Requirements

- **Functional Requirements**
    - As a user, I can choose the number of rows in the game board, so that I can change the dimensions of the game board.
    - As a user, I can choose the number of columns in the game board, so that I can change the dimensions of the game board.
    - As a user, I can choose the number in a row needed to win, so that I can change the number in a row needed to win.
    - As a user, I can see the number in a row needed to win, so that I can know the number I need to reach in order to win the game.
    - As a user, I can enter in the number of players that I want to play with, so that I can play with up to nine other players.
    - As a user, I can select the character that I want to represent my player, so that I can distinguish my tokens from those of the other players.
    - As a user, I can choose if I want a fast game implementation or a memory efficient game implementation, so that I can choose whether I care more about saving memory or saving time.
    - As a user, I can choose to play again, so that I can try to win the game against my opponent again.
    - As a user, I can respecify the number of rows each time I play, so that I can change the dimensions of the game board each time.
    - As a user, I can respecify the number of columns each time I play, so that I can change the dimensions of the game board each time.
    - As a user, I can respecify the number in a row needed to win, so that I can know the number I need to reach in order to win the game.
    - As a user, I can respecify whether I want a fast or memory efficient game implementation, so that I can choose this specification for each individual game.
    - As a user, I can respecify the number of players each time I play, so that I can play with a different number of players each time.
    - As a user, I can see when my opponent or I has won the game, so that I can know if the game is over.
    - As a user, I can see when the game has tied, so that I can know if the game is over.
    - As a user, I can see if my input was invalid (out of bounds), so that I can know to choose a different row to place my token.
    - As a user, I can see if my input was invalid (column was full), so that I can know to choose a different row to place my token.
    - As a user, I can try input that may be invalid without losing my turn, so that I can continue to play the game.
- **Non-Functional Requirements**
    - Program must run on Unix
    - Program must include a makefile
    - A new game should always start with Player X
    - Code should be well formatted and documented
    - Program should be easy to use
    - Program should be easy to read and understand

Testing

- This program (like any program) needs to be tested thoroughly in order to make sure it meets all the above requirements. You need to run through all types of scenarios that could happen during a game of connect 4 including: what happens when a column fills up (does it print the correct message saying so?), what happens when the entire board fills up (does it recognize a tie?), what happens when one player wins horizontally, vertically, or diagonally (does the program recognize these wins and print out the correct winning message for each?), what happens when the user inputs invalid numbers (does it print an error

message?), and does the program print a message asking to play again. All of these things should be tested over and over again to ensure the program runs properly.

Constructor

Input: State: 3x3, numtowin = 3	Output: assertEquals(actual board, expected board) = true	Reason: This is to test if the constructor is able to make a gameboard that is on the lower boundary of how small the board can be according to our preconditions. Function: constructorTest_Small

Input: State: 100x100, numtowin = 25	Output: assertEquals(actual board, expected board) = true	Reason: This is to test if the constructor is able to make a gameboard that is on the upper boundary of how big the board can be according to our preconditions. Function: constructorTest_Large

Input: State: 50x50, numtowin = 10	Output: assertEquals(actual board, expected board) = true	Reason: This is to test if the constructor is able to make a gameboard that is in the middle of the boundaries of how big the board can be according to our preconditions. Function: constructorTest_Med

CheckIfFree

Input: State: 5x5, numtowin = 3 O X O X	Output: checkIfFree = false State of the board is unchanged	Reason: This is to test the main reason why we have the checkIfFree function: to see if a column is full (that is, there are no empty spots in that column). This returns false because the column we checked is full. Function:

