CMPUT 350 Lab 10 Prep Problems

Contained are the following files:

- `Solve.h`
- `Solve.cpp`
- `solve_main.cpp`
- `make`(a shell script for generating a.out)
- `solve` (working Linux executable - for reference, use `chmod +x solve` to make it executable)
- `RPS.mge` (test input for -e option - Rock-Paper-Scissors!)
- `RPS.mgr` (test input for -r option)
- `test2x4.mge` (test input for -e option)
- `test2x4.mgr` (test input for -r option)

In this lab you will implement a software tool for processing zero-sum matrix games. Data will be read from `stdin`, and results are printed to `stdout`.

We provide a working executable (`solve`) and input samples that can be used to check whether your program generates correct output, like so:

```
./solve -e < RPS.mge
./a.out -e < RPS.mge      # should generate same output
./solve -r < test2x4.mgr
./a.out -r < test2x4.mgr  # should generate same output
```

The input consists of an N1 by N2 payoff matrix which is encoded like so:

```
3 3
0 -1 +1
+1 0 -1
-1 +1 0
```

(i.e., N1=#rows and N2=#cols followed by (N1*N2) payoff values for the row player)

Depending on the command line option (-r or -e), one or two vectors of length N1 or N2 follow the matrix in the input (see below).

---

1. In file `Solve.cpp` implement function

```
void expected_value(const Matrix &A, const Vector &strat1, const Vector &strat2);
```

which is invoked when using the -e option.

It takes a payoff matrix (in view of the row player), a row player strategy, and a column player strategy as input and computes the expected game result for the row player (see AI Part 5 p.20) and writes it to `stdout`.

For example, for input file `RPS.mge` which contains the standard Rock-Paper-Scissors payoff matrix, a row strategy, and a column strategy:

```
3 3
0 -1 +1
+1 0 -1
-1 +1 0
0.2  0.2 0.6
0.25 0.5 0.25
```

the output of `./solve -e < RPS.mge` is:

```
3 by 3 game
+0.000000 -1.000000 +1.000000
+1.000000 +0.000000 -1.000000
-1.000000 +1.000000 +0.000000
row strategy: 0.2 0.2 0.6
col strategy: 0.25 0.5 0.25
expected value for p1: 0.1
```

Test your program by comparing its output to that of `./solve -e < ...` for various input files (including `RPS.mge`, `test2x4_mge`, and some that you create).

---

2. In file `Solve.cpp` implement function

```
void best_response_to_row(const Matrix &A, const Vector &strat1);
```

which is invoked when using the -r option.

It takes a payoff matrix (in view of the row player) and a row player strategy as input and computes a best-response strategy for the column player and the resulting game value for the row player and writes them `stdout` (see AI Part 5 p.23).

For example, for input file `RPS.mgr` which contains the standard Rock-Paper-Scissors payoff matrix and a row strategy:

```
3 3
0 -1 +1
+1 0 -1
-1 +1 0

0.2 0.2 0.6
```

the output of `./solve -r < RPS.mgr` is:

```
3 by 3 game
+0.000000 -1.000000 +1.000000
+1.000000 +0.000000 -1.000000
-1.000000 +1.000000 +0.000000
row strategy: 0.2 0.2 0.6
best response to row strategy: 1 0 0
value for p1: -0.4
```

I.e., a best response to the row strategy that chooses Rock 20%, Paper 20%, and Scissors 60% of the time is the 100% Rock column player strategy.

Test your program by comparing its output to that of `./solve -r < ...` for various input files (including `RPS.mgr`, `test2x4.mgr`, and some that you create).