Assignment 2 Part 3
Team 8

- Akshat Goyal 2018101075
- Kanish Anand 2018101025

Markov Decision Process (MDP):

MDP is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming and reinforcement learning.

Linear Programming :

Linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints.
Linear programs are problems that can be expressed in canonical form such as

$$\begin{array}{ll}\text{Maximize} & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ \text{and} & \mathbf{x} \geq \mathbf{0}\end{array}$$

where x represents the vector of variables (to be determined), c and b are vectors of (known) coefficients, A is a (known) matrix of coefficients, and $(\cdot)^{\mathrm{T}}$ is the matrix transpose. The expression to be maximized or minimized is called the objective function (cTx in this case). The inequalities Ax ≤ b and x ≥ 0 are the constraints which specify a convex polytope over which the objective function is to be optimized. In this context, two vectors are comparable when they have the same dimensions. If every entry in the first is less-than or equal-to the corresponding entry in the second, then it can be said that the first vector is less-than or equal-to the second vector.

SOLVING MDP USING LP

To solve the MDP problem, we need to solve the below equation using LP.

$$\max \sum_i \sum_a x_{ia} r_{ia}$$

subject to the constraints:

$$\sum_a x_{ja} - \sum_i \sum_a x_{ia} p_{ij}^a = \alpha_j,$$

$$x_{ia} \geq 0$$

or, equivalently:

$$\max \sum_i \sum_a x_{ia} r_{ia} \;\middle|\; \begin{array}{r} \sum_i \sum_a (\delta_{ij} - p_{ij}^a) x_{ia} = \alpha_j, \\ x_{ia} \geq 0, \end{array}$$

where δ_{ij} is the Kronecker delta, defined as $\delta_{ij} = 1 \iff i = j$.

In Vectorized Form:

$$\max(\mathbf{rx}) \mid \mathbf{Ax} = \boldsymbol{\alpha},\ \mathbf{x} \geq 0,$$

where **A** = **δ** - **P,** where P, probability is given to us.

NOTE: In the assignment, preference order of action for the policy is SHOOT -> DODGE -> RECHARGE

Procedure of making A Matrix:

- Find the list **V** of all the possible (states, action) pair. This pair represents that this action is possible for this state.
- Then we create **A**. **A**'s size is total states (60) X size of **V** (100).
- (i, j)th element in **A** represents that **V**[j].state reaches state i after taking action **V**[j].action.
- Now **A**[i][j]th element is equal to 1 - $p^a{}_{ij}$ if i and **V**[j].state are equal otherwise - $p^a{}_{ij}$.

Procedure of making Policy:

- Now that we have X calculated using the cvxpy library, we can find, for each state, which action has the highest value.
- Action corresponding to the highest value will be selected for the policy.

Can there be multiple policies? Why? What changes can you make in your code to generate another policy?

- Yes, there can be multiple policies. For example, state [1, 3, 1] has the same value for SHOOT or RECHARGE. As the stamina is less it can go for recharge or as the arrows are full, it can shoot also, both possibilities are equal.
- Changing the preference order to RECHARGE -> SHOOT -> DODGE will give a different policy.
- Due to less precision in python and programming languages, we are getting different values for different actions for the same state, varying by very little difference. If we round the values to 3, 4 decimal places, we will get the same value for different actions.