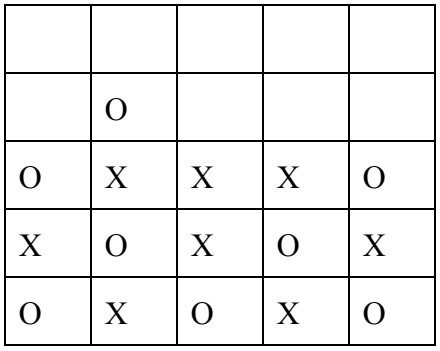
O		checkIfFreeTest_ColumnFull

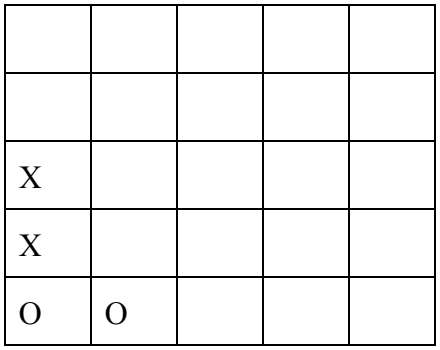
Input: State: 5x5, numtowin = 3 O X O X O	Output: checkIfFree = true State of the board is unchanged	Reason: This is to test if check if free will still work on columns that are not empty (but also not full and therefore still have space in them). It should return true. Function: checkIfFreeTest_NoneEmptyNoneFull

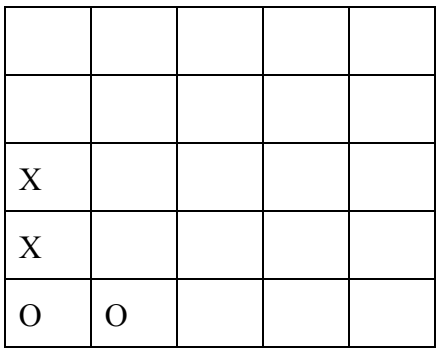
Input: State: 5x5, numtowin = 3 X O X O	Output: checkIfFree = true State of the board is unchanged	Reason: This is to test if checkIfFree works on the only empty column in a board. Function: checkIfFreeTest_OnlyEmptyIs0

CheckHorizWin

Input: State: 5x5, numtowin = 3 X X O O O	Output: checkHorizWin = true State of the board is unchanged	Reason: This is to test a simple horizontal win on the bottom row from the bottom left (average input). Function: checkHorizWinTest_OWinBottomRow

Input: State: 5x5, numtowin = 3 	Output: checkHorizWin = true State of the board is unchanged	Reason: This is to test for a horizontal win in the middle of the board, as opposed to the edges. Function: checkHorizWinTest_XWinMiddleRow

Input: State: 5x5, numtowin = 3 	Output: checkHorizWin = false State of the board is unchanged	Reason: This is to test if checkHorizWin will return false if there is, in fact, no horizontal win yet. Function: checkHorizWinTest_NoWin

Input: State: 5x5, numtowin = 3 	Output: checkHorizWin = false State of the board is unchanged	Reason: This is to test if checkHorizWin will return false if it is called on a spot in the board where no token has been placed. Function: checkHorizWinTest_NoTokenNoWin

Input: State: 5x5, numtowin = 3	Output: checkHorizWin = true State of the board is unchanged	Reason: This is similar to the Bottom Row test, *EXCEPT* the token passed in was not the last token placed. (Token 0,0 was passed in). checkHorizWin should still be able to recognize this as a horizontal win. Function: checkHorizWinTest_OWinFromNotLastPlacedToken

X				
X				
O	O	O		

CheckVertWin

Input: State: 5x5, numtowin = 3	Output: checkVertWin = true State of board is unchanged	Reason: This is to test if checkVertWin can recognize a vertical win on the left boundary of the board. Function: checkVertWinTest_OWinLeftSide

O				
O	X			
O	X			

Input: State: 5x5, numtowin = 3	Output: checkVertWin = true State of board is unchanged	Reason: This is to test if checkVertWin can recognize a vertical win on the right boundary of the board. Function: checkVertWinTest_OWinRightSide

				O
			X	O
			X	O

Input: State: 5x5, numtowin = 3	Output: checkVertWin = true	Reason: This is to test if checkVertWin

<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>X</td><td></td><td></td></tr><tr><td></td><td></td><td>X</td><td></td><td></td></tr><tr><td>O</td><td>O</td><td>X</td><td>O</td><td></td></tr></table>	State of board is unchanged	can recognize a vertical win in the middle of the board (that is, *not* near the boundaries). Function: checkVertWinTest_XWinMiddleCol

Input: State: 5x5, numtowin = 3 <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>O</td><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td><td></td><td></td></tr></table>	Output: checkVertWin = false State of board is unchanged	Reason: This test is to make sure that checkVertWin will not signal a win if there has been no vertical win reached yet. Function: checkVertWinTest_NoWin

