Bennett Meares
CPSC 2150
18 November 2018

ConnectX Program Report

Requirements Analysis

Functional Requirements (i.e. User Stories)

1. A user can place a token in a specific row so that the piece is in that row.
2. A user can win the game, so that he will be notified that he's won.
3. Two users can take turns, so that they play the game.
4. A user can have enter another row in the event of bad input, so that he doesn't lose a turn.
5. A user can specify the number of rows in order to play a game of the desired height.
6. A user can specify the number of columns in order to play a game of the desired width.
7. A user can specify the number of pieces in a row needed to win in order to play with custom rules.
8. A user can specify the number of players in order to play with the number of desired players.
9. A user can specify unique tokens for each player so that every player can express his personality through his chosen token.
10. A user can choose the implementation version of the game, so that he can play the implementation of his choosing.
11. A user can choose to play a fast version of the game, so that he can play a fast version of the game.
12. A user can choose to play a memory-efficient version of the game, so that he can play a memory-efficient version of the game.
13. A user can play ConnectX to challenge a friend.
14. A user can play ConnectX to satisfy personal interest.
15. A student can play ConnectX to study its software design.
16. A user can play ConnectX to quell boredom.
17. A user can select an appropriate number of players if previous choice exceeds the limit.
18. A user can choose another player token if the choice is already taken.

Non-functional Requirements

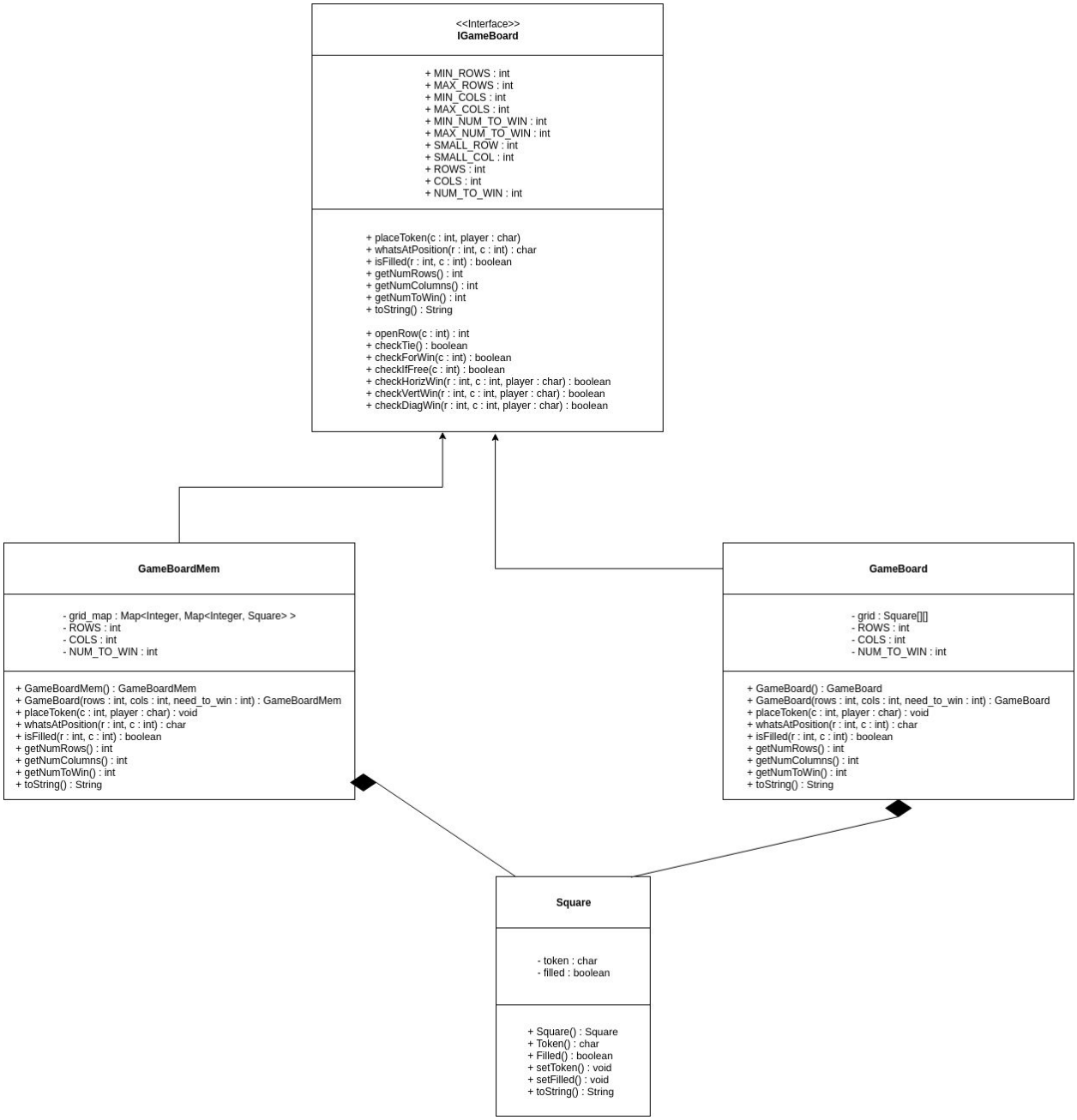
1. ConnectX must run on Java 8.
2. ConnectX must be compatible with Unix.
3. ConnectX must run in a bash environment that supports escape sequences.
4. ConnectX must be run in a console / terminal / text-based environment.
5. ConnectX will be implemented either as a 2D matrix of Squares (which contain character tokens.
6. The GameBoard can implemented as a 2D array of Squares.
7. The GameBoard can be implemented as a 2D map of Squares.

