# Programming Assignment 2

CS6700

Dhruv Laad, AE16B011

08 March 2020

## 1 Q-Learning and SARSA

### 1.1 Question 1

The environment is created in the folder `puddle-world`. The state transitions are handled by the function `puddle_world_transition()`

### 1.2 Question 2

#### 1.2.1 Goal $\mathcal{A}$

Using 1000 episodes per run allows for convergence of the Q-learning algorithm. The reward accumulated at the end of 1000 episodes, averaged over 50 episodes, almost converges to 10, indicating that the agent is able to find a policy that majorly avoids the puddle region, ref. Fig. 1. The steps taken to completion also approach 22, which indicates the algorithm is taking the minimum steps possible.

An interesting point to note is that when Q-learning was initially run with "typical" values for $\epsilon$ in the $\epsilon$-greedy policy, $\epsilon = 0.1$, the algorithm does not converge, and the rewards keep oscillating about 0. This is because "too much" exploration occurs with this $\epsilon$ value, since the stochastic transitions of the puddle world also add an $\epsilon$-greedy type exploration to the agent. Thus, to achieve satisfactory results, $\epsilon = 0.01$ was used in the $\epsilon$-greedy policy.

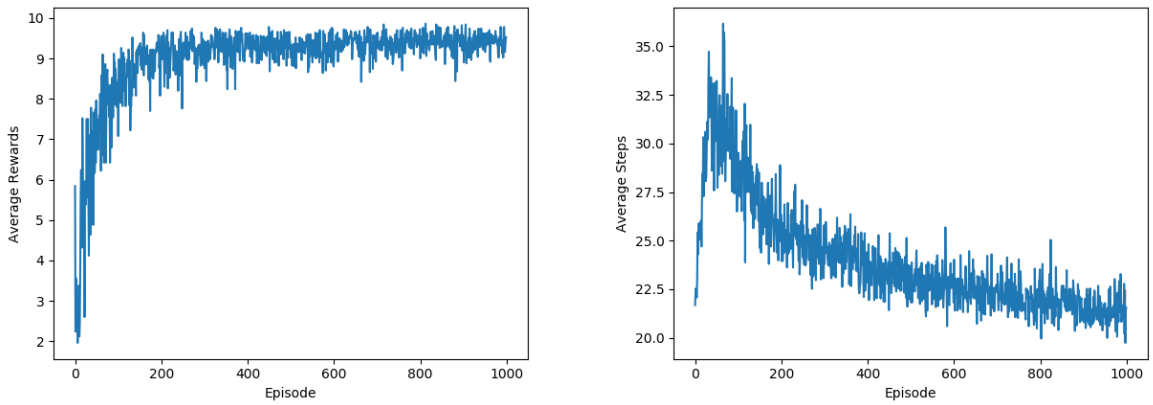



Figure 1: Rewards and steps to completion for Q-learning applied to the puddle world with goal state $\mathcal{A}$

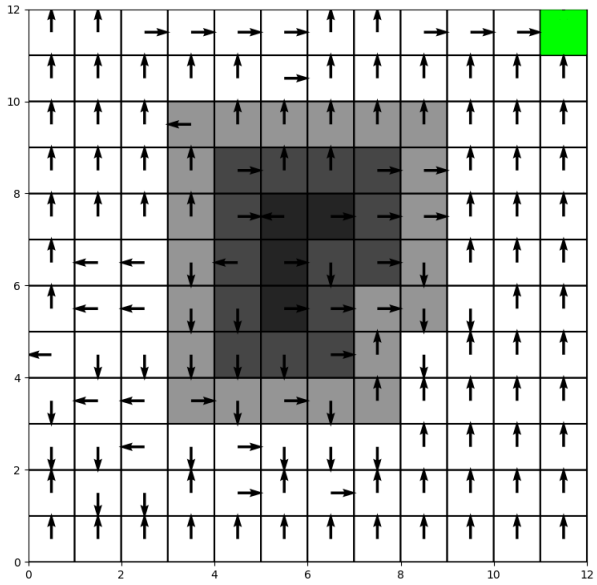


Figure 2: Optimal policy derived greedily from Q-learning agent for goal $\mathcal{A}$

### 1.2.2 Goal $\mathcal{B}$

Results similar to the case of goal $\mathcal{A}$ are observed here. Fig. 5 indicates once again that the policy derived by the agent is able to avoid a large part of the puddle. Initially, the agent takes a very long time to reach the goal state, from Fig. 5, with the first episode requiring steps of $\mathcal{O}(10^5)$, but eventually the algorithm converges to $\approx 25$ steps required to completion, another sign that the agent has learnt to "avoid" the puddle to reach the goal state.

