UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
School of Computer Science & Applied Mathematics

Artificial Intelligence (COMS4033A/7044A)
Game Theory and Adversarial Search

1 Introduction

In this lab, we'll be implementing some of the ideas on game theory and adversarial search that we looked at in the lecture.

2 Iterated Elimination of Dominated Strategies

In this section, we will implement an algorithm that finds a Nash equilibrium under certain circumstances. To do this, we will follow the algorithm from the lecture, where we remove a dominated strategy from the first player, resulting in a smaller payoff matrix. We then remove a dominated strategy for the other player and repeat the process until there is only one strategy remaining for each player. This process may not always lead to a solution, but if it does, then the resulting strategies are a Nash equilibrium, and the game is said to be dominance-solvable.

For example, consider the payoff matrix from the Prisoners' Dilemma below.

	C	**D**
C	3,3	0,5
D	5,0	1,1

Looking at the "row" player first, we see that strategy C is dominated by D, so we can exclude that from consideration. On the resulting payoff matrix, the "column" player's strategy C is also dominated by D. This leaves both players with strategies of D, corresponding to the payoff $(1, 1)$, which is a Nash equilibrium.

Submission 1: Iterated Elimination of Dominated Strategies (50 marks)

Given a payoff matrix, write a C++ program that executes the IEDS algorithm to find a Nash equilibrium for a two-player game. Run the algorithm by first eliminating a dominated strategy for the "row" player, then alternate until one strategy remains. You may assume that the payoff matrix is dominance-solvable and so the algorithm results in one remaining strategy for each player.

Input

Input begins with an integer N, representing the number of strategies for both players. Next follows a representation of the payoff matrix with N rows and N columns. Each column is separated by a space, and the entries at a cell in the matrix are separated by a comma. You may also assume that each payoff is an integer value. For example, the payoff matrix on the previous page is represented by the input

```
2
3,3 0,5
5,0 1,1
```

Output

Print the outcomes at the Nash equilibrium found by the algorithm. The outcomes should be separated by a comma.

Example Input-Output Pairs

Sample Input #1

```
2
3,3 0,5
5,0 1,1
```

Sample Output #1

```
1,1
```

Sample Input #2

```
3
6,1 1,0 6,2
1,4 0,5 5,5
3,4 4,3 2,0
```

Sample Output #2

```
6,2
```

3 Minimax

The previous section considered single move games only. We will now look to implement minimax, which can be used to solve alternating-turn, multi-move games. As a testbed, we will be using the game Tic-tac-toe, which is contested by two players, `X` and `O` with `X` to play first (note that player two is the letter O, not the number zero). For those unfamiliar or requiring a refresher, please see here: `https://www.exploratorium.edu/explore/puzzles/tictactoe`.

Submission 2: Implementing the board (2 marks)

The first step is to represent the state of the board in memory. Write a C++ program that takes in an initial configuration and outputs the state of the board in a human-readable form.

Input

Input consists of a single line containing the characters `.`, `o` (the lowercase letter) and `x`. The symbol `.` represents an empty cell on the board, the symbol `o` represents a mark placed by player `O` and the symbol `x` represents player `X`'s move.

Output

Output the board as a 3×3 grid, where each entry is separated by a space. Empty cells should be indicated by an underscore character.

Example Input-Output Pairs

Sample Input #1

```
.........
```

Sample Output #1

```
_ _ _
_ _ _
_ _ _
```

Sample Input #2

```
.xox..o.x
```

Sample Output #2

```
_ x o
x _ _
o _ x
```

Sample Input #3

```
oxoxxoxox
```

Sample Output #3

```
o x o
x x o
x o x
```

Submission 3: Implementing moves (3 marks)

The next step is to implement action dynamics. Write a C++ program that takes in an initial configuration and an action, and outputs the resulting board.

Input

Input consists of two lines. The first line is a single string of characters representing the current state of the board, as in the previous submission. The next line contains three integers A, B, C separated by spaces. A indicates the player to move, with 0 representing player `X` and 1 representing player `O`. B and C represent the row and column of the move to be made.

Output

Output the resulting board as a 3×3 grid, where each entry is separated by a space. Empty cells should be indicated by an underscore character.

Example Input-Output Pairs

Sample Input #1

```
.........
0 0 0
```

Sample Output #1

```
x _ _
_ _ _
_ _ _
```

Sample Input #2

```
.xox..o.x
0 1 1
```

Sample Output #2

```
_ x o
x x _
o _ x
```

Sample Input #3

```
oxoxxox.x
1 2 1
```

Sample Output #3

```
o x o
x x o
x o x
```

Submission 4: Termination check (5 marks)

Write a C++ program that accepts a state of the game and determines its status. It may be the case that the game is ongoing, or that one of the players has won, or it is a draw. The game is won by one of the players if they have connected three of their symbols vertically, horizontally or diagonally. The game is drawn if this is not the case, but all the cells have been filled. Otherwise, the game is still in progress.

Input

Input consists of a single line containing the characters `.`, `o` (the lowercase letter) and `x`. The symbol `.` represents an empty cell on the board, the symbol `o` represents a mark placed by player `O` and the symbol `x` represents player `X`'s move.

Output

If player `X` has won, output `X wins`. Otherwise, if player `O` has won, output `O wins` (the letter O) If it is a draw, output `Draw` and otherwise output `In progress`

Example Input-Output Pairs

Sample Input #1	Sample Input #2	Sample Input #3
`.........`	`.xoxo.o.x`	`oxoxxoxox`
Sample Output #1	**Sample Output #2**	**Sample Output #3**
`In progress`	`O wins`	`Draw`

Submission 5: Minimax (30 marks)

The previous question evaluated the board at the current state. But now we wish to know, given perfect play from both sides, whether a position is winning for one of the players in the future. Building on what you've done so far, write a C++ program that accepts a (possibly) unfinished state of the game and determines who will win. Note that to determine whose turn it is to player, we need only count the moves already played. If there are an equal number of `x` and `o` symbols, then it is player `X` to move. Otherwise, it is player `O`.

Input

Input consists of a single line containing the characters `.`, `o` (the lowercase letter) and `x`. The symbol `.` represents an empty cell on the board, the symbol `o` represents a mark placed by player `O` and the symbol `x` represents player `X`'s move.

Output

Given the position, output the result of the game under perfect play. If player `X` wins, output `X wins`. Otherwise, if player `O` wins, output `O wins` (the letter O) If it is a draw, output `Draw`.

Example Input-Output Pairs

Sample Input #1	Sample Input #2	Sample Input #3
`.........`	`.xoxo.o.x`	`xo.......`
Sample Output #1	**Sample Output #2**	**Sample Output #3**
`Draw`	`O wins`	`X wins`