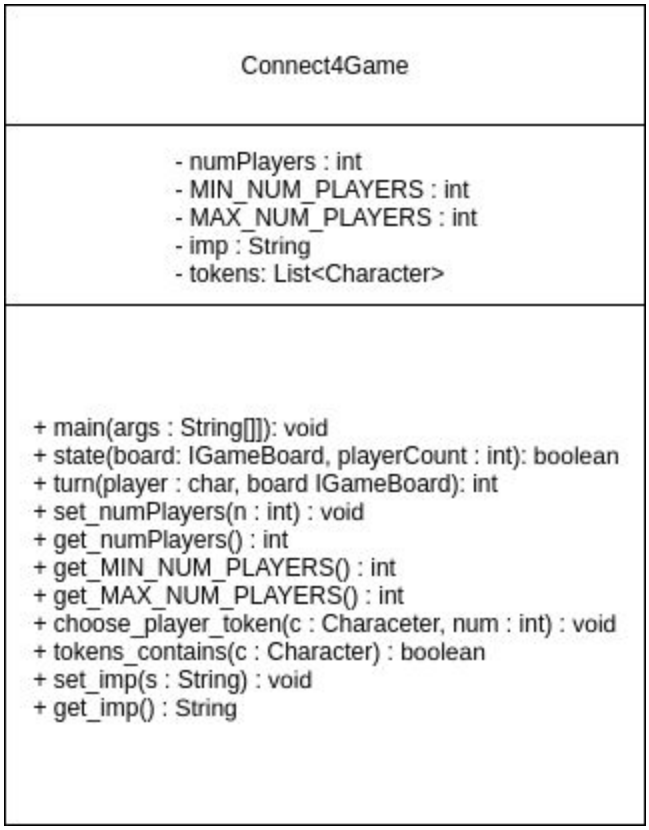
8. ConnectX will be written as an interface that can be implemented as a 2D array of Squares (but could be something else as well).

Design

View on draw.io:
https://drive.google.com/file/d/1TCnAHHec8bNrSxX6f2s9y2JMotc3p3Lx/view?usp=sharing

