Requirements and Preliminary Design for Reinforcement Learning and Safety test bed.

Jonathan Azpur
York University

1. Introduction

We are looking to test and implement a strategy for safety in machine learning agents. In this project we will build a test bed in which an agent has to achieve some goal **G**. The implementation of the project will support a machine learning agent in the form of reinforcement learning that will evaluate its environment, and produce optimal strategies to achieve its goal state from any possible initial state in it's environment. In order to test safety strategies for this agent we will introduce danger situations into a non-deterministic environment and our algorithms/agent will have to work around these circumstances to obtain not only an optimal strategy but also a safe strategy. The project will be developed in Python.

2. Domain

The domain of this test bed will involve a planetary robot in a starting point (initial state $\mathbf{s_{start}}$) that moves around in a grid world and who's task is to reach a target destination (a goal state $\mathbf{s_{goal}}$). The grid world will have craters in which the robot can fall (safety concern) and borders that limit their movement to a finite number of locations. The robot, or agent, will have the ability to observe the state of its environment ($s \in \mathbf{S}$), and a set of actions ($a \in \mathbf{A}$) it can perform to alter this state. Also our environment will be non-deterministic, where an agent has 95% chance of performing the intended action and 5% chance of performing any other possible action. Here is a visual example of a grid world with a goal state (green square), crates (red squares) and and initial state (blue circle which is the robot):

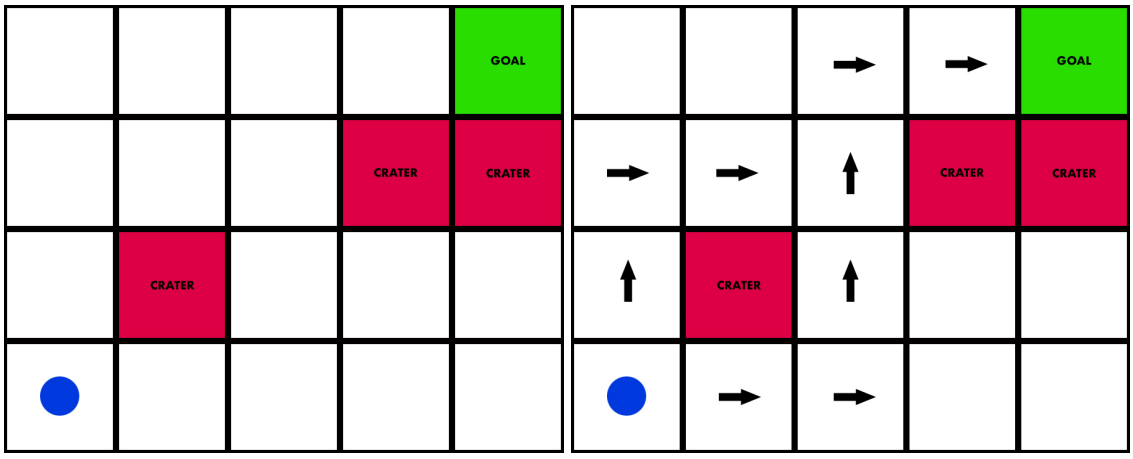

Figure 1 (on the left) Grid map, with an initial state (blue circle), a goal state (green square) and a set of crates (red squares). **(on the right)** Best path from initial state to the goal using Q-learning algorithm

The algorithm will not be limited to this domain. But for test purposes we implement reinforcement learning in the context of a grid world. Our algorithm will be able to work on any circumstances as long as a set of states (including initial and goal state), actions, rewards and punishments are defined. We will be able to adapt our algorithm to any domain.

3. Architecture & Components of Test bed

The test bed will be divided in 2 main components, the environment and the intelligent agent. The environment will be a class that contains all the data necessary to describe and act on an environment, it includes a tuple of states, actions, a reward function and a transition function. The agent will be a class that contains all data necessary for an agent, it includes the specific environment the agent is acting upon, the current state of agent, a transition function, it's agent function and a safety Boolean function.

3.1. Environment Specification

The environment will be specified as a class (i.e. environment.py) with the following components:

- States, S: set of states s.t every s $\in S$ is a possible state of the environment.
- Actions, A: set of actions s.t every a $\in A$ is an action that an agent can perform in this environment.
- Reward Function, $r: S, A \rightarrow R$ s.t for each pair of state/action inputs (s, a) it will output the reward r $\in \mathbb{R}$ an agent would obtain by taking an action a from state s in this environment.
- Transition Function, $tf: S, A \rightarrow S$ s.t for each pair of state-action inputs (s_0, a) it will output a state s_1 an agent would transition into if the agent would take action a from state s_0 (i.e. if agent is in $s_0 = [0,1]$, and I take action $a = UP$, then $tf(s_0, a) = s_1 = [0,2]$).

