# Parallel Double Cart Pole
# 239AS Project Report S2021


**Samuel Gessow**
sgessow@ucla.edu
604781350

**Sunay Bhat**
sunaybhat1@ucla.edu
905629072

**Yi-Chun Hung**
yichunhung@ucla.edu
705428593

**Vahe Gyuloglyan**
vgyulogl@ucla.edu
905528327



## Abstract

This report explores using reinforcement learning on a novel multi-cart version of the standard cart-pole controls environment. In this extension of cart-pole, there are two carts that operate two poles, attached at their ends together, to support a pendulum. This extension allows us to explore the standard cart-pole problem with considerably more complex dynamics. We explore the application of the DQN, DDQN, Policy Gradient, Actor-Critic, and Rainbow algorithms in order to solve this problem. We found that DDQN, Actor-Critic, and Rainbow are able to solve the problem with reasonable performance, with DDQN giving the best results. Finally, we explored a multi-agent version in which each cart is an independent agent, thus allowing for more complex reward structures and dynamics beyond the base multi-cart problem.


## 1 Introduction

The original cart-pole problem is well studied in reinforcement learning (RL) [1], and our extension will allow us to explore more advanced dynamics. Using Pymunk for the physics simulation and OpenAi Gym for the RL framework, we developed the "carts-poles" test environment. An illustration of the environment can be seen in Figure 1 below. There are twelve total state variables (2 per cart, 2 per pole). In Figure 1, a rendering of our completed environment can be seen as well.

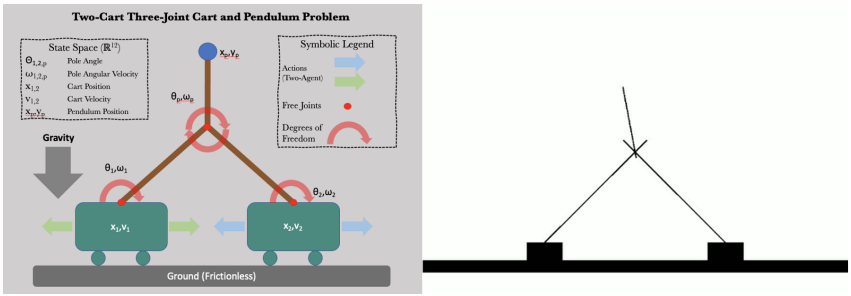



Figure 1: The illustration and render of our environment, including two carts, three joints, and a pendulum.

We decided to explore our novel cart-pole as there exist extensions on the traditional cart-pole, but none which can expand into a multi-agent system. These include the double [2] and triple pendulums [3]. The dynamics of the real world often include complex systems which have high degrees of non-linearity and multiple "agents" that operate on imperfect or minimal information. Our goal is to begin to explore such dynamics with the state-of-the-art RL techniques and discuss our results.

Based on the definition of success in previous cart-pole problems, we define a successful completion as a method that can stabilize the pendulum for an average of 100 seconds over all the starting angles


35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

from -12 to 12 degrees. We put a maximum length of 200 seconds in both the training and evaluation episodes to prevent infinite runs.

## 2 Environments

We made two different environments following the OpenAI gym environment format. The first 'carts_poles.py' designed for a single agent, and the second 'carts_poles_2agent.py' was very similar to the first but can accommodate two agents with variable amounts of information.

In carts_poles.py the agent can execute 1 of 9 actions consisting of combinations of applying a left, a right, or no force on each cart. This action was applied every 'dt' seconds which was set as 1/100, but this could be changed. For our purposes, we kept the time-step the same as a 100Hz controller seemed like a reasonable condition. The environment also controlled the ending condition and rewards for the agents. The ending condition was set using the following three criteria: 1) The carts moved more than 2.5 meters from the starting condition, 2) The upper pendulum moved more than $\pi/8$ radians from vertical 3) The y-coord of the upper pendulum pole fell below .15 meters. If the run was not done the environment returned a reward based on the time that the agent was alive. Although this is coded into the environment we explored changing the reward function for certain methods to achieve better results. After every step the environment returned a state vector as follows:

$$State = [x_1, v_1, x_2, v_2, \theta_1, \omega_1, \theta_2, \omega_2, \theta_p, \omega_p, x_p, y_p]$$

where the states are defined in Figure. 1, and the action space is defined as follows:

$$Action = [(left, left), (left, right), (left, none), (right, left), (right, right), (right, none),$$
$$(none, left), (none, right), (none, none)]$$