UML Class Diagrams

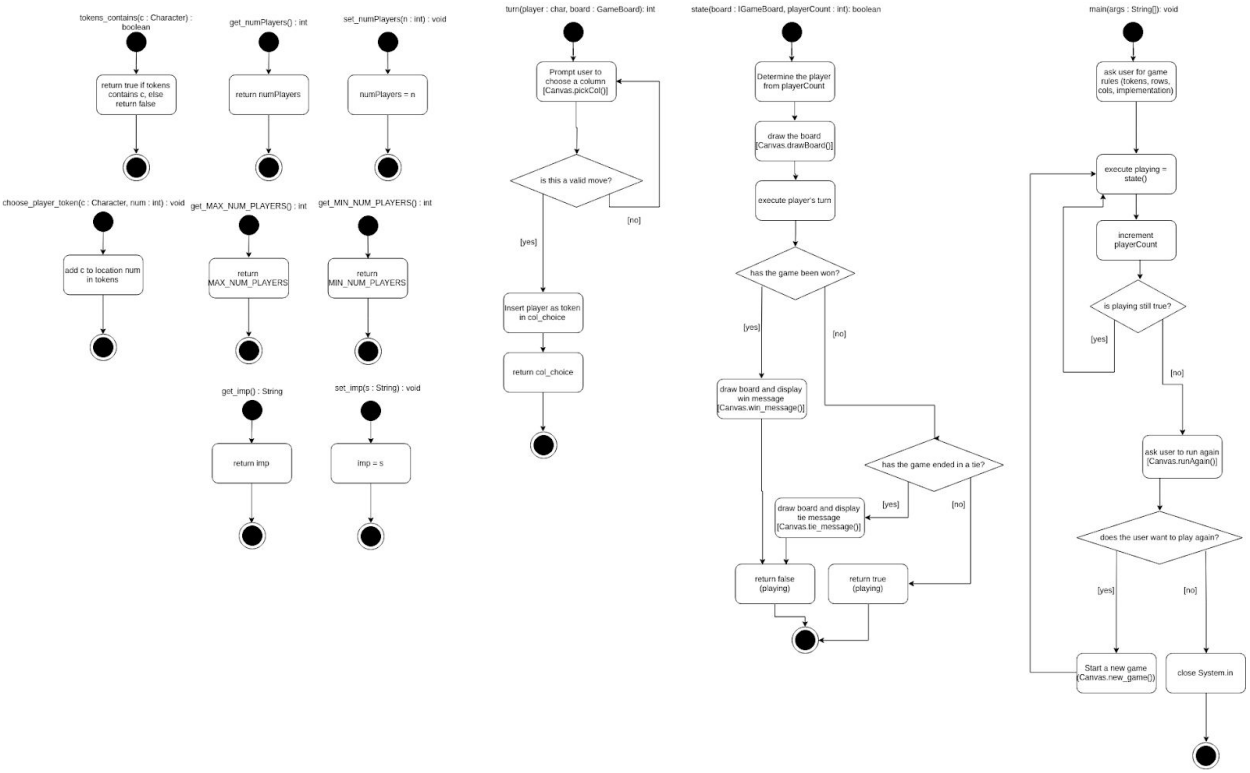

<<Interface>>
IGameBoard
+ MIN_ROWS : int
+ MAX_ROWS : int
+ MIN_COLS : int
+ MAX_COLS : int
+ MIN_NUM_TO_WIN : int
+ MAX_NUM_TO_WIN : int
+ SMALL_ROW : int
+ SMALL_COL : int
+ ROWS : int
+ COLS : int
+ NUM_TO_WIN : int
+ placeToken(c : int, player : char)
+ whatsAtPosition(r : int, c : int) : char
+ isFilled(r : int, c : int) : boolean
+ getNumRows() : int
+ getNumColumns() : int
+ getNumToWin() : int
+ toString() : String
+ openRow(c : int) : int
+ checkTie() : boolean
+ checkForWin(c : int) : boolean
+ checkIfFree(c : int) : boolean
+ checkHorizWin(r : int, c : int, player : char) : boolean
+ checkVertWin(r : int, c : int, player : char) : boolean
+ checkDiagWin(r : int, c : int, player : char) : boolean
GameBoardMem
- grid_map : Map<Integer, Map<Integer, Square> >
- ROWS : int
- COLS : int
- NUM_TO_WIN : int
+ GameBoardMem() : GameBoardMem
+ GameBoard(rows : int, cols : int, need_to_win : int) : GameBoardMem
+ placeToken(c : int, player : char) : void
+ whatsAtPosition(r : int, c : int) : char
+ isFilled(r : int, c : int) : boolean
+ getNumRows() : int
+ getNumColumns() : int
+ getNumToWin() : int
+ toString() : String
GameBoard
- grid : Square[][]
- ROWS : int
- COLS : int
- NUM_TO_WIN : int
+ GameBoard() : GameBoard
+ GameBoard(rows : int, cols : int, need_to_win : int) : GameBoard
+ placeToken(c : int, player : char) : void
+ whatsAtPosition(r : int, c : int) : char
+ isFilled(r : int, c : int) : boolean
+ getNumRows() : int
+ getNumColumns() : int
+ getNumToWin() : int
+ toString() : String
Square
- token : char
- filled : boolean
+ Square() : Square
+ Token() : char
+ Filled() : boolean
+ setToken() : void
+ setFilled() : void
+ toString() : String