Input: State: 5x5, numtowin = 3 <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>O</td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td><td></td><td></td></tr></table>	Output: checkVertWin = true State of board is unchanged	Reason: This is to test if checkVertWin can recognize a vertical win even if the place on the board that is passed in is not the last placed token. (Token passed here was 0,0). checkVertWin should return true. Function: checkVertWinTest_OWinFromNotLastPlacedToken

CheckDiagWin

Input: State: 5x5, numtowin = 3 <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></table>	Output: checkDiagWin = true State of board is unchanged	Reason: This is to test if checkDiagWin can recognize a diagonal win from the bottom left corner of the board and the token passed in

		O		
X	O	O		
O	X	X		

		was the last one placed. Function: checkDiagWinTest_OWinFrom BottomLeft_CheckLast

Input: State: 5x5, numtowin = 3 					
---	---	---	---	---
		O		
X	O	O		
O	X	X				Output: checkDiagWin = true State of board is unchanged	Reason: This is to test if checkDiagWin can recognize a diagonal win from the bottom left corner of the board and the token passed in was not the last one placed. (token passed here was 0,0) Function: checkDiagWinTest_OWinFrom BottomLeft_CheckFirst

Input: State: 5x5, numtowin = 3 					
---	---	---	---	---
		O		
		O	O	X
		X	X	O		Output: checkDiagWin = true State of board is unchanged	Reason: This is to test if checkDiagWin can recognize a diagonal win from the bottom right corner of the board and the token passed in was the last one placed. Function: checkDiagWinTest_OWinFrom BottomRight_CheckLast

Input: State: 5x5, numtowin = 3 					
---	---	---	---	---
		O				Output: checkDiagWin = true State of board is unchanged	Reason: This is to test if checkDiagWin can recognize a diagonal win from the bottom right corner of the board and the token passed in was not the last one placed. Function: checkDiagWinTest_OWinFrom

		O	O	X
		X	X	O

BottomRight_CheckFirst

Input:
State: 5x5, numtowin = 3

X	O	O		
O	X	X		

Output:
checkDiagWin = false

State of board is unchanged

Reason:
This test is to make sure that checkDiagWin will not signal a diagonal win if, in fact, no diagonal win has been achieved yet.

Function:
checkDiagWinTest_NoWin

Input:
State: 5x5, numtowin = 3

X	O	O		
O	X	X		

Output:
checkDiagWin = false

State of board is unchanged

Reason:
This test is to make sure that checkDiagWin will not signal a diagonal win if the spot on the board passed into it has no token placed in it. Therefore, there should be no win from that spot. (location passed here was 0,4)

Function:
checkDiagWinTest_

Input:
State: 5x5, numtowin = 3

O				
O	X			

Output:
checkDiagWin = false

State of board is unchanged

Reason:
This test is to make sure that checkDiagWin will not signal a diagonal win if there is no diagonal win and there has been a different type of win achieved. In this case, it is a vertical win.

Function:
checkDiagWinTest_ThisIsAVertWin

O X		

Input: State: 5x5, numtowin = 3 O X X X O O O	Output: checkDiagWin = false State of board is unchanged	Reason: This test is to make sure that checkDiagWin will not signal a diagonal win if there is no diagonal win and there has been a different type of win achieved. In this case, it is a horizontal win. Function: checkDiagWinTest_ThisIsAHorizWin

CheckTie

Input: State: 3x3, numtowin = 3 O O X X X O O O X	Output: checkTie = true State of board is unchanged	Reason: This is to test if checkTie will signal a tie if the entire board is full and there is no type of win. Function: checkTieTest_BoardFull

Input: State: 3x3, numtowin = 3 O X O X	Output: checkTie = false State of board is unchanged	Reason: This test is to make sure that checkTie will not signal a tie if the board is not full and there is no type of win. (so the game should continue) Function: checkTieTest_BoardNotEmptyNotFull

Input: State: 3x3, numtowin = 3	Output: checkTie = false	Reason: This test is to make sure that checkTie will not signal a tie if