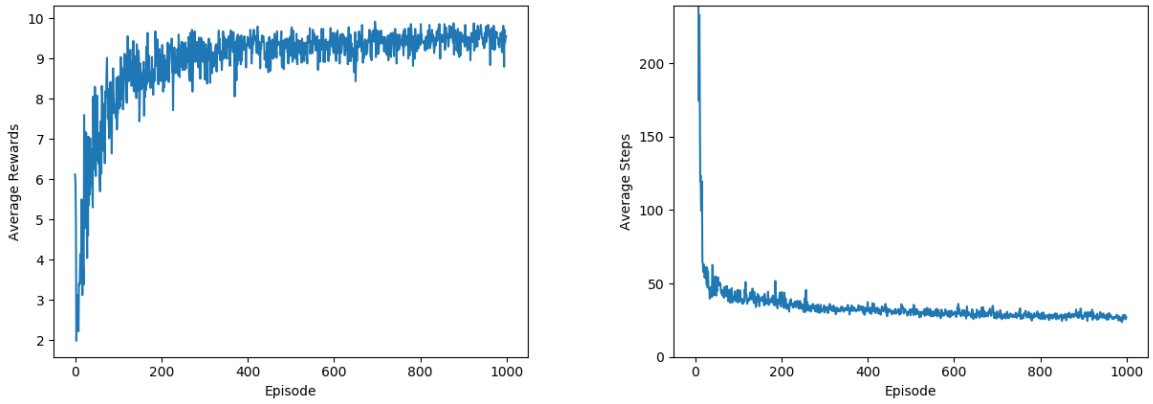



Figure 3: Rewards and steps to completion for Q-learning applied to the puddle world with goal state $\mathcal{B}$

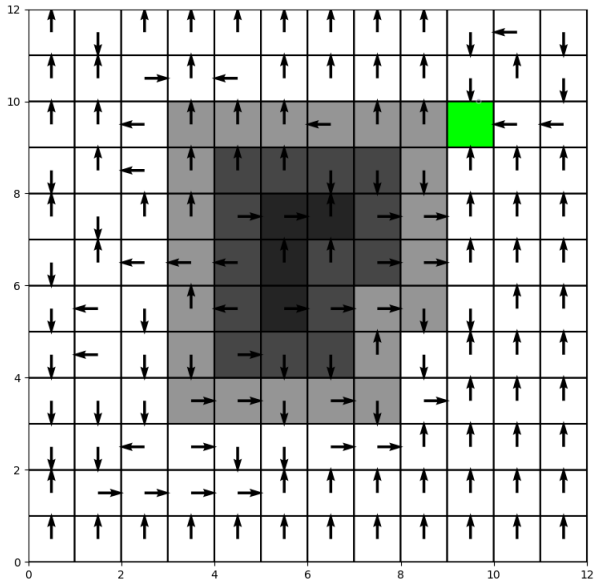



Figure 4: Optimal policy derived greedily from Q-learning agent for goal $\mathcal{B}$

### 1.2.3 Goal $\mathcal{C}$

Without turning the "wind" off for this goal, the agent is not able to converge to the goal state. Preliminary runs showed episodes running well beyond $\mathcal{O}(10^6)$ steps. For the results shown in Fig. the wind was turned off.

Initially the agent encounters a lot of negative reward, as it explores through the puddle region. The average reward accumulated in the first episode is $\mathcal{O}(-10^2)$, but it quickly drops down close to 0, to finally converge to $\approx 8$, indicating it is able to avoid the puddle region. This inference is reinforced by the fact that the steps to completion, initially taking $\mathcal{O}(10^5)$, converge to $\approx 23$, indicating that the agent is able to find a policy around the puddle.

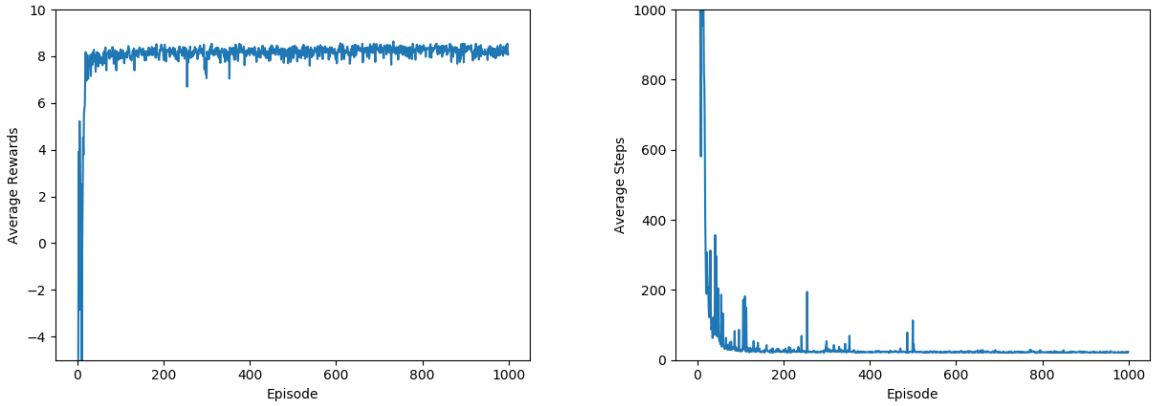



Figure 5: Rewards and steps to completion for Q-learning applied to the puddle world with goal state $\mathcal{C}$