In carts_poles_2agent.py we created a new environment that accommodates two agents. Each agent independently chooses one of three actions: left, right, or no force. The new environment accepts two actions and returns two states. This allowed for different agent information states. In the 'full' information state, both agents receive the full environment state as in the single agent problem, and in the 'partial' information state, each agent receives everything but the values associated with the other cart. The time step, ending conditions, and initial reward structure was the same as the single agent problem.

$$'Full'\ Information\ State$$

$$State\_1,\ State\_2 = [x_1, v_1, x_2, v_2, \theta_1, \omega_1, \theta_2, \omega_2, \theta_p, \omega_p, x_p, y_p]$$
$$Action\_1 = [(left), (right), (none)]$$
$$Action\_2 = [(left), (right), (none)]$$

$$'Partial'\ Information\ State$$

$$State\_1 = [x_1, v_1, \theta_1, \omega_1, \theta_p, \omega_p, x_p, y_p]$$
$$State\_2 = [x_2, v_2, \theta_2, \omega_2, \theta_p, \omega_p, x_p, y_p]$$
$$Action\_1 = [(left), (right), (none)]$$
$$Action\_2 = [(left), (right), (none)]$$

# 3 Methods

## 3.1 DQN

The following algorithm outlines the DQN methodology that was applied to the system.[4]

**Algorithm 1:** DQN

Algorithm parameters: learning rate $\alpha \in (0, 1]$, discount factor $\gamma \in [0, 1]$, small $epsilon > 0$;
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$;
**foreach** *episode* **do**
    Initialize S;
    **foreach** *step of episode* **do**
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., ε-greedy);
        Take action $A$, observe $R$, $S'$;
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$;
        $S \leftarrow S'$;
    **end foreach**
**end foreach**

## 3.2 Policy Gradient

The following is the algorithm we used for Policy Gradient. We parameterized $\pi$ using a neural network. [5]

**Algorithm 2:** REINFORCE

Algorithm parameters: A differential policy parameterization $\pi(a|s, \theta)$ step size $\alpha \in (0, 1]$ ,
$\gamma \in [0, 1]$;
Initialize $\theta \in \mathbf{R}^{d'}$;
**foreach** *episode* **do**
    Generate and episode $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$, following $\pi(\dot{|}\dot{;}\theta)$ ;
    **foreach** *step of episode* **do**
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$;
        $\theta \leftarrow \theta + \alpha\gamma^t G \nabla ln\pi(A_t|S_t, \theta)$
    **end foreach**
**end foreach**

### 3.3 Actor Critic

The following is the algorithm implemented for Actor-Critic [5]. We function approximated the policy and value function using a Deep Neural Network for the actor and critic respectively.

**Algorithm 3:** Actor-Critic (One-Step) $\pi \approx \pi_*$

Actor: Deep Neural Network that converts a state to action;
Critic: Deep Neural Network that converts a state to value;
Algorithm parameters: step size: $\alpha^A > 0$, $\alpha^C > 0$, discount factor: $\gamma \in [0, 1]$;
Initialize Actor and Critic with weight = 0;
**foreach** *episode* **do**
  Initialize S;
  **foreach** *step of episode* **do**
    Get $A$ from $S$ by sampling Actor policy $A \sim Actor(S)$;
    Take action $A$, observe $R$, $S'$;
    $\delta \leftarrow [R + \gamma Critic(S') - Critic(S)]$;
    $Critic(S) \leftarrow Critic(S) + \alpha^C \delta \nabla Critic(S)$;
    $Actor(S) \leftarrow Actor(S) + \alpha^A \delta \nabla Actor(S)$;
    $S = S'$
  **end foreach**
**end foreach**

### 3.4 DDQN

The following algorithm outlines the DDQN method we applied to the system [6]