<table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>	State of board is unchanged	the board is empty (and therefore the board is not full and there is no type of win so the game should continue). Function: checkTieTest_BoardEmpty

Input: State: 3x3, numtowin = 3 <table><tr><td>O</td><td></td><td></td></tr><tr><td>O</td><td>X</td><td></td></tr><tr><td>O</td><td>X</td><td></td></tr></table>	Output: checkTie = false State of board is unchanged	Reason: This test is to make sure that checkTie will not signal a tie if there is a type of win, and therefore, there is not a tie. In this case, the win was a vertical win. Function: checkTieTest_VertWinNotATie

WhatsAtPos

Input: State: 5x5, numtowin = 3 <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>X</td><td></td><td></td><td></td><td></td></tr><tr><td>O</td><td>O</td><td></td><td></td><td></td></tr></table>	Output: whatsAtPos = ‘O’ State of board is unchanged	Reason: This is a test to see if whatsAtPos works for the <u>bottom left</u> boundary of the board. Function: whatsAtPosTest_CheckBottomLeftForO

Input: State: 5x5, numtowin = 3 <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>X</td></tr></table>	Output: whatsAtPos = ‘O’ State of board is unchanged	Reason: This is a test to see if whatsAtPos works for the <u>bottom right</u> boundary of the board. Function: whatsAtPosTest_CheckBottomRightForO

			O	O

Input: State: 5x5, numtowin = 3	Output: whatsAtPos = ‘O’ State of board is unchanged	Reason: This is a test to see if whatsAtPos works for the top left boundary of the board. Function: whatsAtPosTest_CheckTopLeftForO

O				
X				
O				
X				
O				

Input: State: 5x5, numtowin = 3	Output: whatsAtPos = ‘O’ State of board is unchanged	Reason: This is a test to see if whatsAtPos works for the top right boundary of the board. Function: whatsAtPosTest_CheckTopRightForO

				O
				X
				O
				X
				O

Input: State: 5x5, numtowin = 3	Output: whatsAtPos = ‘X’ State of board is unchanged	Reason: This is a test to see if whatsAtPos works for the center of the board (away from the boundaries) Function: whatsAtPosTest_checkMiddleForX

		X		
X	O	X	O	X
O	X	O	X	O

Input: State: 5x5, numtowin = 3	Output: whatsAtPos = ‘ ’ State of board is unchanged	Reason: This is a test to see if whatsAtPos works for a <u>blank space</u> in the middle of the board (away from the boundaries). Function: whatsAtPosTest_CheckMiddleForBlank

X	O	X	O	X
O	X	O	X	O

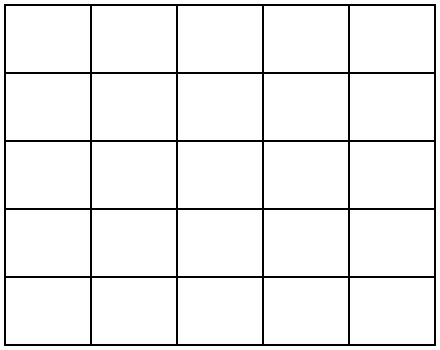
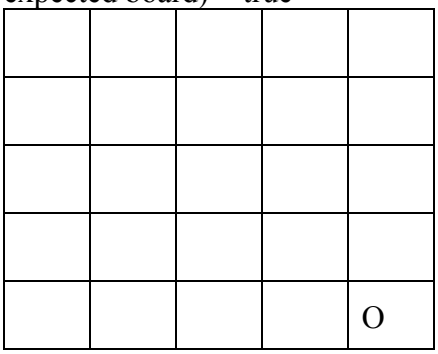
Input: State: 5x5, numtowin = 3	Output: whatsAtPos = ‘ ’ State of board is unchanged	Reason: This is a test to see if whatsAtPos works for the <u>top middle</u> boundary of the board. Function: whatsAtPosTest_CheckMiddleTopForBlank

