Kendall Weistroffer
AI Assignment #1- Extra Credit
Due: 10.9.2016

For this extra credit assignment, I implemented Depth-First-Search in order to make a Sudoku solver. In order to do this, I had to specify certain constraints to ensure that the Sudoku puzzles would be solved correctly. This was done by implementing three methods: inRow(), in Col(), and inBlock() with each took in variables relevant to the current position in the grid and the current value that was being checked. Each method would loop through the respective row, column, or 3x3 block to ensure that the passed-in variable val (which is the current 1-9 value that we are looking to insert into our Sudoku grid) can be legally placed.

The method IterateSudoku() is used to iterate over the entire Sudoku grid and look for empty spaces that are still unfilled (have the value 0). If we do come across an empty space in the grid, then we return false and use the method sudokuSolver() to find a legal move for that particular space. If we iterate through the grid and do not find an empty space, then we know that the grid has been filled and we have completed the puzzle, returning true. The Depth-First-Search occurs within the method: sudokuSolver(). Here, we call iterateSolved(), passing in two variables that represent our current row and column, whose references are updated based on where the next available space is within the grid. Once we have come accross a space in the grid, we loop through each value 1-9 and check if each value would result in a legal move (done by calling the method isLegal(), which calls inRow(), in Col(), and inBlock(), and returns true if our value would result in a valid move). If the value is legal, we set that position in the grid to be that particular value, and begins the process again by making a recursive call to sudokuSolver(). If the produced solution does not align with the constraints, backtracking is enabled as the previous gaps (that have since been assigned values 1-9) are reset to be zero and we try an alternate solution.

My program is able to solve 9x9 sudoku problems, and has been tested on a range of difficulties. Each new Sudoku problem must be represented as a 2D vector which is assigned to the variable gameBoard with in the Board class's constructor method. The empty spaces are represented by zeros, and the hints must be specified within the gameBoard.