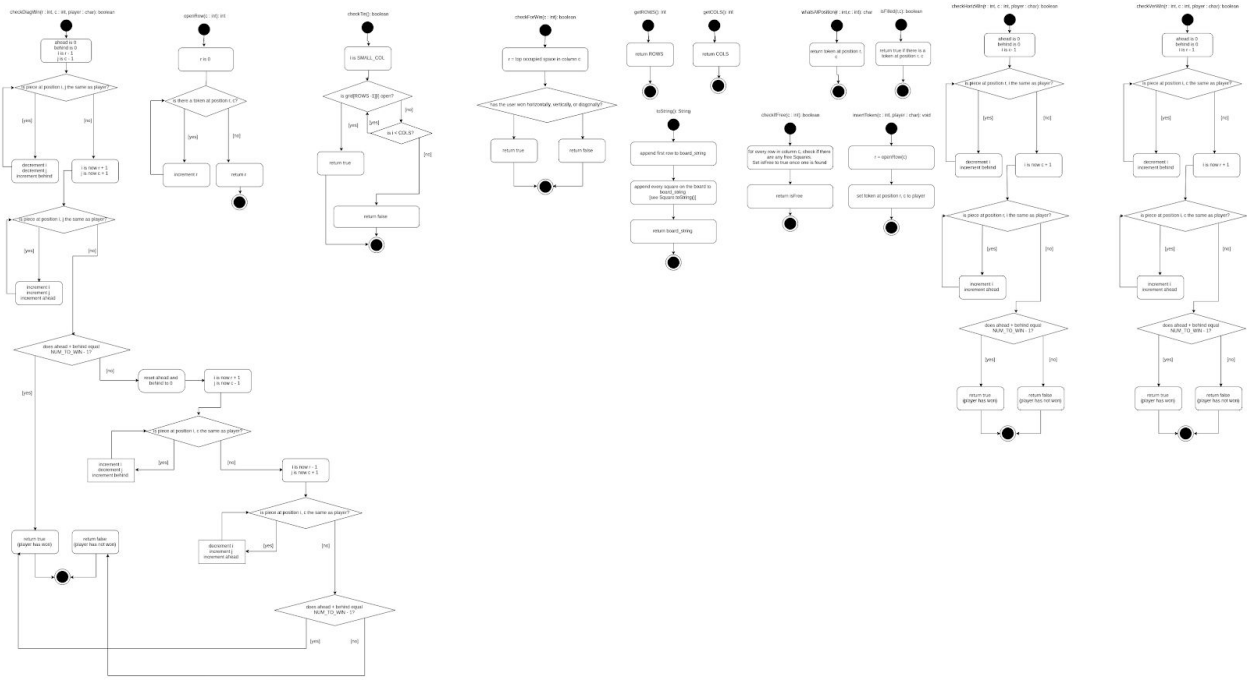
UML Activity Diagrams

Connect4Game

(full res on imgur: https://i.imgur.com/12nYQ0H.png)



IGameBoard (includes GameBoard and GameBoardMem)
(full res version on imgur: https://i.imgur.com/ShsXVRU.png)



Testing

Function Name	Input	Expected Output	Description
IGameBoard_default_constructor_GameBoard()	None	6 x 7 empty game board	This tests whether the GameBoard default constructor creates an empty board of the default 6 x 7 size
IGameBoard_default_constructor_GameBoardMem()	None	6 x 7 empty game board	This tests whether the GameBoardMem default constructor creates an empty board of the default 6 x 7 size
IGameBoard_regular_constructor_GameBoard_100x100()	rows: 100 cols: 100 need_to_win: 6	100 x 100 empty game board	This tests whether the GameBoard regular constructor can initialize a custom board size at the largest allowed size (100x100).
IGameBoard_regular_constructor_GameBoardMem_100x100()	rows: 100 cols: 100 need_to_win: 6	100 x 100 empty game board	This tests whether the GameBoardMem regular constructor can initialize a custom board size at the largest allowed size (100x100).
IGameBoard_regular_constructor_GameBoard_1x1()	rows: 1 cols: 1 need_to_win: 1	1 x 1 empty game board	This tests whether the GameBoard regular constructor can initialize a custom board size at the smallest allowed size (1x1).
IGameBoard_regular_constructor_GameBoardMem_1x1()	rows: 1 cols: 1 need_to_win: 1	1 x 1 empty game board	This tests whether the GameBoardMem regular constructor can initialize a custom board size at the smallest allowed size (1x1).
IGameBoard_regular_constructor_GameBoard_need_to_win_3()	rows: 13 cols: 11 need_to_win: 3	3 in a row needed to win	This tests whether the GameBoard regular constructor allows for a custom number of pieces in a row needed to win a game at the smallest allowed number (3).
IGameBoard_regular_constructor_GameBoardMem_need_to_win_3()	rows: 14 cols: 12 need_to_win: 3	3 in a row needed to win	This tests whether the GameBoardMem regular constructor allows for a custom number of pieces in a row needed to win a game at the smallest allowed number (3).
IGameBoard_regular_constructor_GameBoard_need_to_win_25()	rows: 13 cols: 11 need_to_win: 25	25 in a row needed to win	This tests whether the GameBoard regular constructor allows for a custom number of pieces in a row needed to win a game at the largest allowed number (25).
IGameBoard_regular_constructor_GameBoardMem_need_to_win_25()	rows: 13 cols: 11 need_to_win: 25	25 in a row needed to win	This tests whether the GameBoardMem regular constructor allows for a custom number of pieces in a row needed to win a game at the largest allowed number (25).