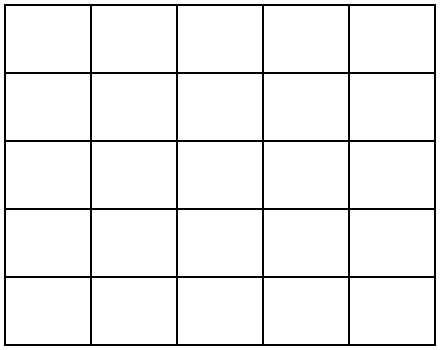
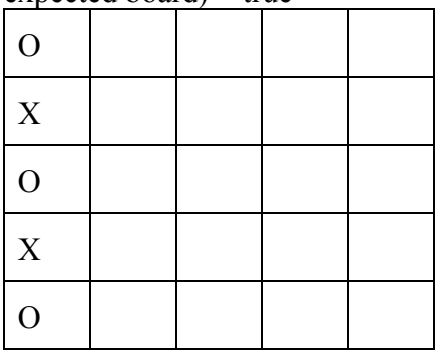
X	O	X	O	X
O	X	O	X	O

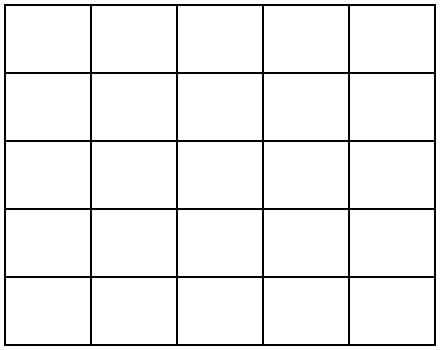
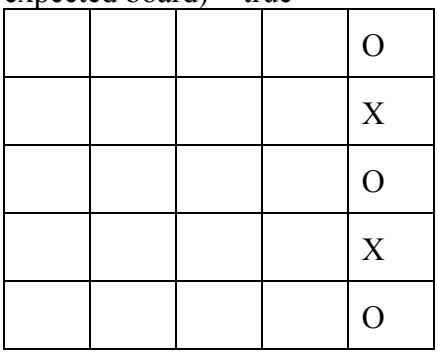
PlaceToken

Input: State: 5x5, numtowin = 3	Output: assertEquals(actual board, expected board) = true	Reason: This is a test to see if placeToken works for the <u>bottom left</u> boundary of the board. Function: placeTokenTest_PlaceBottomLeftAndCheck

O				

Input: State: 5x5, numtowin = 3	Output: assertEquals(actual board, expected board) = true	Reason: This is a test to see if placeToken works for the bottom right boundary of the board. Function: placeTokenTest_PlaceBottomRightAndCheck
		

Input: State: 5x5, numtowin = 3	Output: assertEquals(actual board, expected board) = true	Reason: This is a test to see if placeToken works for the top left boundary of the board. Function: placeTokenTest_PlaceTopLeftAndCheck
		

Input: State: 5x5, numtowin = 3	Output: assertEquals(actual board, expected board) = true	Reason: This is a test to see if placeToken works for the top right boundary of the board. Function: placeTokenTest_PlaceTopRightAndCheck
		

Input: State: 5x5, numtowin = 3	Output: assertEquals(actual board, expected board) = true	Reason: This is a test to see if placeToken works for the middle of the