**Algorithm 4:** DDQN

Algorithm parameters: learning rate $\alpha \in (0, 1]$, discount factor $\gamma \in [0, 1]$, small $epsilon > 0$,
DQN sync rate *copy*;
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that
$Q(\text{terminal}, \cdot) = 0$;
**foreach** *episode* **do**
  Initialize S;
  **foreach** *step of episode* **do**
    Choose $A$ from $S$ using policy derived from $Q_1$ (e.g., ε-greedy);
    Take action $A$, observe $R$, $S'$;
    $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha[R + \gamma \max_a Q_2(S', a) - Q_1(S, A)]$;
    **if** *step % copy == 0* **then**
      $Q_2(S, A) \leftarrow Q_1(S, A)$
    **end if**
    $S \leftarrow S'$;
  **end foreach**
**end foreach**

### 3.5 Rainbow

Rainbow [7] is one of the most successful Q-learning method based on different improvements in deep reinforcement learning. It includes double Q-learning, prioritized replay, dueling networks, multi-step learning, distributional RL and noisy nets [8, 9, 10, 11, 12, 13]. For distributional RL and prioritized replay, $V_{min}$, $V_{max}$, $N_{atoms}$, buffer capacity, $\alpha$ and $\beta$ are set as $-1$, 1, 51, $10^6$, 0.5 and 0.4. We also set the two layered fully connected model with distributional dueling head and 512 hidden nodes.

# 4 Results

## 4.1 DQN

We found the DQN algorithm to be highly volatile and unstable. This was exhibited in two ways. First, the DQN would exhibit catastrophic forgetting and the performance would go to the unlearned state as seen in figure 2a [14]. In addition, even at some of the peaks the performance was not good when tested on multiple start angles. Second, during some runs the agent would achieve excellent performance as seen in figure 2b and during others it would never achieve perfromance as good as seen in figure 2c. To improve on this we implemented the DDQN and rainbow algorithms.