IGameBoard_checkIfFree_GameBoard_empty_0()	rows: 6 cols: 7 need_to_win: 4 column that is being checked: 0	true	This tests whether GameBoard returns true if column 0 is empty.
IGameBoard_checkIfFree_GameBoardMem_empty_0()	rows: 6 cols: 7 need_to_win: 4 column that is being checked: 0	true	This tests whether GameBoardMem returns true if column 0 is empty.
IGameBoard_checkIfFree_GameBoard_full_0()	rows: 6 cols: 7 need_to_win: 4 column that is being checked: 0	false	This tests whether GameBoard returns false if column 0 is full.
IGameBoard_checkIfFree_GameBoardMem_full_0()	rows: 6 cols: 7 need_to_win: 4 column that is being checked: 0	false	This tests whether GameBoardMem returns false if column 0 is full.
IGameBoard_checkIfFree_GameBoard_empty_cols_minus_1()	rows: 6 cols: 7 need_to_win: 4 column that is being checked: 0	true	This tests whether GameBoard returns true if the last column is empty.
IGameBoard_checkIfFree_GameBoardMem_empty_cols_minus_1()	rows: 6 cols: 7 need_to_win: 4 column that is being checked: 0	true	This tests whether GameBoardMem returns true if the last column is empty.
IGameBoard_checkIfFree_GameBoard_full_cols_minus_1()	rows: 6 cols: 7 need_to_win: 4 column that is being checked: 0	false	This tests whether GameBoard returns false if the last column is full.
IGameBoard_checkIfFree_GameBoardMem_full_cols_minus_1()	rows: 6 cols: 7 need_to_win: 4 column that is being checked: 0	false	This tests whether GameBoardMem returns false if the last column is full.

IGameBoard_checkHorizWin_GameBoard_0_0()	rows: 6 cols: 7 need_to_win: 4 pieces being placed: (0, 0)	false	This tests whether GameBoard returns false if a horizontal victory has not been achieved by placing a piece at 0,0.
IGameBoard_checkHorizWin_GameBoardMem_0_0()	rows: 6 cols: 7 need_to_win: 4 pieces being placed: (0, 0)	false	This tests whether GameBoardMem returns false if a horizontal victory has not been achieved by placing a piece at 0,0.
IGameBoard_checkHorizWin_GameBoard_0_0123()	rows: 6 cols: 7 need_to_win: 4 pieces being placed: (0, 0) (0, 1) (0, 2) (0, 3)	true	This tests whether GameBoard returns true if a horizontal victory has been achieved by placing pieces at (0, 0); (0, 1); (0, 2); and (0, 3), i.e. by placing four pieces in a row horizontally from left to right.
IGameBoard_checkHorizWin_GameBoardMem_0_0123()	rows: 6 cols: 7 need_to_win: 4 pieces being placed: (0, 0) (0, 1) (0, 2) (0, 3)	true	This tests whether GameBoardMem returns true if a horizontal victory has been achieved by placing pieces at (0, 0); (0, 1); (0, 2); and (0, 3), i.e. by placing four pieces in a row horizontally from left to right.
IGameBoard_checkHorizWin_GameBoard_0_3210()	rows: 6 cols: 7 need_to_win: 4 pieces being placed: (0, 3) (0, 2) (0, 1) (0, 0)	true	This tests whether GameBoard returns true if a horizontal victory has been achieved by placing pieces at (0, 3); (0, 2); (0, 1); and (0, 0), i.e. by placing four pieces in a row horizontally from right to left.
IGameBoard_checkHorizWin_GameBoardMem_0_3210()	rows: 6 cols: 7 need_to_win: 4 pieces being placed: (0, 3) (0, 2)	true	This tests whether GameBoardMem returns true if a horizontal victory has been achieved by placing pieces at (0, 3); (0, 2); (0, 1); and (0, 0), i.e. by placing four pieces in a row horizontally from right to left.

