CSE546 – REINFORCEMENT LEARNING SPRING 2023

FINAL PROJECT

REPORT

Forest Fire Mitigation using MARL

Team 30:
Amit Kumar Dey
Kumar Raja Chidella
Tanwin Chowdary

Project Description:

- Objective: Learn best way to solve a forest fire simulation environment
- Develop RL-based forest fire grid setup to detect, prevent, and manage forest fires more effectively.
- Create a grid-based simulation environment to model forest fire dynamics and simulate different scenarios.
- Use multiple agents, such as firefighters and drones, that can move around the grid and take actions to contain and extinguish fires.
- Train RL algorithms to learn optimal firefighting strategies for different situations.
- Ultimately, we hope to reduce the damage caused by forest fires and save lives.

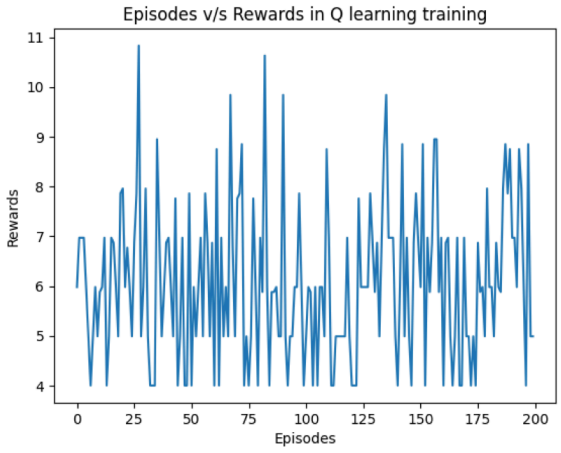
Env description:(pics of both type of env's)

- Observation space: n*n grid
- Actions : Move Up, Down, Left, Right
- Rewards:
  - -0.1 for visiting cell without fire
  - +2 for visiting cell with fire and dousing it
  - +5 for dousing all cells with fire
- Has stochastic and Deterministic variability.
- Stochastic Environment
  - Random position fires
  - Fire spread with random probability
- Deterministic Environment
  - Fire at Static positions
  - No spread of fire
    - Qtable with grid position where particular agent is in state
    - Qtable with values of all grid positions in state

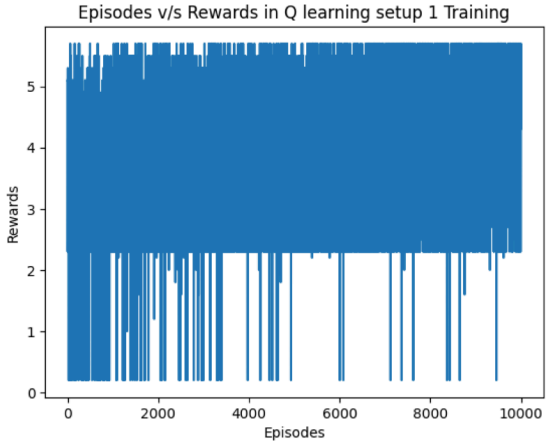
Q-learning:

- Implemented Q learning on the deterministic environment, where the initialized fire stays in that same cell and does not spread into its neighboring cells.
- Agents start at the opposite ends of the environment.
- For each run,the goal is to douse all the fires present in the environment.Once all the fire is doused the run resets.
- Each agent has its own qtable of actions which updates itself for each action and reward awarded.
- Later tested the environment by choosing the best action possible for a given state.
- Here are the results achieved:

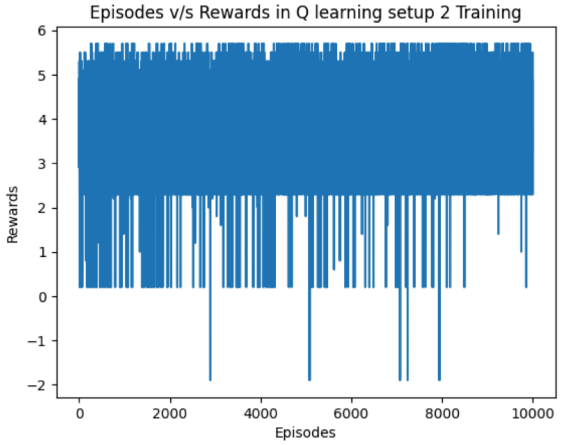
Stochastic Environment



Deterministic Environment 1



Deterministic Environment 2



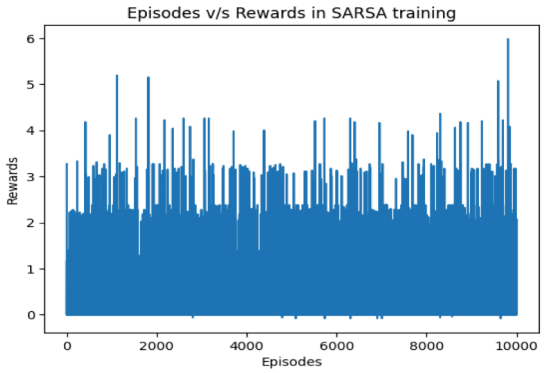
This is how the q-values look like for a given state:

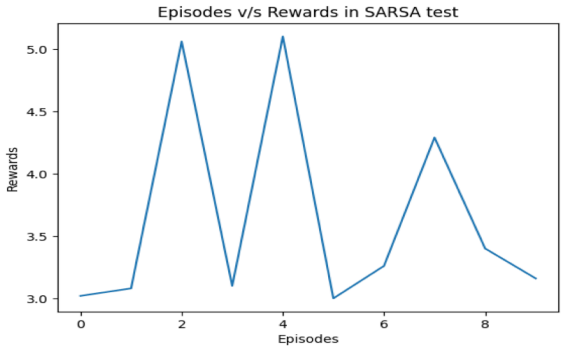
```
'000010010': array([0.2653758 , 0.43568479, 0.33503511, 0.56826015]),
```

SARSA:

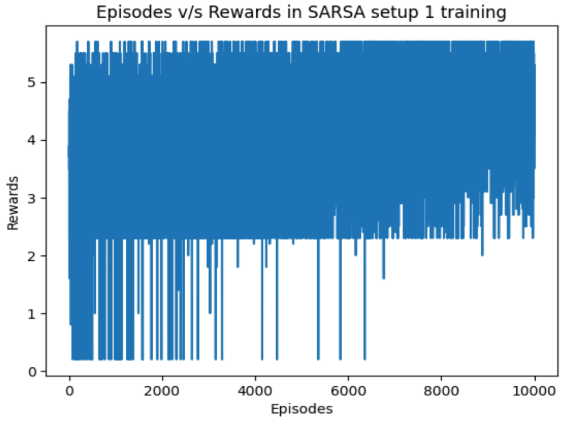
- We have followed a similar approach as Q-learning with the difference being how the q-values are calculated given the agent state, action and reward. And achieved the following reward.

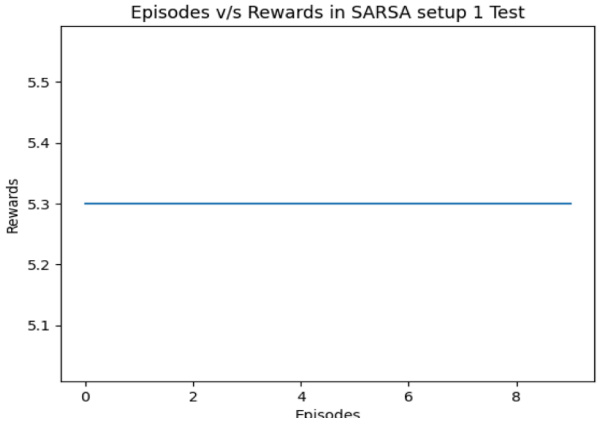
Stochastic Environment:



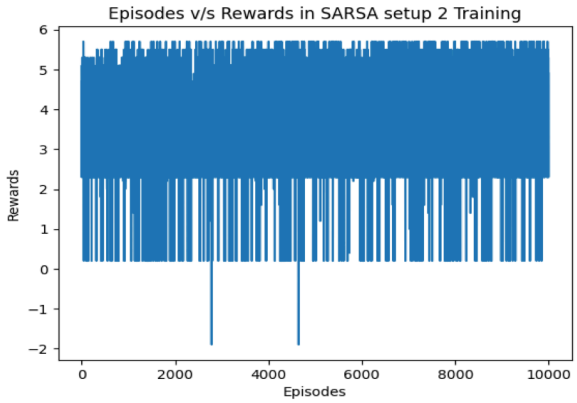


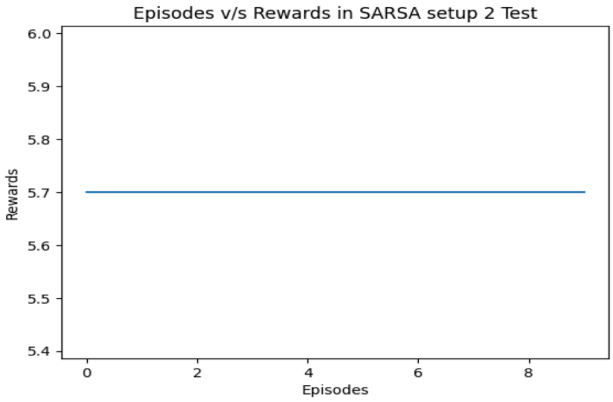
Deterministic Environment 1:





Deterministic Environment 2:

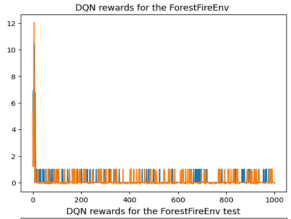


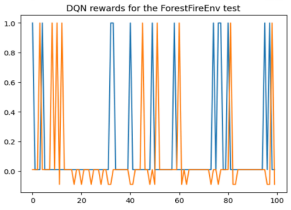


Deep RL Methods:

- DQN:
  - The DQNAgent class represents our agent in the DQN algorithm.
  - Its inputs are state_size, action_size, device, learning rate (lr), discount factor (gamma), buffer size, and batch size as input parameters.
  - The agent has a neural network model that predicts the Q-values of each possible action for a given state.
  - The agent uses an experience replay buffer to store experiences (state, action, reward, next_state, done) during training.
  - The agent chooses an action using an epsilon-greedy policy, where it selects a random action with probability epsilon and selects the action with the highest Q-value with probability (1-epsilon).
  - The agent updates the target network every update_frequency steps.
  - The agent trains its neural network by sampling a random batch of experiences from the replay buffer and updating the model's parameters using the Bellman equation.
  - We have used the Adam optimizer to update its model's parameters for the agent.

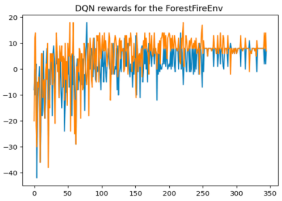
DQN stochastic v/s static





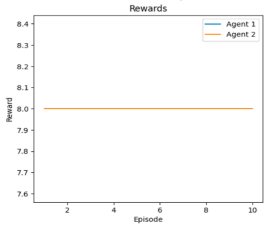
Due to the volatility of the environment defined the results of the DQN implemented are no as expected with max timesteps of 1000 on a 6x6 grid env the results are [-0.1,-0.1] for the agents.

And because of the training handling the volatile env the test results are fluctuating for the stochastic env for the 100 episodes.



Due to the stability of the environment defined the results of the DQN implemented are quite impressive with max timesteps of 100 on a 4x4 grid env the results are stabilizing at [8, 8] for the agents.

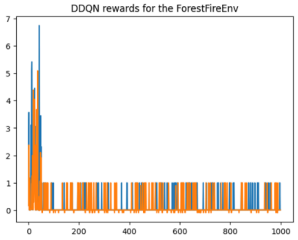
DQN Test Results:



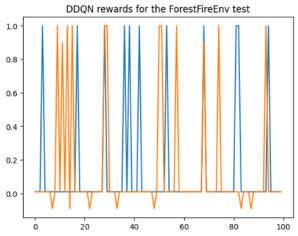
And because of the training handling the stable env the test results are constant for the static env for the 10 episodes. Results are achieved in 4 timesteps.

- DDQN:
  - It is an improvement over DQN, which reduces overestimation of Q-values.
  - It involves using two separate neural networks to estimate Q-values: the online model and the target model.
  - The online model is used to select actions, while the target model is used to evaluate the actions.
  - In DDQN, the target Q-values are computed using the target model, but the action selection is done using the online model.
  - This is achieved by selecting the action with the highest Q-value using the online model, and then evaluating that action using the target model to obtain the target Q-value.
  - This helps to mitigate the problem of overestimation of Q-values that is often observed in DQN.
  - The code for DDQNAgent is similar to that of DQNAgent, but with modifications in the train() method to compute the target Q-values using the target model and the online model.
  - The DDQN code presented here trains the agents using the ForestFireEnv environment of both deterministic and stochastic in nature.
  - The training loop is similar to that of DQN, but with the use of two separate models to estimate Q-values.

