Project Topic -2
Reinforcement Learning using WoLF Algorithm

1. Problem Statement:

Two robots are operating in a factory to bring metal bars to two different production halls. The metal bars are dispensed in one place, only one bar can be picked up at a time, and each robot can only carry one bar at a time. Once a metal bar is picked up, a new one will appear at the dispenser with probability 0.5 every time step (every action taken corresponds to one-time step). Each robot has 3 action choices. It can either try to pick up a metal bar, deliver it to the production hall, or wait. If it tries to pick up a metal bar, it will succeed with probability 0.5 (due to imprecisions in its programming) if there is a metal bar available and fail if there is none available. If it tries to deliver a metal bar to the production hall, it will succeed with probability 1 if it is holding a metal bar and fail otherwise. If it decides to wait it will stay in place. If both robots try to pick up a metal bar at the same time, they will both fail. Each robot receives a payoff of 4 if it successfully delivers a metal bar to the production hall and incurs a cost of 1 if it tries to pick up a metal bar or if it tries to deliver one to the production hall (reflecting the energy it uses up). The wait action does not incur a cost.

2. Project Goal:

The goal of this project is to implement the WoLF learning algorithm on the above problem statement.
I have applied the WoLF Policy Hill Climbing (WoLF-PHC) algorithm to the problem statement.

3. Introduction to WoLF-PHC:

There are 2 desirable properties of any Multiagent Algorithm:

- **Rationality**- If the other players' policies converge to stationary policies then the learning algorithm will converge to a policy that is a best-response to their policies.

- **Convergence-** The learner will necessarily converge to a stationary policy. This property will usually be conditioned on the other agents using an algorithm from some class of learning algorithms.

With WoLF-PHC, we are able to achieve both the above-mentioned properties. The policy matrix of each agent is converged to an optimal policy matrix and that optimal policy matrix gives us the Nash Equilibrium of each state.

WoLF-PHC Algorithm: -

Given below is the policy hill climbing algorithm for player *i*.
The WoLF-PHC algorithm is the extension of the PHC algorithm. The (a) and (b) step are same. Rest other steps are updated. Please refer to the WoLF-PHC figure below.

1. Let α and δ be learning rates. Initialize,

$$Q(s,a) \leftarrow 0, \qquad \pi(s,a) \leftarrow \frac{1}{|\mathcal{A}_i|}.$$

2. Repeat,

   (a) From state s select action a with probability $\pi(s,a)$ with some exploration.

   (b) Observing reward r and next state s',

   $$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right).$$

   (c) Update $\pi(s,a)$ and constrain it to a legal probability distribution,

   $$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \operatorname{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases}.$$

Given below is the WoLF policy hill-climbing algorithm for player.

1. Let $\alpha, \delta_l > \delta_w$ be learning rates. Initialize,

$$Q(s,a) \leftarrow 0, \qquad \pi(s,a) \leftarrow \frac{1}{|\mathcal{A}_i|}, \qquad C(s) \leftarrow 0.$$

2. Repeat,

(a,b) Same as PHC in Table 1

(c) Update estimate of average policy, $\bar{\pi}$,

$$\begin{aligned} & C(s) & \leftarrow & \; C(s)+1 \\ \forall a' \in \mathcal{A}_i \quad & \bar{\pi}(s,a') & \leftarrow & \; \bar{\pi}(s,a') + \\ & & & \; \tfrac{1}{C(s)}\left(\pi(s,a') - \bar{\pi}(s,a')\right). \end{aligned}$$

(d) Update $\pi(s,a)$ and constrain it to a legal probability distribution,

$$\pi(s,a) \leftarrow \pi(s,a) + \begin{cases} \delta & \text{if } a = \operatorname{argmax}_{a'} Q(s,a') \\ \frac{-\delta}{|A_i|-1} & \text{otherwise} \end{cases},$$

where,

$$\delta = \begin{cases} \delta_w & \text{if } \sum_a \pi(s,a)Q(s,a) > \sum_a \bar{\pi}(s,a)Q(s,a) \\ \delta_l & \text{otherwise} \end{cases}.$$

4. Results of my project: -

I followed the algorithm as stated above. In the project we have 8 states and 3 actions. I have done this project in Python 3.