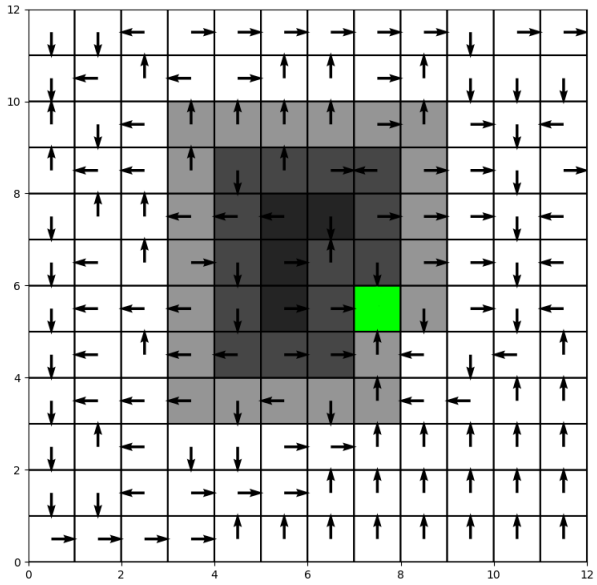



Figure 6: Optimal policy derived greedily from Q-learning agent for goal $\mathcal{C}$

#### 1.2.4 Reaching goal $\mathcal{C}$ with wind

An attempt to solve the problem with the Westerly wind active is made in this section. As mentioned previously, using the original rewards we cannot get a solution in reasonable time. So, the rewards are modified as follows:

- The standard reward for moving into the puddle depths is included, `puddle_reward`($\boldsymbol{s}$)
- The termination condition for each episode is changed so that the episode now terminates both when the goal state is reached and when the number of steps taken by the agent is $> 1000$. If the episode terminates due to the latter, the agent is penalized with $r_t(\boldsymbol{s}) = -4$.
- To further promote the agent to find the goal state fast, a dense reward is specified that depends on the distance of the agent from the goal state, $r_d(\boldsymbol{s}) = -0.1||\boldsymbol{s} - \boldsymbol{s}_{\text{goal}}||$

Thus, the final rewards are, $r(\boldsymbol{s}) = r_d(\boldsymbol{s}) + r_t(\boldsymbol{s})$+ `puddle_reward`($\boldsymbol{s}$)

The results for the puddle world, with the wind on and with these modified rewards, are shown in Fig. 7. As expected, initially the agent takes the terminal number of steps (1000) and accumulates a large negative reward ($\mathcal{O}(-10^2)$). But there is quick improvement and the number of steps taken to reach the goal converges to $\approx 20$ where as the reward accumulated at the end of the $1000^{\text{th}}$ episode is $\approx 0$. The agent is thus able to reach goal state, with minimal interaction with the puddle, despite the wind.

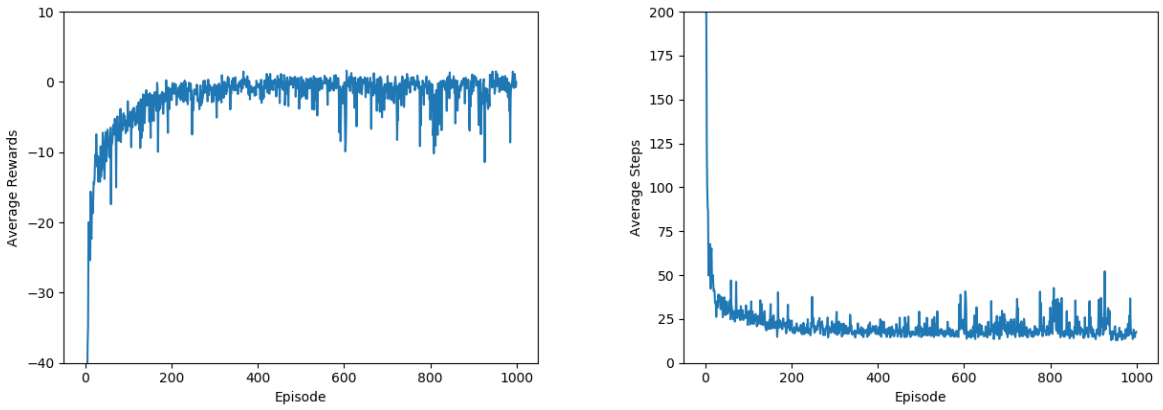



Figure 7: Rewards and steps to completion for Q-learning applied to the puddle world with goal state $\mathcal{C}$ with wind

