This is a pdf version of https://git.tu-berlin.de/lis-public/ai-student-workspace/-/blob/main/03/README.md

To obtain the code for this assignment, you will need to fetch and pull new commits from git@git.tu-berlin.de:lis-public/ai-student-workspace.git

As always, only modify the file `solution_??.py`. And even in `solution_??.py`, only modify what the functions do - don't change the function's names. Don't add any additional files, put all your code in `solution_??.py`.

You can run tests by navigating to the task folder `??`, and then simply typing `python3 -m pytest`. If you haven't yet, you will need to install pytest first: `sudo apt install python3-pytest`.

Assignment 3: Sequential Decisions – Decision Tree Methods

3.1: Monte-Carlo Estimation of a Q-function

The default test environment for this exercise is the Integer Domain (state $s \in \mathbb{Z}$ and actions $a \in \{-1, +1\}$) we introduced in other exercises. However, the method you are to implement should be general, i.e., applicable to any domain with the interface of the `simulator` object described below.

For this part of the exercise, you need to modify the function `mc_random_Q(simulator)` in `solution_03.py`. This function's only input, `simulator`, is an object that simulates the domain, and has the following methods:

- `y = reset()`, which resets the simulator to a start state s_0, and returns the observation `y`
- `K = getNumActions()`, which returns the number of discrete possible actions `K`.
- `(r, y, done) = step(a)`, which executes action `a`, leading to a new internal state s', and returns a real-valued reward `r`, an observation `y` (which can be s', but it can also be something different), and `done`, a Boolean that indicates whether the new state is terminal.

In this exercise, `y=0` is constant and not informative, and `r=0` except for the last step into a terminal state. For the last step into a terminal state, `r=-1` (lose) and `r=1` (win) are possible.

In this exercise, there are 2 actions, namely `a=-1` and `a=1`. Let the method generate random rollouts, i.e., when querying the simulator during data collection, always choose random actions. Based on this data, estimate the Q-function $Q(a) = E[r|a_0 = a]$ under the random policy for both possible first decisions a_0.

Your function, `mc_random_Q`, can call `simulator.step` up to 10,000 times (after that, an exception is raised).

Return a numpy array containing the 2 Q-values for $a_0 = -1$ and $a_0 = 1$, in that order.

3.2: Monte-Carlo Tree Search (MCTS, UCT)

Exactly as above, but this time do not estimate $Q(a) = E[r|a_0 = a]$ under the random policy. Instead, estimate $Q(a) = E[r|a_0 = a]$ using UCT with UCB1 and $\beta = 2$ as policy.

For this, you will need to build a decision tree, start at the root node after each `reset`, then use UCT to decide which parts of the tree to explore and expand, until you meet a terminal node (win or lose). Finally, you need to update all nodes along the path once the episode is finished.

For this part of the exercise, you need to modify the function `mc_uct_Q(simulator)` in `solution_03.py`. The inputs and outputs are exactly the same as in 3.1.

For details on UCT you can refer to Browne et al., "A Survey of Monte Carlo Tree Search Methods".