- Before I start explaining what I did in the project, I want to tell that the system has 8 states and each agent (Robot_1 and Robot_2 in our case) has 3 actions

		Dispenser	**R1**	**R2**
States	S1	0	0	0
	S2	0	0	1
	S3	0	1	0
	S4	0	1	1
	S5	1	0	0
	S6	1	0	1
	S7	1	1	0
	S8	1	1	1

Actions	Robot Waits	Robot Picks up	Robot Delivers

- I have initialized the value of alpha to be 0.01, discount factor to be 0.9, delta win to be 0.2 and delta loose to be 0.8
- I initialized the Q(s,a) for both the robots. This returned a numpy array of dimension (8 X 3) with zero values as shown in screenshot below.

```
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
```

Figure 1. Q(s,a) -> 0

- Then, I initialized the π(s,a) matrix with the values 0 because as per the algorithm, it should be initialized with the values equal to 1/|A|. Since the cardinality of the action set in my project is 3, I have initialized it with 0.33 as show in the figure below.

```
[[ 0.33333333  0.33333333  0.33333333]
 [ 0.33333333  0.33333333  0.33333333]
 [ 0.33333333  0.33333333  0.33333333]
 [ 0.33333333  0.33333333  0.33333333]
 [ 0.33333333  0.33333333  0.33333333]
 [ 0.33333333  0.33333333  0.33333333]
 [ 0.33333333  0.33333333  0.33333333]
 [ 0.33333333  0.33333333  0.33333333]]
```

Figure 2- π(s,a) - > 1/|A|

- Then I initialized the counter constant C(s) to 0 as suggested in the algorithm. The goal of the counter is to average the policy in step (c) of the algorithm.

- The next step is to update the q values of both the agents using the equation given in the (b) part of the Policy Hill Climbing algorithms. I calculated and updated the q values using the same equation. Please refer to the screenshots below for the final values of both the robots.

```
Final Q values for Robot 1

[[ 1.10671098  0.2419109   0.2419109 ]
 [ 2.42632421  1.56152414  1.56152414]
 [ 1.73721321  0.87241313  4.33161343]
 [ 0.41760489 -0.44719519  3.01200511]
 [ 1.10673502  0.24193494  0.24193494]
 [ 2.42640721  1.56160714  1.56160714]
 [ 1.73745466  0.87265459  4.33185489]
 [ 0.41817732 -0.44662276  3.01257754]]
```

```
Final Q values for Robot 2

[[ 1.94728548  1.0824854   1.0824854 ]
 [ 0.89663871  0.03183864  3.49103893]
 [ 1.94729039  1.08249031  1.08249031]
 [ 0.89665784  0.03185777  3.49105807]
 [ 1.94735425  1.08255418  1.08255418]
 [ 0.89683543  0.03203536  3.49123565]
 [ 1.9477491   1.08294902  1.08294902]
 [ 0.89749353  0.03269345  3.49189375]]
```

- The next step is to update the estimate of the average policy using the equation mentioned step (c) of WoLF algorithm. I updated using the same equation.

```
Final Policy matrix for Robot 1

[[-52.01916667 -52.04666667 -52.04666667]
 [-76.52166667 -76.71416667 -76.68666667]
 [-89.66666667 -89.66666667 -89.39166667]
 [-89.66666667 -89.66666667 -89.33666667]
 [-52.01916667 -52.04666667 -52.04666667]
 [-76.52166667 -76.71416667 -76.68666667]
 [-89.66666667 -89.66666667 -89.39166667]
 [-89.66666667 -89.66666667 -89.33666667]]
```

```
Final Policy matrix for Robot 2

[[-74.54166667 -75.06416667 -75.00916667]
 [-89.66666667 -89.66666667 -89.33666667]
 [-74.54166667 -75.06416667 -75.00916667]
 [-89.66666667 -89.66666667 -89.33666667]
 [-74.54166667 -75.06416667 -75.00916667]
 [-89.66666667 -89.66666667 -89.33666667]
 [-74.54166667 -75.06416667 -75.00916667]
 [-89.66666667 -89.66666667 -89.33666667]]
```

- The final step is to update the policy matrix π(s,a) in order to get the converged policy matrix. This final policy matrix for the both the agents leads to the Nash Equilibrium of all the states. I was successfully able to achieve the correct Nash Equilibrium of all the states. Please refer the screenshots below.

```
Nash Equilibrium for State 0
wait , wait

Nash Equilibrium for State 1
wait , deliver

Nash Equilibrium for State 2
deliver , wait

Nash Equilibrium for State 3
deliver , deliver

Nash Equilibrium for State 4
wait , wait

Nash Equilibrium for State 5
wait , deliver

Nash Equilibrium for State 6
deliver , wait

Nash Equilibrium for State 7
deliver , deliver
```

- Later on, after reaching the convergence of the optimal policy matrix (in around 2000 iterations per agent), I was able to get the Nash Equilibrium for all the states.

5. <u>References: -</u>

a. http://www.cs.cmu.edu/~mmv/papers/01ijcai-mike.pdf
b. http://www.masfoundations.org/mas.pdf