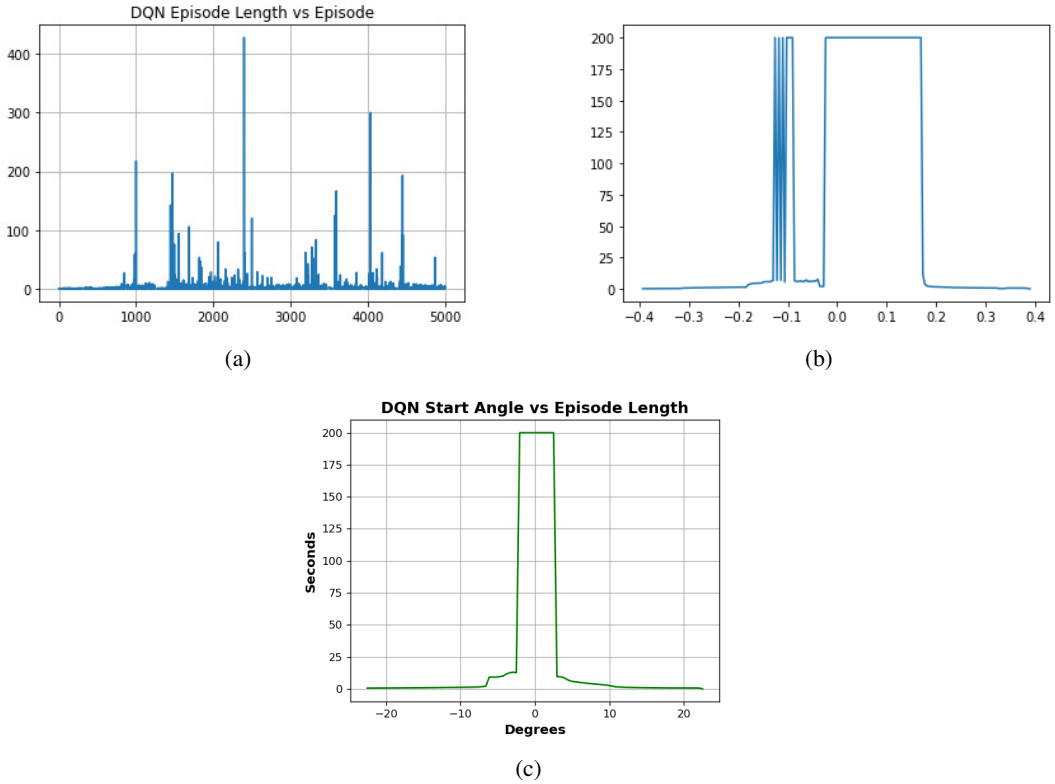


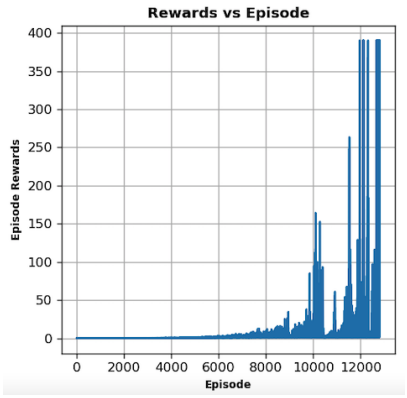

Figure 2: DQN results. (a) Example of DQN training. (b) Best DQN performance across different starting angles. (c) Typical DQN performance across different starting angles.

## 4.2 Policy Gradient

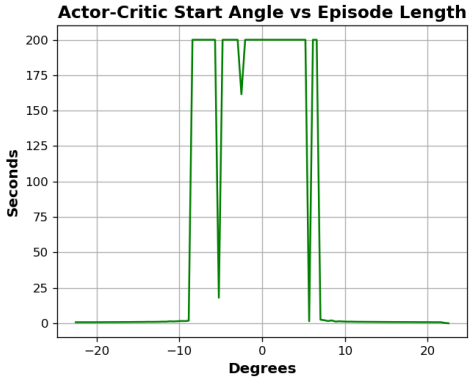
We found the policy gradient algorithm to be ineffective at solving this environment. Using policy gradient, the performance did increase over time, but it never achieved performance better than 1 sec even after hours of training. For this reason we stopped exploring the policy gradient algorithm and focused on actor critic.

## 4.3 Actor Critic

The Actor-Critic algorithm was quite stable in terms of learning progress, but it was also very slow. Despite tweaking the learning rate, we found the algorithm rarely exceeded a few seconds until at least 10,000 episodes, and then training time was quite significant. Despite all this, the algorithm was successful at meeting the goal, with a mean episode length of 123 seconds between -12 and 12 degrees. Figure 3 shows the training log and results. Although it is possible additional training might further improve this result, the payoff after achieving max episodes was vastly diminished for large amounts of training time.




(a)

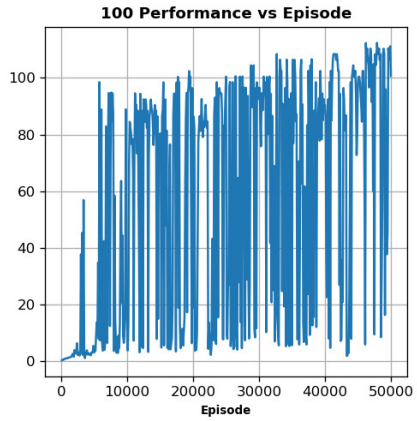



(b)

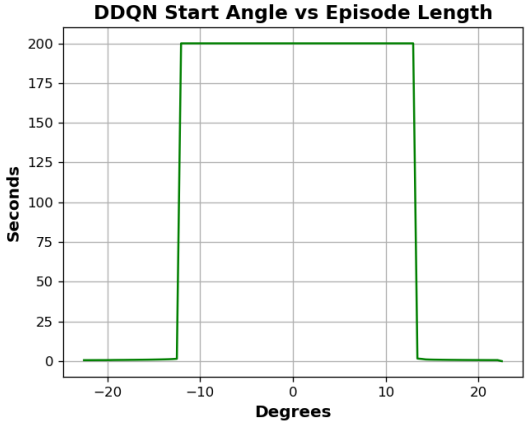
Figure 3: Actor-Critic results. (a) Example of Actor-Critic training log with respect to episode. (b) Typical Actor-Critic performance across different starting angles.