	(0, 1) (0, 0)		
IGameBoard_checkHorizWin_GameBoard_0_0312()	rows: 6 cols: 7 need_to_win: 4 pieces being placed: (0, 0) (0, 3) (0, 1) (0, 2)	true	This tests whether GameBoard returns true if a horizontal victory has been achieved by placing pieces at (0, 0); (0, 3); (0, 1); and (0, 2), i.e. by placing four pieces in a row from both ends toward the center and checking from the center.
IGameBoard_checkHorizWin_GameBoardMem_0_0312()	rows: 6 cols: 7 need_to_win: 4 pieces being placed: (0, 0) (0, 3) (0, 1) (0, 2)	true	This tests whether GameBoardMem returns true if a horizontal victory has been achieved by placing pieces at (0, 0); (0, 3); (0, 1); and (0, 2), i.e. by placing four pieces in a row from both ends toward the center and checking from the center.
IGameBoard_checkVertWin_GameBoard_c0_1p_00()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0)	false	This tests whether GameBoard returns false if a vertical victory has not been achieved by placing a piece at 0,0.
IGameBoard_checkVertWin_GameBoardMem_c0_1p_00()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0)	false	This tests whether GameBoardMem returns false if a vertical victory has not been achieved by placing a piece at 0,0.
IGameBoard_checkVertWin_GameBoard_c0_4p_30()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 0) (2, 0) (3, 0)	true	This tests whether GameBoard returns true if a vertical victory has been achieved by placing pieces at (0, 0); (1, 0); (2, 0); and (3, 0), i.e. by placing four pieces in a row vertically in the same column. This test checks from point (3,0) and counts downward.
IGameBoard_checkVertWin_GameBoardMem_c0_4p_30()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 0)	true	This tests whether GameBoardMem returns true if a vertical victory has been achieved by placing pieces at (0, 0); (1, 0); (2, 0); and (3, 0), i.e. by placing four pieces in a row vertically in the same column. This test checks from point (3,0) and

	(2, 0) (3, 0)		counts downward.
IGameBoard_checkVertWin_GameBoard_c0_4p_00()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 0) (2, 0) (3, 0)	true	This tests whether GameBoard returns true if a vertical victory has been achieved by placing pieces at (0, 0); (1, 0); (2, 0); and (3, 0), i.e. by placing four pieces in a row vertically in the same column. This test checks from point (0,0) and counts upward.
IGameBoard_checkVertWin_GameBoardMem_c0_4p_00()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 0) (2, 0) (3, 0)	true	This tests whether GameBoardMem returns true if a vertical victory has been achieved by placing pieces at (0, 0); (1, 0); (2, 0); and (3, 0), i.e. by placing four pieces in a row vertically in the same column. This test checks from point (0,0) and counts upward.
IGameBoard_checkVertWin_GameBoard_c0_4p_20()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 0) (2, 0) (3, 0)	true	This tests whether GameBoard returns true if a vertical victory has been achieved by placing pieces at (0, 0); (1, 0); (2, 0); and (3, 0), i.e. by placing four pieces in a row vertically in the same column. This test checks from point (2,0) and counts both upward and downward.
IGameBoard_checkVertWin_GameBoardMem_c0_4p_20()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 0) (2, 0) (3, 0)	true	This tests whether GameBoardMem returns true if a vertical victory has been achieved by placing pieces at (0, 0); (1, 0); (2, 0); and (3, 0), i.e. by placing four pieces in a row vertically in the same column. This test checks from point (2,0) and counts both upward and downward.
IGameBoard_checkDiagWin_GameBoard_c0_1p_00()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0)	false	This tests whether GameBoard returns false if a diagonal victory has not been achieved by placing a piece at 0,0.
IGameBoard_checkDiagWin_GameBoard_c0_1p_00()	rows: 6 cols: 7 need_to_win: 4	false	This tests whether GameBoardMem returns false if a diagonal victory has not been achieved by placing a piece at 0,0.