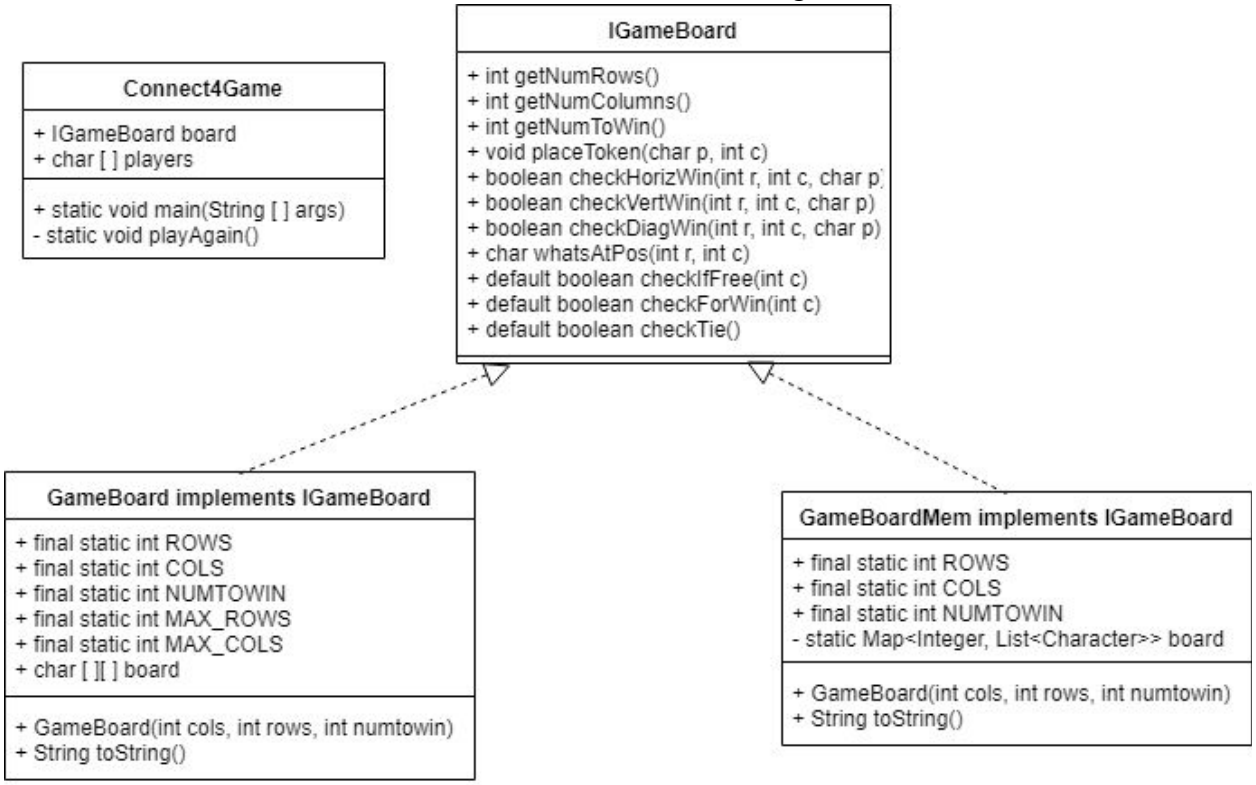
		O		
		X		
		O		
		X		
		O		

board (away from the boundaries).