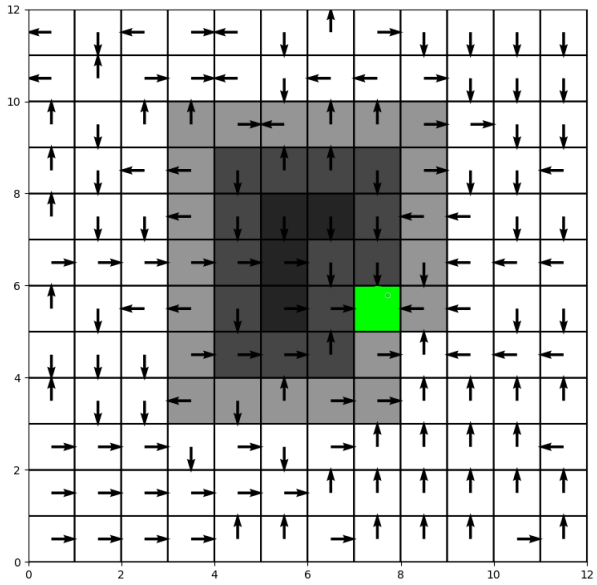


Figure 8: Optimal policy derived greedily from Q-learning agent for goal $\mathcal{C}$ with wind

## 1.3 Question 3

The results for using `SARSA` with the puddle-world are very similar to the ones obtained using Q-learning. Once again in the cases of goals $\mathcal{B}$ and $\mathcal{C}$, we see a sharp decrease in the number of steps required to completion, e.g. for goal $\mathcal{B}$ the steps required for completion in the first episode are 581616 which converges to $\approx 28$ by the $1000^{\text{th}}$ episode, ref. Fig. 11. The strategy for reaching goal $\mathcal{C}$ with the stochastic wind on, is successful with `SARSA` also, ref. Fig. 15.

### 1.3.1 Goal $\mathcal{A}$

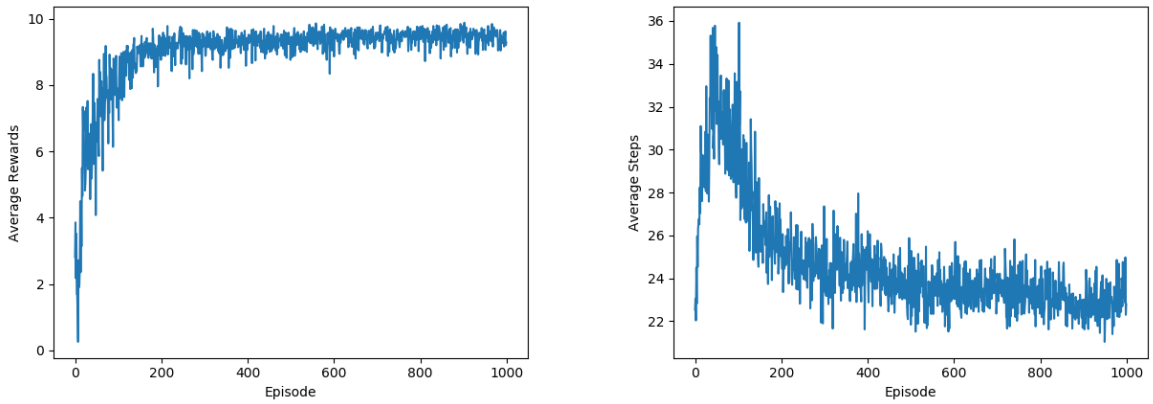



Figure 9: Rewards and steps to completion for `SARSA` applied to the puddle world with goal state $\mathcal{A}$