	pieces placed: (0, 0)		
IGameBoard_checkDiagWin_GameBoard_c0_4p_00()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 1) (2, 2) (3, 3)	true	This tests whether GameBoard returns true if a diagonal victory has been achieved by placing pieces at (0, 0); (1, 1); (2, 2); and (3, 3), i.e. diagonally facing upward to the right, This test checks from point (0,0) and counts upward to the right.
IGameBoard_checkDiagWin_GameBoardMem_c0_4p_00()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 1) (2, 2) (3, 3)	true	This tests whether GameBoardMem returns true if a diagonal victory has been achieved by placing pieces at (0, 0); (1, 1); (2, 2); and (3, 3), i.e. diagonally facing upward to the right, This test checks from point (0,0) and counts upward to the right.
IGameBoard_checkDiagWin_GameBoard_c0_4p_33()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 1) (2, 2) (3, 3)	true	This tests whether GameBoard returns true if a diagonal victory has been achieved by placing pieces at (0, 0); (1, 1); (2, 2); and (3, 3), i.e. diagonally facing upward to the right, This test checks from point (3, 3) and counts downward to the left.
IGameBoard_checkDiagWin_GameBoardMem_c0_4p_33()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 1) (2, 2) (3, 3)	true	This tests whether GameBoardMem returns true if a diagonal victory has been achieved by placing pieces at (0, 0); (1, 1); (2, 2); and (3, 3), i.e. diagonally facing upward to the right, This test checks from point (3, 3) and counts downward to the left.
IGameBoard_checkDiagWin_GameBoard_c0_4p_22()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 0) (1, 1) (2, 2) (3, 3)	true	This tests whether GameBoard returns true if a diagonal victory has been achieved by placing pieces at (0, 0); (1, 1); (2, 2); and (3, 3), i.e. diagonally facing upward to the right, This test checks from point (2, 2) and counts both upward to the right and downward to the left.
IGameBoard_checkDiagWin_GameBoardMem_c0_4p_22()	rows: 6 cols: 7	true	This tests whether GameBoardMem returns true if a diagonal victory has

	need_to_win: 4 pieces placed: (0, 0) (1, 1) (2, 2) (3, 3)		been achieved by placing pieces at (0, 0); (1, 1); (2, 2); and (3, 3), i.e. diagonally facing upward to the right, This test checks from point (2, 2) and counts both upward to the right and downward to the left.
IGameBoard_checkDiagWin_GameBoard_c0_4p_30()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 3) (1, 2) (2, 1) (3, 0)	true	This tests whether GameBoard returns true if a diagonal victory has been achieved by placing pieces at (0, 3); (1, 2); (2, 1); and (3, 0), i.e. diagonally facing upward to the left, This test checks from point (3, 0) and counts upward to the left.
IGameBoard_checkDiagWin_GameBoardMem_c0_4p_30()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 3) (1, 2) (2, 1) (3, 0)	true	This tests whether GameBoardMem returns true if a diagonal victory has been achieved by placing pieces at (0, 3); (1, 2); (2, 1); and (3, 0), i.e. diagonally facing upward to the left, This test checks from point (3, 0) and counts upward to the left.
IGameBoard_checkDiagWin_GameBoard_c0_4p_03()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 3) (1, 2) (2, 1) (3, 0)	true	This tests whether GameBoard returns true if a diagonal victory has been achieved by placing pieces at (0, 3); (1, 2); (2, 1); and (3, 0), i.e. diagonally facing upward to the left, This test checks from point (0, 3) and counts downard to the right.
IGameBoard_checkDiagWin_GameBoardMem_c0_4p_03()	rows: 6 cols: 7 need_to_win: 4 pieces placed: (0, 3) (1, 2) (2, 1) (3, 0)	true	This tests whether GameBoardMem returns true if a diagonal victory has been achieved by placing pieces at (0, 3); (1, 2); (2, 1); and (3, 0), i.e. diagonally facing upward to the left, This test checks from point (0, 3) and counts downard to the right.
IGameBoard_checkTie_GameBoard_empty()	rows: 3 cols: 3 need_to_win: 2	false	This tests whether GameBoard returns false if the game has not yet ended in a tie, because there have been no pieces placed yet.
IGameBoard_checkTie_GameBoardMem_empty()	rows: 3 cols: 3	false	This tests whether GameBoardMem returns false if the game has not yet