Function: placeTokenTest_PlaceMiddleAndCheck

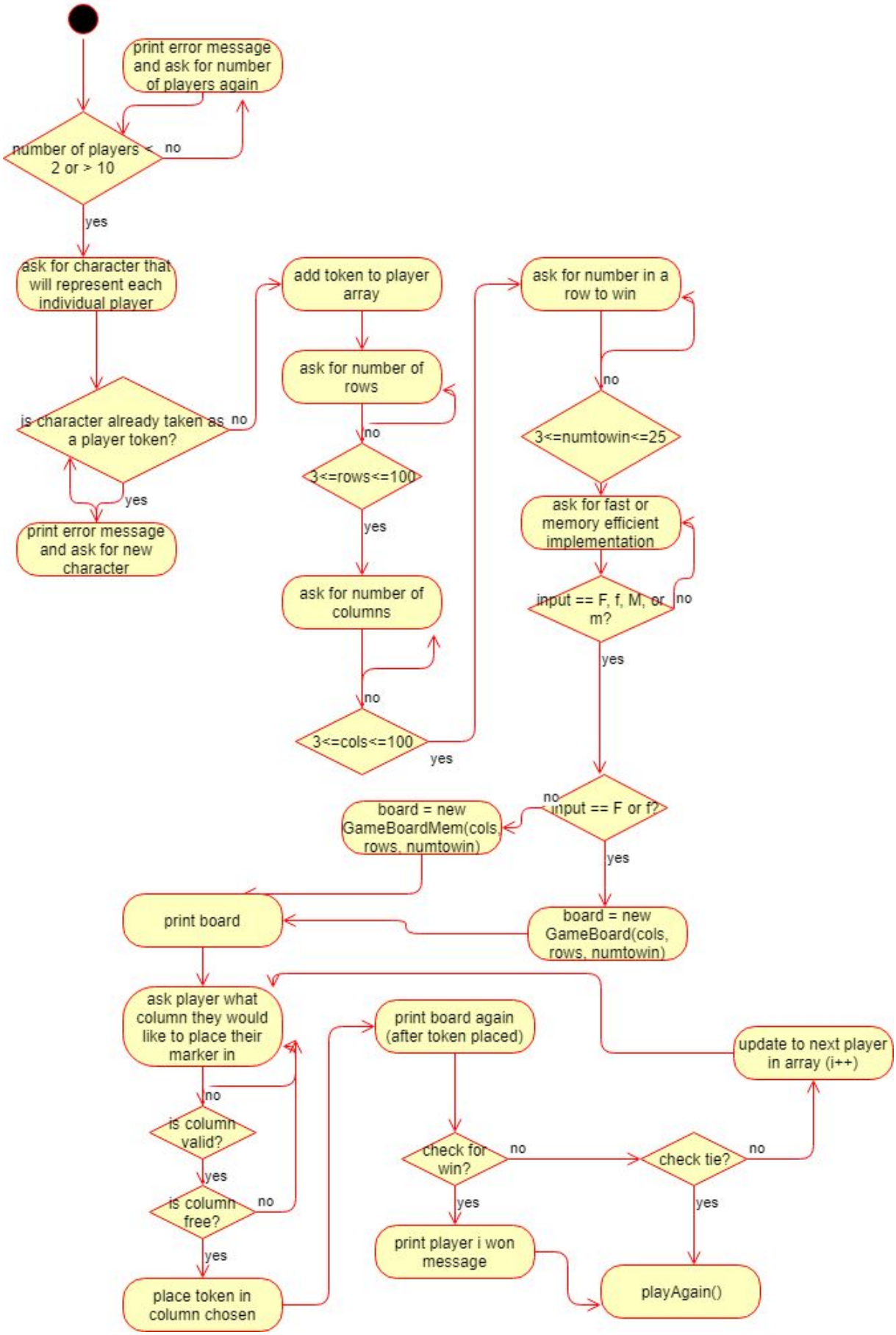
Deployment

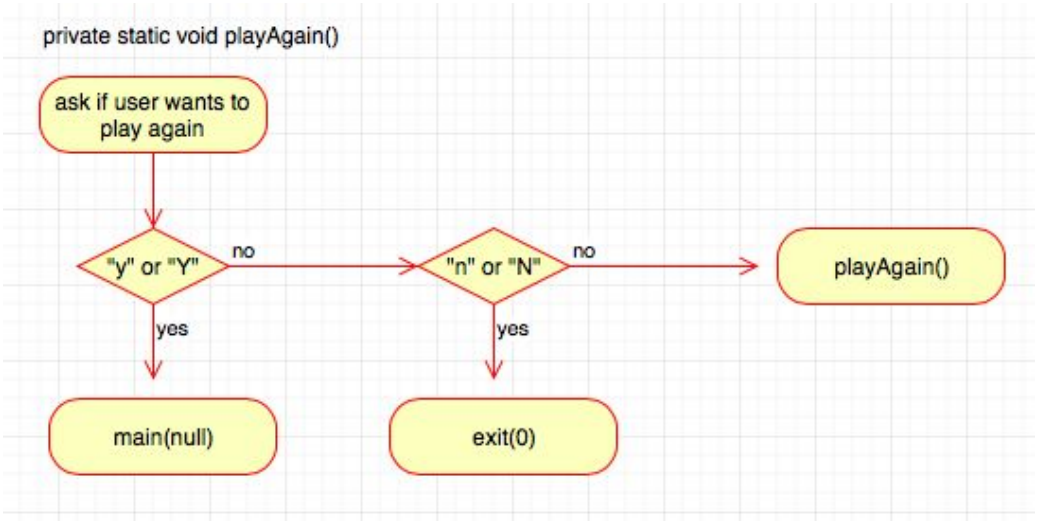
- The command “make” should compile the program files.
- The command “make run” should run the program.
- The command “make clean” should remove all existing .class files from the folder.