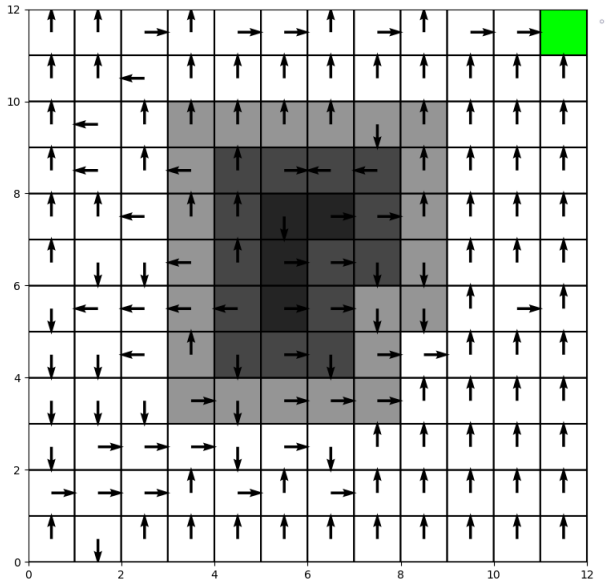


Figure 10: Optimal policy derived greedily from `SARSA` for goal $\mathcal{A}$

### 1.3.2 Goal $\mathcal{B}$

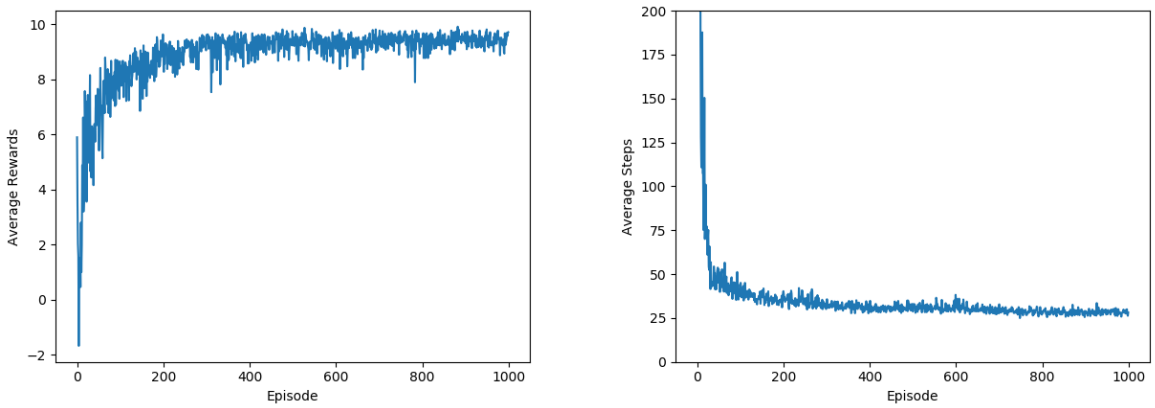



Figure 11: Rewards and steps to completion for `SARSA` applied to the puddle world with goal state $\mathcal{B}$

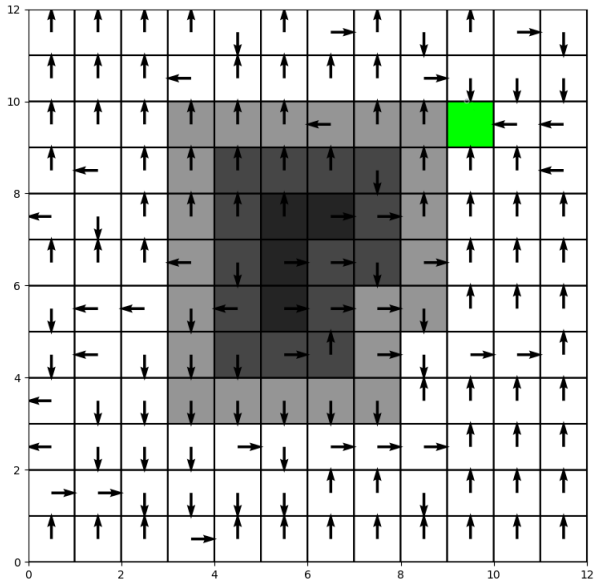


Figure 12: Optimal policy derived greedily from `SARSA` for goal $\mathcal{B}$

### 1.3.3 Goal $\mathcal{C}$

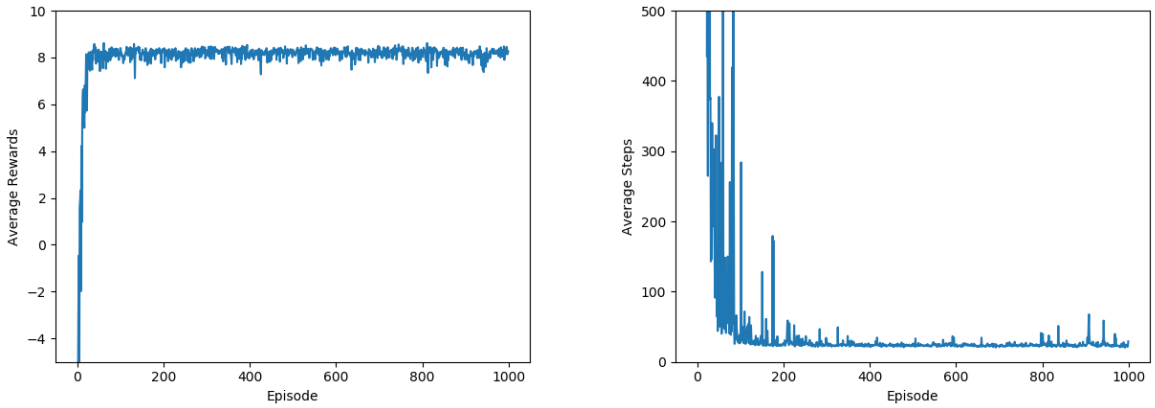



Figure 13: Rewards and steps to completion for `SARSA` applied to the puddle world with goal state $\mathcal{C}$

