Bottom_Left_Boundary

X				

Input
pos = <4,4>
Player: X
State:

				O
				X
				O
				X

Output

State:

				X
				O
				X
				O
				X

Reason:
This test case is unique because I am placing my token in the top right boundary of the board.

Function:
test_placeToken_Top_Right_Boundary

Input
pos = <4,0>
Player: X
State:

O				
X				
O				
X				

Output

State:

X				
O				
X				
O				
X				

Reason:
This test case is unique because I am placing my token in the top left boundary of the board.

Function:
test_placeToken_Top_Left_Boundary

Deployment:

Type “make” to compile the code.

Type “make run” to run the code

Type “make clean” to remove the files made from compilation.

Type “make test” to compile the test cases

Type “make testGB” to run the test cases for the fast implementation

Type “make testGBMem” to run the test cases for the memory efficient implementation