	need_to_win: 2		ended in a tie, because there have been no pieces placed yet.
IGameBoard_checkTie_GameBoard_one_left_0()	rows: 3 cols: 3 need_to_win: 2 pieces placed: every space except (2,2)	false	This tests whether GameBoard returns false if the game has not yet ended in a tie, because there is still one open space in the top row in column 0.
IGameBoard_checkTie_GameBoardMem_one_left_0()	rows: 3 cols: 3 need_to_win: 2 pieces placed: every space except (2,2)	false	This tests whether GameBoardMem returns false if the game has not yet ended in a tie, because there is still one open space in the top row in column 0.
IGameBoard_checkTie_GameBoard_one_left_cols_minus_1()	rows: 3 cols: 3 need_to_win: 2 pieces placed: every space except (2,2)	false	This tests whether GameBoard returns false if the game has not yet ended in a tie, because there is still one open space in the top row in column cols - 1..
IGameBoard_checkTie_GameBoardMem_one_left_cols_minus_1( )	rows: 3 cols: 3 need_to_win: 2 pieces placed: every space except (2,2)	false	This tests whether GameBoardMem returns false if the game has not yet ended in a tie, because there is still one open space in the top row in column cols - 1..
IGameBoard_checkTie_GameBoard_full()	rows: 3 cols: 3 need_to_win: 2 pieces placed: every space	true	This tests whether GameBoard returns true if the game has ended in a tie, because there are no longer any open spaces.
IGameBoard_checkTie_GameBoardMem_full()	rows: 3 cols: 3 need_to_win: 2 pieces placed: every space	true	This tests whether GameBoardMem returns true if the game has ended in a tie, because there are no longer any open spaces.
IGameBoard_whatsAtPos_GameBoard_x_bottom_left()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space space to be checked: (0, 0)	x	This tests whether GameBoard returns the correct character token. The space being analyzed in this test is the bottom left at (0, 0).

IGameBoard_whatsAtPos_GameBoardMem_x_bottom_left()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space space to be checked: (0, 0)	x	This tests whether GameBoardMem returns the correct character token. The space being analyzed in this test is the bottom left at (0, 0).
IGameBoard_whatsAtPos_GameBoard_x_bottom_right()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space space to be checked: (0, cols - 1)	x	This tests whether GameBoard returns the correct character token. The space being analyzed in this test is the bottom right at (0, cols - 1).
IGameBoard_whatsAtPos_GameBoard_x_bottom_right()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space space to be checked: (0, cols - 1)	x	This tests whether GameBoardMem returns the correct character token. The space being analyzed in this test is the bottom right at (0, cols - 1).
IGameBoard_whatsAtPos_GameBoard_x_top_left()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space space to be checked: (0, cols - 1)	x	This tests whether GameBoard returns the correct character token. The space being analyzed in this test is the top left at (rows - 1, 0).
IGameBoard_whatsAtPos_GameBoardMem_x_top_left()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space	x	This tests whether GameBoardMem returns the correct character token. The space being analyzed in this test is the top left at (rows - 1, 0).

	space to be checked: (0, cols - 1)		
IGameBoard_whatsAtPos_GameBoard_x_top_right()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space space to be checked: (0, cols - 1)	x	This tests whether GameBoard returns the correct character token. The space being analyzed in this test is the top right at (rows - 1, cols - 1).
IGameBoard_whatsAtPos_GameBoardMem_x_top_right()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space space to be checked: (0, cols - 1)	x	This tests whether GameBoardMem returns the correct character token. The space being analyzed in this test is the top right at (rows - 1, cols - 1).
IGameBoard_placeToken_GameBoard_0()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: x at (0, 0)	Game board with x in (0, 0)	This tests whether GameBoard places a token in the specified column at the lowest free space, in this case (0, 0).
IGameBoard_placeToken_GameBoardMem_0()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: x at (0, 0)	Game board with x in (0, 0)	This tests whether GameBoardMem places a token in the specified column at the lowest free space, in this case (0, 0).
IGameBoard_placeToken_GameBoard_0x_0y()	rows: 3 cols: 3 need_to_win: 2 player_x: ‘x’ player_y: ‘y’ pieces placed: x at (0, 0) y at (1, 0)	Game board with x in (0, 0) and y at (1, 0)	This tests whether GameBoard places different tokens in the specified column at the lowest free spaces, in this case (0, 0) and (1, 0).
IGameBoard_placeToken_Game	rows: 3	Game board with x	This tests whether GameBoardMem

Board_0x_0y()	cols: 3 need_to_win: 2 player_x: ‘x’ player_y: ‘y’ pieces placed: x at (0, 0) y at (1, 0)	in (0, 0) and y at (1, 0)	places different tokens in the specified column at the lowest free spaces, in this case (0, 0) and (1, 0).
IGameBoard_placeToken_GameBoard_full()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space	Game board filled with x	This tests whether GameBoard places tokens atop one another at every possible column.
IGameBoard_placeToken_GameBoardMem_full()	rows: 3 cols: 3 need_to_win: 2 player: ‘x’ pieces placed: every space	Game board filled with x	This tests whether GameBoardMem places tokens atop one another at every possible column.

Deployment

Execute the following commands to run this program.

1. unzip ConnectX.zip
2. make
    a. This will compile everything, including the tests.
3. make run
4. Optional: make test
    a. This will run the 68 JUnit tests.