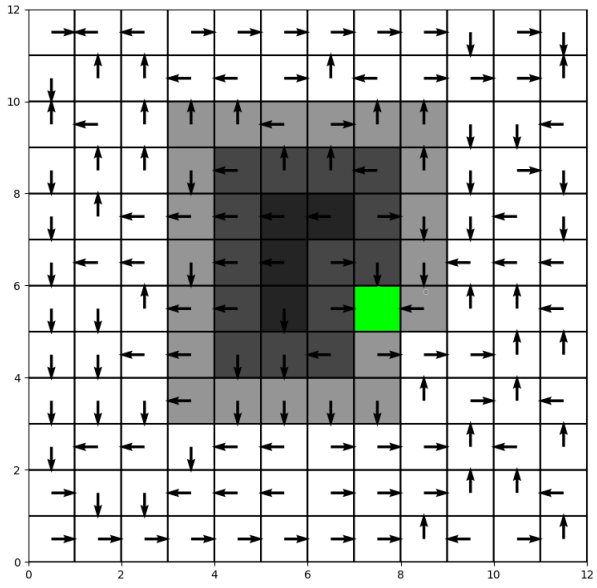


Figure 14: Optimal policy derived greedily from `SARSA` for goal $\mathcal{C}$

### 1.3.4 SARSA for reaching goal $\mathcal{C}$ with wind

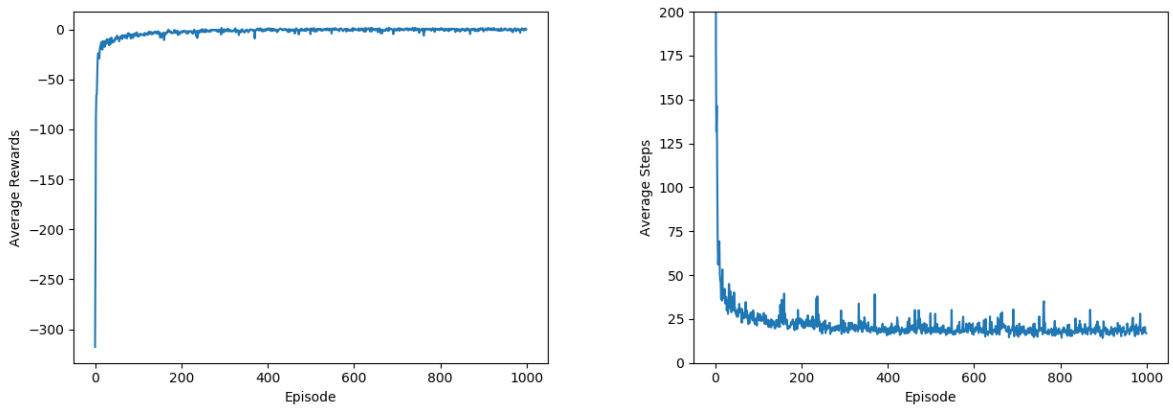



Figure 15: Rewards and steps to completion for SARSA applied to the puddle world with goal state $\mathcal{C}$ with wind

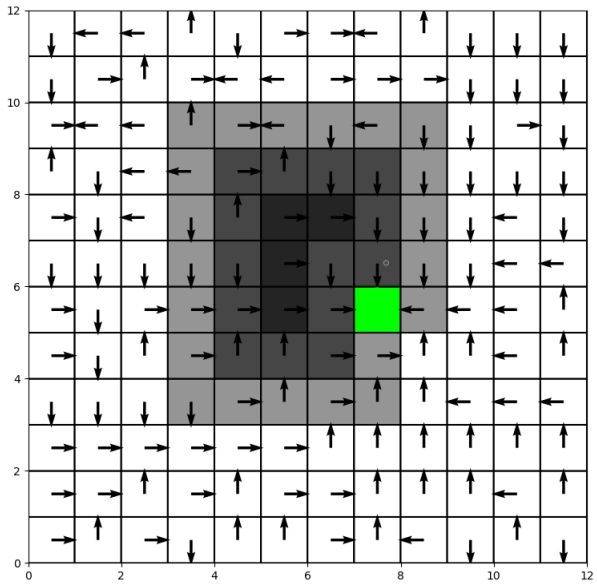


Figure 16: Optimal policy derived greedily from SARSA for goal $\mathcal{C}$ with wind

### 1.3.5 SARSA($\lambda$)

The experiment was only carried out for Goal $\mathcal{A}$ as the remaining goals took too long to terminate every episode. The results for SARSA($\lambda$) are shown in Fig. 17. There is not much of a difference in the learning curves for the different $\lambda$ values.

