**Task 1 (30 points, programming)**

In this task you will implement the value iteration algorithm.

**Arguments**

Your function will be invoked as follows:
`value_iteration(<environment_file>, <non_terminal_reward>, <gamma>, <K>)`
If you use Python, just convert the Matlab function arguments shown above to command-line arguments. The arguments provide to the program the following information:

- The first argument, <environment_file>, is the path name of a file that describes the environment where the agent moves (see details below). The path name can specify any file stored on the local computer.
- The second argument, <non_terminal_reward>, specifies the reward of any state that is non-terminal.
- The third argument, <gamma>, specifies the value of $\gamma$ that you should use in the utility formulas.
- The fourth argument, <K>, specifies the number of times that the main loop of the algorithm should be iterated. The initialization stage, where `U[s]` is set to 0 for every state s, does not count as an iteration. After the first iteration, if you implement the algorithm correctly, it should be the case that `U[s]=R[s]`.

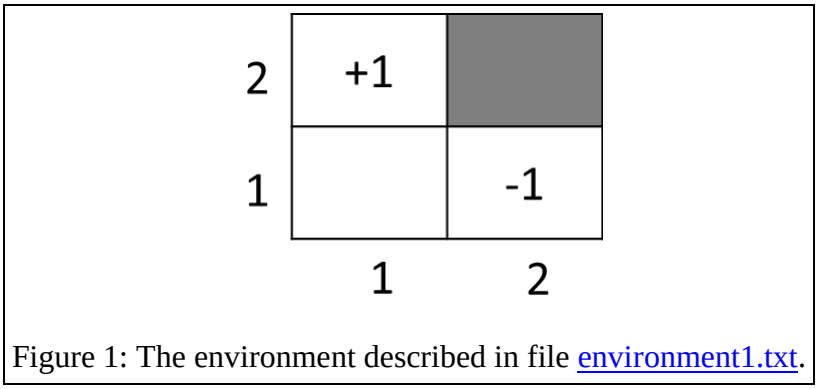

Figure 1: The environment described in file environment1.txt.

The environment file will follow the same format as files environment1.txt and environment2.txt. For example, file environment1.txt describes the world shown in Figure 1, and it has the following contents:

```
1.0,X
.,-1.0
```

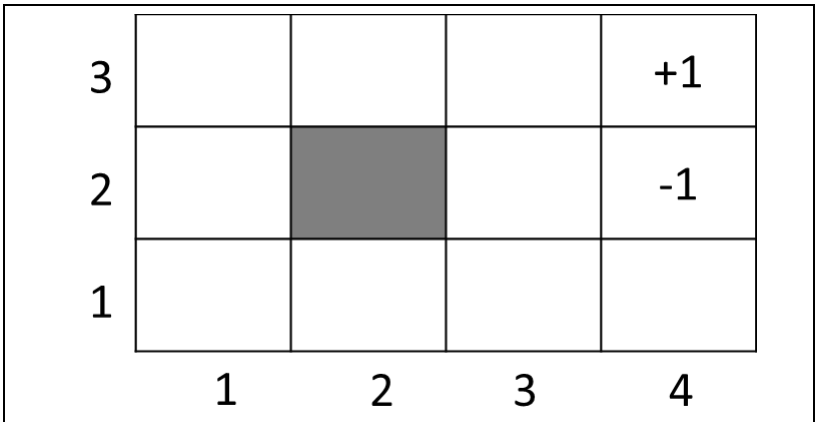

Figure 2: The environment described in file environment2.txt.

Similarly, file environment2.txt describes the world shown in Figure 2, and it has the following contents:

```
.,.,.,1.0
.,X,.,-1.0
.,.,.,.
```

As you see from the two examples, the environment files are CSV (comma-separated values) files, where:

- Character '.' represents a non-terminal state.
- Character 'X' represents a blocked state, that cannot be reached from any other state. You can assume that blocked states have utlity value 0.
- Numbers represent the rewards of TERMINAL states. So, if the file contains a number at some position, it means that that position is a terminal state, and the number is the reward for reaching that state. These rewards are real numbers, they can have any value.

**Implementation Guidelines**

- For the state transition model (i.e., the probability of the outcome of each action at each state), use the model described in pages 9-10 of the MDP slides.
- For terminal states, your model should not allow any action to be performed once you reach those states. For those states, you can just hardcode that their utility is equal to their reward.
- For blocked states, your code should capture the fact (by implementing the appropriate transition model) that they cannot be reached from any other state. You should hardcode the utility values of blocked states to be 0.

**Output**

At the end of your program, you should print out the utility values for all states.

The output should follow this format:

```
%6.3f,%6.3f,...
...
```

In other words, each row in your output corresponds to a row in the environment, and you use the %6.3f format specifier (or equivalents, depending on the programming language) for each utility value. For blocked states, just print a utility of 0.

Do NOT print out this output after each iteration. You should only print out this output after the final iteration.