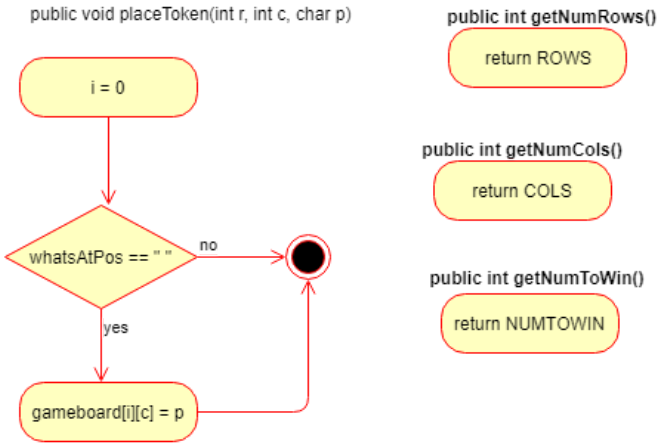
Connect4Game

- Main

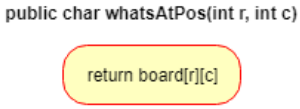


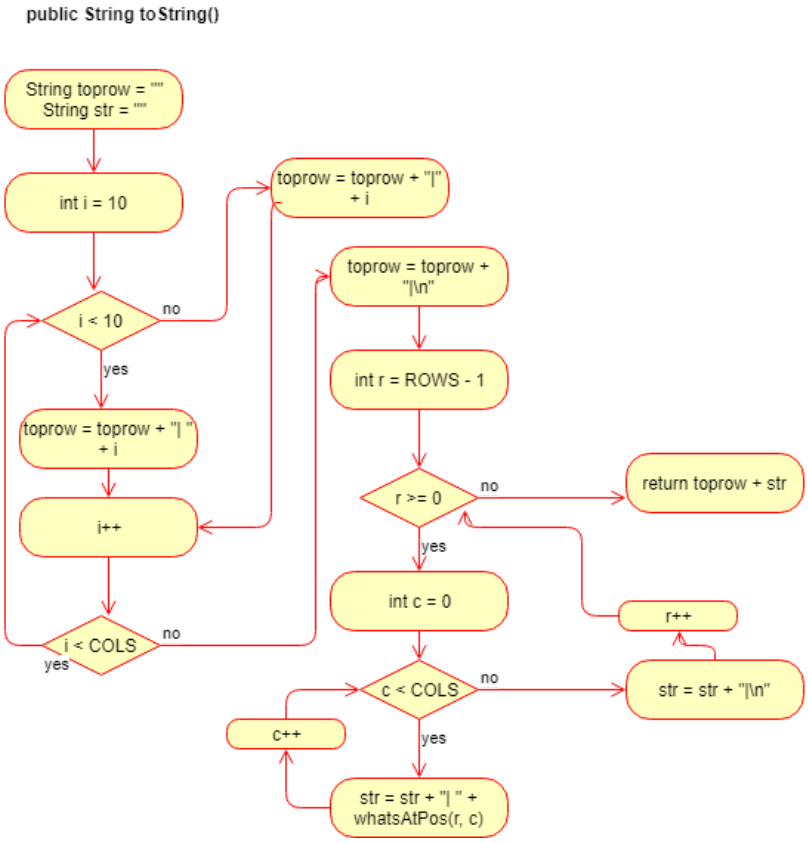

private static void playAgain()
ask if user wants to play again
"y" or "Y"
no
"n" or "N"
no
playAgain()
yes
yes
main(null)
exit(0)

GameBoard


public void placeToken(int r, int c, char p)
i = 0
whatsAtPos == " "
no
yes
gameboard[i][c] = p

public int getNumRows()
return ROWS
public int getNumCols()
return COLS
public int getNumToWin()
return NUMTOWIN


public char whatsAtPos(int r, int c)
return board[r][c]


public String toString()
String toprow = ""
String str = ""
int i = 10
i < 10
no
yes
toprow = toprow + "| " + i
i++
toprow = toprow + "|" + i
i < COLS
no
yes
toprow = toprow + "|\n"
int r = ROWS - 1
r >= 0
no
return toprow + str
yes
int c = 0
r++
c < COLS
no
str = str + "|\n"
c++
yes
str = str + "| " + whatsAtPos(r, c)