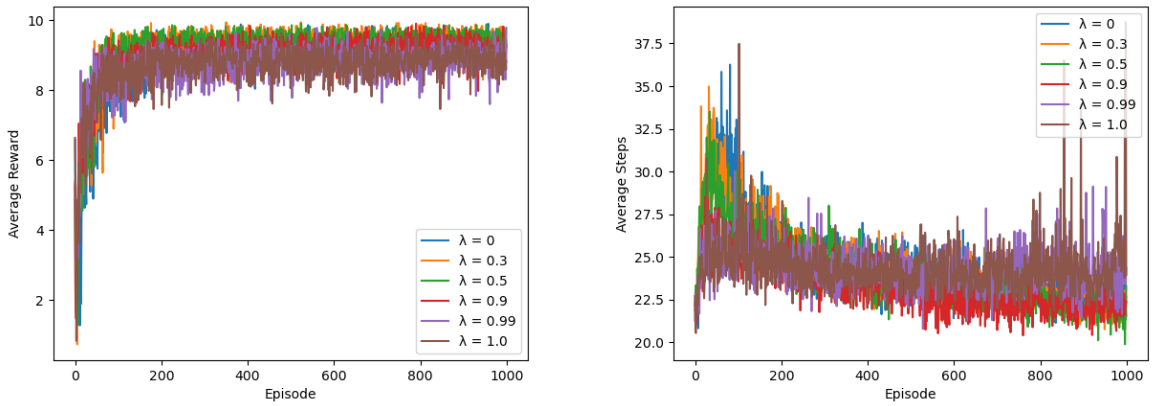



Figure 17: Rewards and steps to completion for `SARSA`($\lambda$) applied to the puddle world with goal state $\mathcal{A}$ with wind

# 2 Policy Gradient

## 2.1 Question 1

The policy gradient algorithm is very sensitive to hyper-parameters. The optimal set of parameters for `chakra` was found to be $(\alpha, \gamma, N) = (0.025, 0.7, 500)$, where $\alpha$ is the gradient ascent step size, $\gamma$ is the discount factor and $N$ is the batch size, i.e., the number of trajectories sampled per iteration. For `vishamC`, a lot of parameter settings were tried, ranging from $alpha \in \left[1, 10^{-13}\right]$, $gamma \in [0.01, 0.99]$ and $N \in [100, 1000]$, however none were shown to improve the learning process. The learning curves for all of the various parameter settings were as in Fig. 18.

The maximum allowed steps per trajectory is 40 and if $\sqrt{x^2 + y^2} < 10^{-3}$ then it is considered that the agent has reached the origin and the episode is terminated. The average rewards variation with number of iterations for `chakra` is shown in Fig. 18. Any other parameter setting causes the learning process to either stagnate, like in the case of `vishamC`, to get worse.

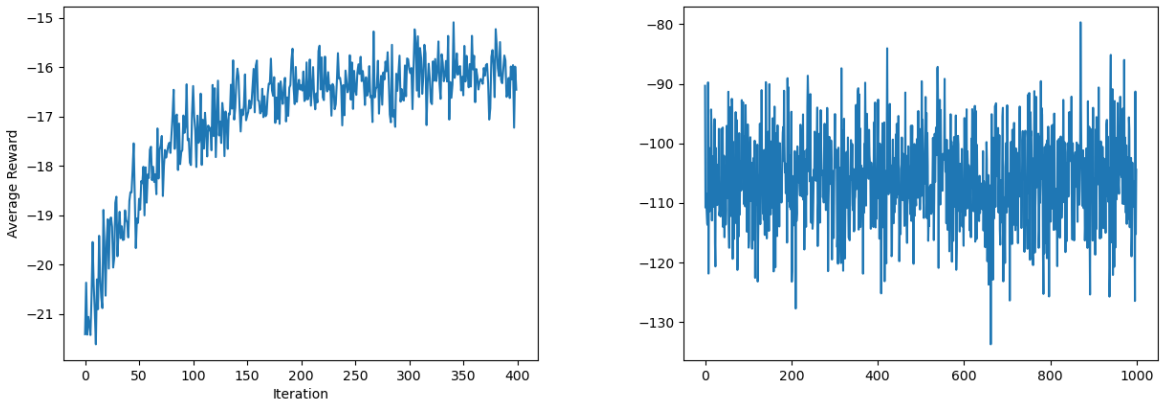
The variance in the learning curve can be reduced by increasing the `batch_size` parameter. This does not speed up the learning process. No comment can be made about the effect of $\gamma$ or $\alpha$ on the learning process, as any value greater or less than 0.7 does not give any improvement. Lastly, no baseline is used for either since for `vishamC` no improvement was being observed and for `chakra` adding a baseline did not improve the curve (both approximate Value Function and average rewards were tried).