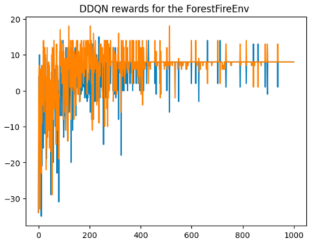
DDQN stochastic v/s static



Due to the volatility of the environment defined the results of the DDQN implemented are no as expected with max timesteps of 1000 on a 6x6 grid env the results are [0.1, 1] for the agents.

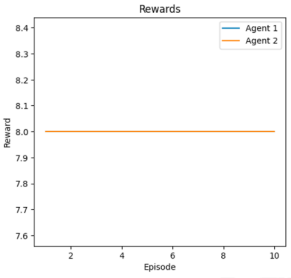


And because of the training handling the volatile env the test results are fluctuating for the stochastic env for the 100 episodes.



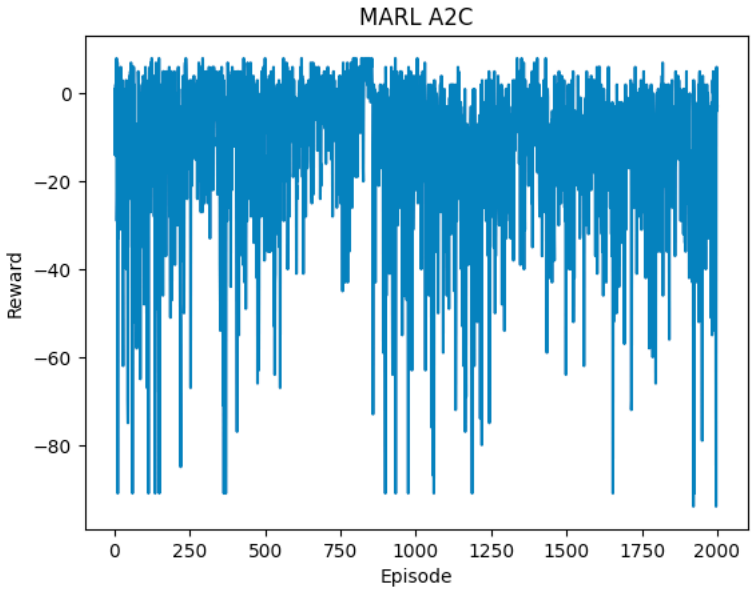
Due to the stability of the environment defined the results of the DDQN implemented are quite impressive with max timesteps of 100 on a 4x4 grid env the results are stabilizing at [8, 8] for the agents.

DDQN Test Results:



And because of the training handling the stable env the test results are constant for the static env for the 10 episodes. Results are achieved in 4 timesteps.

- **A2C:**
  - Two fully connected layers (64 units each),Actor head outputs policy logits, Critic head outputs state-value estimate
  - Calculates policy and state-value using current state, Calculate policy and next state-value using next state
  - Compute advantage estimation (delta) using rewards, state-values, and discount factor
  - Calculate actor loss as negative log probability * detached delta,Calculate critic loss as squared delta, Total loss is the sum of actor loss and 0.5 * critic loss and Perform backpropagation and optimization step
  - Then we Initialize multiple ActorCritic agents and Adam optimizers.
  - For each episode: it reset environment and duplicate state for each agent, and while episode is not done: for each agent, sample an action using policy and execute actions, observe next states, and rewards and update each agent using state, action, reward, next state, and done flag and update total rewards and states.
  - And record average reward per episode and print results



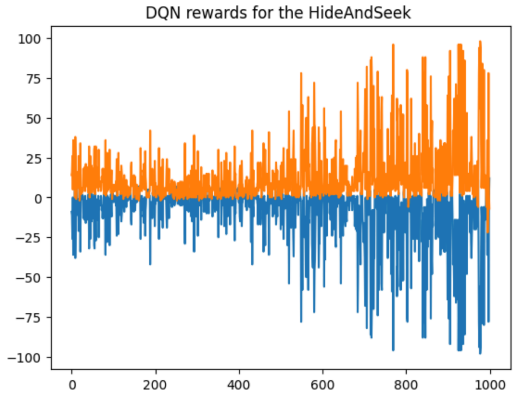
Hide and Seek, DQN:

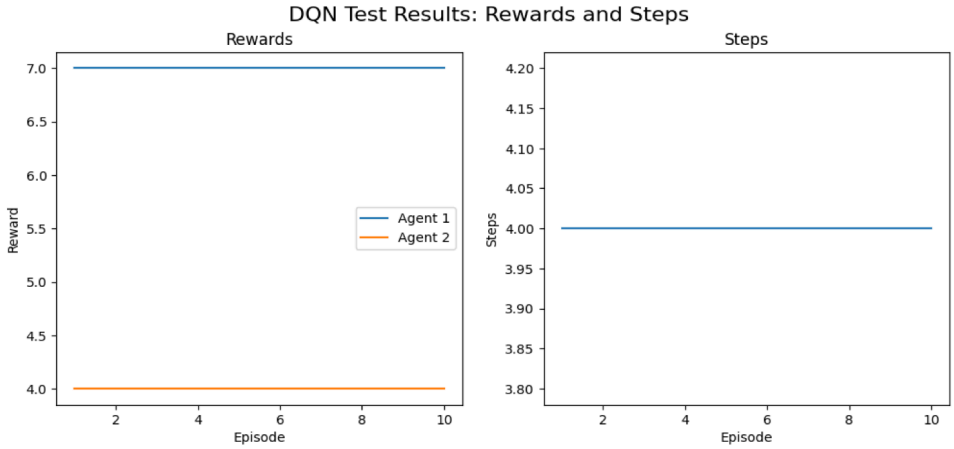
One of the best performances we got on our ForestFireEnv is using DQN. So we have decided to use DQN for the Hide and seek environment.

Environment:

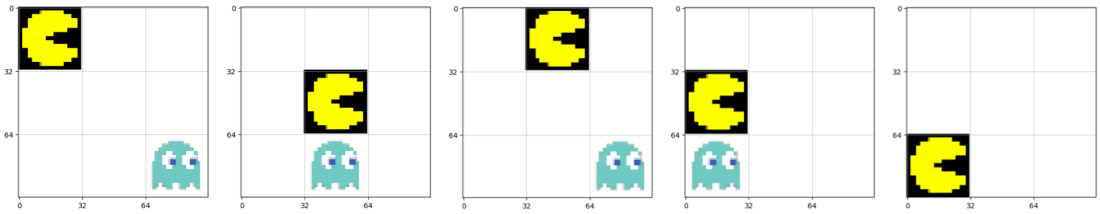
- Two agents at the corners of the grid of the environment. (The size of the environment can be changed with grid_size)
- One is Seeker and the other is Hider, represented by ‘pacman’ and ‘ghost’.
- Depending on how far apart the agents are the rewards are calculated. If the seeker gets closer to hider’s position the seeker gets some positive reward and hider is rewarded with some negative reward and vice versa.
- To avoid halting, a negative reward is awarded for each timestep for each agent.
- On finding the hider the seeker gets a huge reward and the hider is awarded a lower reward.
- Agents do learn to perform better as shown in the graphs, agents attain higher rewards as they perform their actions better.

Results:





One of the Test epoch of the agents:



Summary:

- Challenges we faced:
  - Working in a MARL setup has been difficult initially as our first time encountering it in the course and exploring how it works has been fascinating.
  - Handling stochastic environment for tabular and deep RL methods.
  - Difficulty in implementing entire existing MARL problem like OpenAI Particle or HIDE&SEEK by OpenAI
- Observations:
  - Need a more comprehensive reward structure
  - SARSA and Q-Learning seem to converge but only in deterministic setup since they are tabular methods
  - DQN and DDQN seem to provide most optimal solutions out of all deep RL methods that we tried
  - A2C not suitable for our environment
  - It takes a complex RL algorithm which can adapt to any random events and process huge amount of information and steadily output a reliable solution would be needed. So exploring multiple algorithms to find which can be best suited for this task has been a good challenge to test our skills on the given subject.

References:

- Class notes and Piazza
- Assignment reference document
- Pytorch Documentation
- OpenAI Gym Documentation
- https://github.com/msinto93/DDPG
- https://github.com/nikhilbarhate99/PPO-PyTorch
- "Wildfire Detection using Deep Learning and Satellite Imagery" by J. He and H. Zhang. This paper proposes a deep learning approach to detecting wildfires using satellite imagery. The authors achieved an accuracy of 92% on their test set. DOI: 10.1109/JSTARS.2020.3043111

Contribution:

Though the work was assigned among teammates, each task has been done with the involvement of each teammate.

Name	Parts Contributed	Percentage
Amit Kumar Dey (adey2)	Env, Q_learning, SARSA, ppt	33.33%
Tanwin Chowdary Gunturu (tanwinch)	Env, DQN, DDQN, ppt	33.33%
Kumar Raja Chidella (kumarraj)	Env, A2C, Hide and seek, ppt	33.33%

THANK YOU