# Overview

In this lab, you will work on a class to represent states in the game tic-tac-toe. Each state represents a single possible situation in the game, and could be used by a game tree to plan out all possible moves and select the best one. **Today you will not build the tree, but will add functionality to the state which would allow the tree to be built**, such as giving values for end-game states, generating a list of all possible moves from the current state, and doing or undoing moves on a state to generate a successor state.

**Note:** Once you have finished the code for this lab, you could use it along with GameTree.java from your project to make a minimax tree application for tic-tac-toe.

# Objectives

* Practice writing methods
* Practice representing states for games
* Preparation for building game trees
* Apply test cases to your program

# Set-up

1. Create a folder on your local machine for your Java program, you can name it whatever you like
2. Start Visual Studio Code (VS Code)
3. In VS Code, Open that newly created folder.
4. Download the starter code from the course public folder ([public/21L](https://cs.unh.edu/~cs416/public/21L)) and save it into your project directory.

# **Implementation**

The starter code includes **TicTacToe.java**, **State.java**, and **TicTacToeState.java**. You should not change anything in **TicTacToe.java**, or **State.java** but you can run the main method of **TicTacToe.java** to test your work. Your task will be to complete several methods in **TicTacToeState.java**.

**public ArrayList<State.Move> findAllMoves( )**

Finds all legal moves from the current state, and returns an ArrayList containing those moves. You will need to check all locations in the board, and create a move for that location if its current value is not X or O (it will be null, if so.)

**public boolean doMove( State.Move move )**

Performs the given move on this state if it was a valid move (inside the boundaries of the board, and does not overwrite existing values), changing the value in the board for this state only if the move was valid. Must also change the value of playerTurn. Returns true if move was valid, false otherwise. **Note:** You will need to cast move to be a TicTacToeState.Move inside this method, since the interface specifies it to be a State.Move in the parameter declaration.

**public void undoMove( State.Move move )**

Undoes the effect of the given move on this state, resetting the board value at the move's row and column to null. **Note:** You will need to cast move to be a TicTacToeState.Move inside this method, since the interface specifies it to be a State.Move in the method parameter declaration.

**public boolean gameOver()**

Returns true if the game is over in this state (either player has gotten 3 of their mark in a row, column, or diagonal, or all spaces are not null), false otherwise. You may want to write helper methods to check if the board is full, and whether one player or the other has won, as you may be able to reuse these in the getValue method later.

**public int getValue()**

Returns the value of an end-game state. Throws a new IllegalStateException exception if the current state is not an end-game, because states cannot be evaluated when their outcome isn't known. The GameTree's nodes will be able to assign values to nodes which do not represent end of games using the minimax algorithm, but you cannot do that in this method.