# 3 Question 2

For learning a value function corresponding the policy, use a parameterized form $V(\boldsymbol{s}) = V(\boldsymbol{s}, \boldsymbol{w})$, and use SGD with Monte-Carlo returns to fit the parameters,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_w (U_t - V(\boldsymbol{s}, \boldsymbol{w})) \nabla_w V(\boldsymbol{s}, \boldsymbol{w})$$

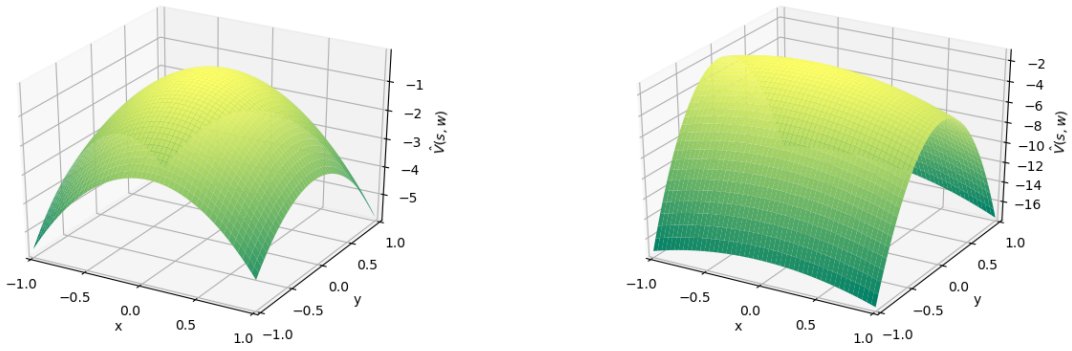
A polynomial basis is used, so that $\nabla_w V(\boldsymbol{s}, \boldsymbol{w}) = \boldsymbol{\phi}(\boldsymbol{s}) = \left[x, y, x^2, y^2, xy, 1\right]^T$. A learning rate of $\alpha_w = 0.001$ is used and the learning is run for 200 iterations with a batch size of 500 trajectories per iteration. The results are shown in Fig. 19.




(a) Learning curve for `chakra` with no baseline

(b) Learning curve for `vishamC` with no baseline

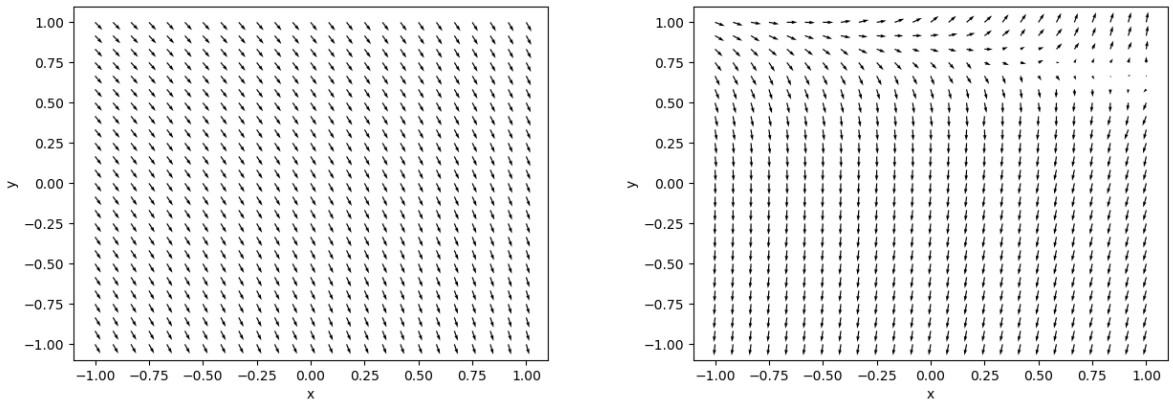
Figure 18: Learning curves for policy gradient




(a) `chakra`

(b) `vishamC`

Figure 19: Approximate Value Function for learnt policies




(a) `chakra` (b) `vishamC`

Figure 20: Learnt policies for both the worlds averaged over 50 runs

# 4 Question 3

The learnt policies are shown in Fig. 20. The policy was sampled and the results averaged 50 times at each point. Clearly, the learning process for both agents has not succeeded, as the policies always point in one direction. Thus, not much can be said about the policy learnt.