3.2. Intelligent Agent Specification

The intelligent agent will be specified as a class (i.e. i-agent.py) with the following components, to build a complete agent one needs at least a specific environment it will run on and a matrix that will represent the agent's knowledge of the maximum potential reward for each possible state-action pair (in this case based on Q-learning):

- State, s: the current state the agent finds himself in. t
- Environment, environment class e, such that e will be the specific environment the agent is acting on.
- Knowledge Base: matrix Q s.t. for each pair of state-actions, $Q[s][a]$ is equivalent to the $Q(s, a)$ obtained from a Q-learning algorithm.
- Transition Function, $tf: S, A \rightarrow S$ s.t for each pair of state-action inputs (s_0, a) it will output a new state s_1 an agent would transition to if the agent would take action a from state s_0. (i.e. if agent is in $s_0 = [0,1]$, and I take action $a = UP$, then $tf(s_0, a) = s_1 = [0,2]$)
- Agent Function, $tf: S \rightarrow A$ s.t for each input state s, it will output the action a that provides the maximum reward.
- Safety Boolean Function, $sb: S, A \rightarrow \{True, False\}$ s.t. for each pair of state/action inputs (s, a) it will output $True$ if the state obtained by applying a to s is safe and it will return $False$ if it is not safe.

3.3 PEAS

These are the specific settings our agent will be working in:

- Performance measure: Safety of agent and ability to reach a goal from an initial state $\mathbf{s_{start}}$
- Environment: Grid world with an initial state, a goal state and craters.
- Actuators: move forwards, backwards, to the right and to the left.
- Sensors: Ability of agent to inspect its current state.

4. Algorithm

We will use the reinforcement learning algorithm **Q-learning** to train our agent to find the most efficient and safety path from it's initial state to the goal. In order to teach our agent the most safe and optimal solution our Q-learning algorithm will generate an optimal value $Q(s,a)$ for each state-action pair in the environment that will return the maximum cumulative reward that can be achieved by performing action a on state s, therefore we will be able to determine which action a is the most optimal for each possible state s.

Once a user inputs the necessary data we will go through the following steps in order to obtain the safest/optimum path an agent should take from the initial state to the goal state:

1. Build the environment our agent will be acting on from input data
2. Run Q-learning algorithm to obtain Q(s,a) for all possible state-action pairs
3. Build Agent based on Q table and Environment
4. Given initial state provided on input calculate safest possible path.

Assumptions: We will have access to a set of states and actions, a reward matrix and transition matrix.

Our algorithm will deal with a non-deterministic environment. For the test bed we will assume that given an action a, our agent will have a 95% chance of performing that action and a 5% chance of taking any other possible action. We will consider this non-deterministic actions when running our Q-learning algorithm and when building the path of our agent, this is further described in the next section.

```
Input:
s_i: initial state for our agent.
S: set of states of size n
A: set of actions of size m
R: set of possible rewards/punishments per state
T: transition matrix of size nxm s.t. T[s_0][a] stores the state s_1 that an agent would
arrive to if it were to perform action a on s_0
Algorithm:
- Build Environment based on Input
#will consider non-deterministic environment when running Q-learning algorithm
-Run Q-learning algorithm on environment and obtain Q(s,a) for each pair state-action.
-Build Agent ag based on Input, Environment and Q-learning algorithm
-While agent is not in goal state then i=0…N
   Action_to_perform = ag.next_action()
   #next_action() is ag's agent function based on each Q(s,a) obtained during Q learning
   algorithm
   Action_to_perform = Choose_randomly(Action_to_perform, other_actions, 0.95, 0.15)
   #safety
   If (ag.safe_state(Action_to_perform)) then action[i] = Action_to_perform
   Else then perform ag.next_action() until action obtained is safe
- end while loop
- Output grid map with action[] array of action to go from initial state to goal state
```

5. Safety

The grid map in which the planetary robot will be operating will have a certain number of crates, the safety concern will be to make sure the robot avoids falling into the crates while searching for it's goal. The crate will represent a dangerous situation by providing a negative reward (punishment) if the robot were to move in it's direction.

For other domains, where we do not have crater like in this test bed the algorithm will adapt to any environment as long as there is a negative reward value (punishment) to any state that challenges the safety of the agent.

Also, our environment will be non-deterministic. So in term of the Q-learning algorithm we will have to take into consideration the probability of taking a specific action (95%) and it's potential reward of taking a certain action plus the sum of products of the probability the agent will take another action (5% total) and the potential reward of said actions. In terms of building our path once we have performed the Q-learning algorithm, every action we take when building the safest path there is a 5% chance that our agent will perform an action other than the one it's supposed to take, therefore before taking any steps our agent will have to ensure (with it's built-in function) that the new state it's about to land on is safe, and if it's not safe it should recalculate a new action.