**Output for answers.pdf**

In your answers.pdf document, you need to provide the complete output for the following invocations of your program:

```
value_iteration('environment2.txt', -0.04, 1, 20)
value_iteration('environment2.txt', -0.04, 0.9, 20)
```

---

**Task 2 (Programming, 40 points)**

In this task, you will implement the `AgentModel_Q_Learning` function from the Reinforcement Learning slides. Implement a function that can be called as:

`q_learning(<environment_file>, <non_terminal_reward>, <gamma>, <number_of_moves>, <Ne>)`

If you use Python, just convert the Matlab function arguments shown above to command-line arguments. The command line arguments should be:

- The first argument, <environment_file>, is the path name of a file that describes the environment where the agent moves, and follows the same format as in the `value_iteration`program.
- The second argument, <non_terminal_reward>, specifies the reward of any state that is non-terminal, as in the `value_iteration` program.
- The third argument, <gamma>, specifies the value of γ that you should use in the utility formulas, as in the `value_iteration` program.
- The fourth argument, <number_of_moves>, specifies how many moves you should process with the AgentModel_Q_Learning function. So, instead of having the main loop run forever, you terminate it after the number of iterations specified by <number_of_moves>.
- The fifth argument, $<N_e>$, is used as discussed in the implementation guidelines, to define the f function.

## Implementation Guidelines

- The outcome of each move should be generated randomly, following the state transition model described in pages 9-10 of the MDP slides.
- As in the previous task, for terminal states your model should not allow any action to be performed once you reach those states. Note that the AgentModel_Q_Learning pseudocode on the slides does handle this case appropriately, and your implementation should handle this case the same way: terminate the current mission and start a new mission. When starting a new mission, the start state should be chosen randomly (with equal probability) from all possible states, except for terminal states and blocked states.
- For the $\eta$ function, use $\eta(N) = 1/N$
- For the f function, use:
    - $f(u,n) = 1$ if $n < N_e$, where $N_e$ is the fifth argument to the `q_learning` function.
    - $f(u,n) = u$ otherwise.

- Your solution needs to somehow simulate the `SenseStateAndReward` function, which should be pretty easy. Your solution should also simulate somehow the `ExecuteAction` function, which should implement the state transition model described in pages 9-10 of the MDP slides, with the probabilities that are used in those slides. As described in those slides, bumping to a wall leads to not moving.
- Note that some computations will require values Q[s,a] that have not been instantiated yet. Uninstantiated values in the Q table should be treated as if they are equal to 0.

### Output

At the end of your program, you should print out the utility values for all states.

The output should follow this format:

```
%6.3f,%6.3f,...
...
```

In other words, each row in your output corresponds to a row in the environment, and you use the %6.3f format specifier (or equivalents, depending on the programming language) for each utility value. For blocked states, just print a utility of 0.

Do NOT print out this output after each iteration. You should only print out this output after the final iteration.

### Output for answers.pdf

In your answers.pdf document, you need to provide the complete output for the following invocations of your program:

```
q_learning('environment2.txt', -0.04, 1, 1000, 20)
q_learning('environment2.txt', -0.04, 0.9, 1000, 20)
```

---

## Task 3 (10 points)

Suppose that you want to implement a Q-Learning algorithm for learning how to play chess.

- What value would you assign for the reward of the non-terminal states? Why?
- What value would you use for $\gamma$ in the `Q_Learning_Update` function? Why?

Your choices should be the best choices that you can make so that your algorithm plays chess as well as possible.

---

## Task 4 (20 points)

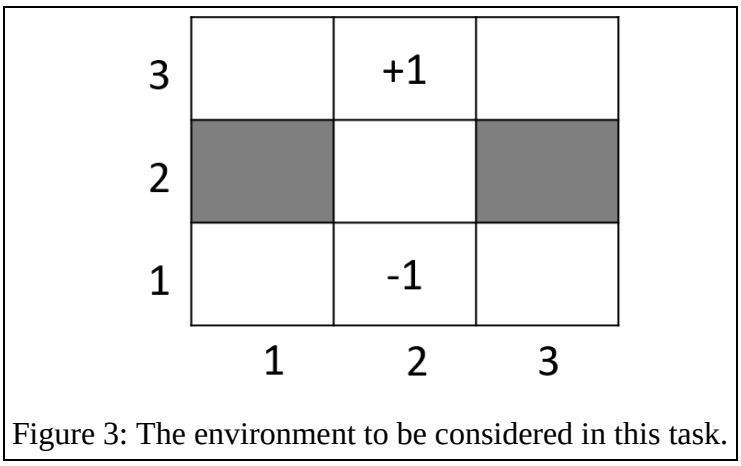

Figure 3: The environment to be considered in this task.

Consider the environment shown on Figure 3. States (1,2) and (3,2) are terminal, with utilities -1 and +1. States (2,1) and (2,3) are blocked. Suppose that actions and state transition models are as described in pages 9-10 of the MDP slides.

**Part a:** Suppose that the reward for non-terminal states is -0.04, and that $\gamma$=0.9. What is the utility for state (2,2)? Show how you compute this utility.

**Part b:** Suppose that $\gamma$=0.9, and that the reward for non-terminal states is an unspecified real number r (that can be positive or negative). For state (2,2), give the precise range of values for r for which the "up" action is not optimal. Show how you compute that range.