Reinforcement Learning: Assignment 1 (100 points).
Due date: Electronic file - before Thursday, January 31, 2019, 21:00.
No need to bring printouts to class on Friday, February 1.
You can work in groups of TWO, but you can also work alone.

You can discuss this assignment only with your partner, if you work in a group, or with the instructor. You cannot use any external resources (including those which are posted on the Web) to complete this assignment. Failure to do this will have negative effect on your mark. By submitting this assignment you acknowledge that you read and understood the course Policy on Collaboration in homework assignments stated in the course management form. Copied work will be penalized.

In each question, the first number of points is for the undergraduate students, while the second number shows the number of points for graduate students. Make sure the TA can run your program and reproduce your results. Provide brief comments in your program to make it more readable for the TA.

Consider a simplified 10-armed bandit problem that we define as a stationary environment that can provide only two types of signals in response to any action: 1 (success) and 0 (failure). Formally, this environment can be represented as a set of 10 different probabilities $\{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}\}$, where q_i is the probability of getting 1 after doing the action a_i (consequently, $1 - q_i$ is the probability of getting 0). In your programs, you have to choose randomly values $q_k, k \in \{1, \ldots, 10\}$ that define each of the available actions. To make sure you get a different sequence of random numbers from different runs of your program, initialize random number generator using system timer, e.g. `srand((unsigned int)time(NULL))`

1 (50 points/30 points). Write a program that implements the UCB algorithm described in Section 2.7 of the 2nd edition of the textbook (2018). Your program must do at least 5000 rounds of interaction with an environment. After every 100 rounds, print how many times an optimal action was chosen and an average reward collected over time. Run your program 100 times, each time with different environments. Notice that you, as the designer of the environment, can determine the probabilities q_i, but your learning algorithm does not know them. The algorithm only knows the sequence of signals of 0 and 1 in response to its actions. Discuss the results of your experiments in the file **report1.txt**

2 (50 points/30 points). Write a program that implements the following two learning automata: L_{R-I} and L_{R-P}. These two types of learning automata are known as linear reward-inaction and linear reward-penalty algorithms, respectively. At each moment of time t, a learning automaton with k actions is represented by a set of probabilities $\vec{p}(t) = \{p_t(1), \ldots, p_t(k)\}$, where $p_t(i) = Prob(a(t) = a_i)$, the probability of doing the action a_i at the moment of time t, and $\sum_{i=1}^{k} p_t(i) = 1$ for any moment of time t. The action chosen by the automaton is the input to the environment which responds with a stochastic reaction 0 or 1. The action m such that $q_m = \max_i q_i$ is the optimal action, but because values q_i are not known to a learning automaton, it must identify the optimal action by updating at each instant t the action probability distribution using the most recent interaction with the environment, namely, using the last chosen action $a(t)$ and the last signal (0 or 1). In the case of linear learning automata, the update is defined by linear functions. For the L_{R-P} automaton, the update is defined as follows. Let the last action $a(t) = \mathbf{a_i}$. If the signal is 1 (success)

$$p_{t+1}(i) = p_t(i) + \alpha \cdot (1 - p_t(i)) \quad \text{/* increase slightly probability of choosing } a_i \text{ */}$$
$$p_{t+1}(j) = (1 - \alpha) \cdot p_t(j), \quad \text{/* for all } j \neq i \text{ decrease probabilities proportionally */}$$

But if the response is 0 (failure)

$$p_{t+1}(j) = \frac{\beta}{k-1} + (1 - \beta) \cdot p_t(j), \quad \text{/* for all } j \neq i \text{ increase probabilities */}$$
$$p_{t+1}(i) = (1 - \beta) \cdot p_t(i) \quad \text{/* decrease this probability to keep the total sum at 1*/}$$

(The learning parameters $0 < \alpha, \beta < 1$ can be equal.) For the L_{R-I} automaton, the parameter $\beta = 0$: the action probabilities are updated only in the case of a reward response from the environment, but penalties are not taken into account.

In your program, you might wish to implement $\vec{p}$ as an array and update this array after each interaction with an environment. Each learning algorithm must do at least 5000 rounds of interaction with an environment. After every 100 rounds, print the number of times a learning algorithm chooses an optimal action and an average reward collected over time. Compare the previously implemented UCB algorithm with learning automata. What algorithm learns faster/better? Discuss briefly your results (in the file **report2.txt**).

3. Bonus work for undergraduates, but **mandatory** for graduate students (**40 points**):

> You may choose to do extra work on this assignment. *Do not attempt any bonus work until the regular part of your assignment is complete.* If your assignment is submitted from a group, write whether this bonus question was implemented by both students (in this case bonus marks will be divided evenly) or whether it was implemented by one person only (in this case only this student will get all bonus marks).

Implement a computer program that learns how to play Tic-Tac-Toe against an opponent that can possible make stupid moves. Follow a description of the learning algorithm presented in Section 1.5 of the new 2018 edition of the textbook. Assume that your program starts the game. Your algorithm has to learn how to play against the following 3 naive opponents: (a) the opponent always randomly takes a move into a position in the same row, where your program played last time, or (b) your opponent always chooses a position in the same column where your program played last time, or (c) if possible, the opponent chooses a diagonally opposite cell to the position where your program played last time, but otherwise, takes a random move in a blank position on the board. For each opponent, try two different versions of your algorithm. First, implement the learning algorithm that learns only from the greedy moves, as described in the textbook. Second, the learning algorithm that learns both from the greedy moves and from exploratory moves. Compare these two versions in terms of total scores they collect against same opponent once they have learned how to play well.

Your algorithm should keep learning until the consecutive value functions change little (e.g., less than 0.001 in every state) or until your algorithm converged to a stable function (policy) choosing an action in every state of the Tic-Tac-Toe game. Initialize value functions as explained in Section 1.5, i.e., do not give your program any bias to choice of actions. When you implement a sequence of step-size parameters α_n, you can either choose a sequence that satisfies the equations (2.7) in the textbook (p.33), or experiment with a sequence of your choice. Write a report, and mention there which sequences α_n did you try to find an optimal policy faster. Compare the results of your program across the 3 opponents. Discuss them in your report that you have to submit as the file **report3.txt**

How to submit this assignment. To submit files electronically do the following. First, create a **zip** archive of all your programs and your reports into a single file:

```
zip yourLoginName.zip [all_your_files] report1.txt report2.txt [report3.txt]
```

where `yourLoginName` is the login name of the person who submits this assignment from a group. Remember to mention at the beginning of each file the *student numbers* and *names* of all people who ever participated in discussions (see the course management form). You may be penalized for not doing so. Second, submit also your assignment ZIP file on D2L using the link provided for the 1st assignment.

Revisions: If you would like to submit a revised copy of your assignment, then simply submit it again on D2L. (The same person must submit.) A new copy of your assignment will override the old copy. You can submit new versions as many times as you like. You do not need to inform anyone about this. Don't ask your team member to submit your assignment, because TA will be confused which version to mark. Only one person from a group should submit different revisions of the assignment. The time stamp of your last ZIP file determines whether you have submitted your work on time.