### 4.4 DDQN

The DDQN was the best performing algorithm in terms of the final results. Like DQN, DDQN also exhibited bouts of forgetting, but unlike DQN, would recover and continue learning quickly. In addition, the DDQN would generate good results every training run, although the amount of episodes would vary. Typically runs would require on the order of 4,000 episodes to achieve excellent performance. The results plot in figure 4b shows the exceptional performance in the target range.




(a)

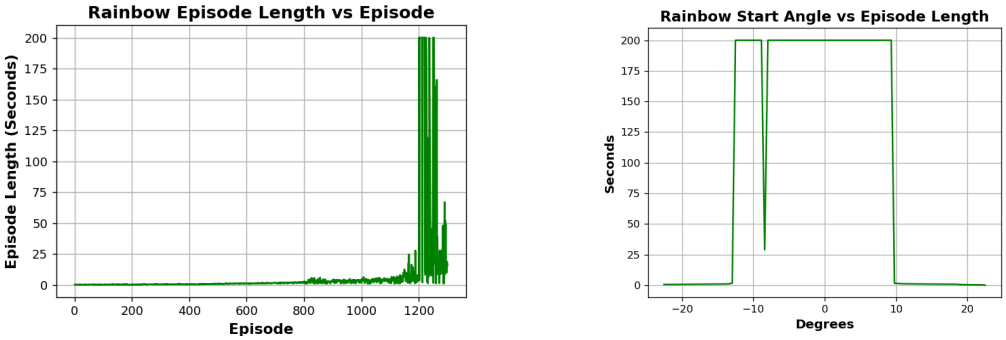



(b)

Figure 4: DDQN results. (a) Example of DDQN training log with respect to episode. (b) Typical DDQN performance across different starting angles.

### 4.5 Rainbow

As shown in fig. 5a, Rainbow performed better than DDQN in the training stage. It only required approximately 1400 episodes, which was less than half of training episodes for DDQN to balance the pole effectively. With a small probability, it could still fail to balance the poles as shown in the Fig. 5b. In addition to the training, the control model with Rainbow gave a different control behavior than DDQN and Actor Critic. The Rainbow model seemed to balance the poles with a quick back and forth motion and maintained the cart in the left hand side of the viewing window. We believe this is due to the design of the reward function. Since the function does not encourage the cart to remain in the middle of the window, the carts can be anywhere in the window once it gets the pole balanced.




(a) (b)

Figure 5: Rainbow results. (a) Example of Rainbow training log with respect to episode. (b) Typical Rainbow performance across different starting angles.

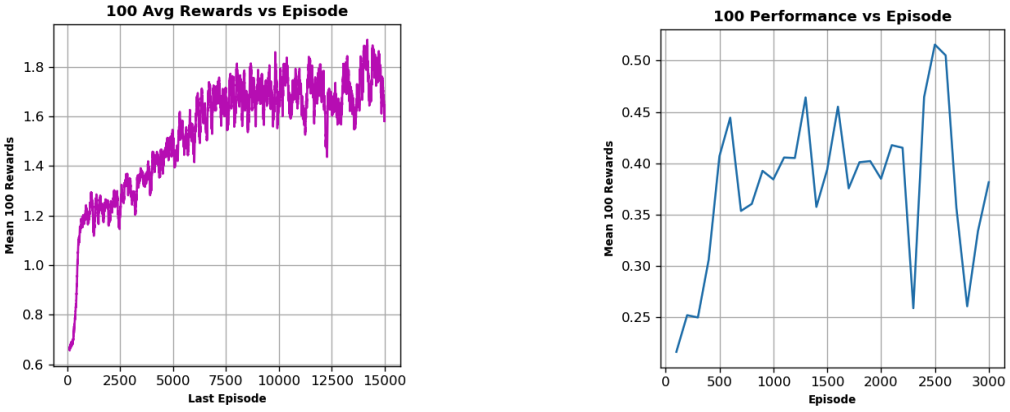
### 4.6 Table of Results

Table 1: The average performance of methods (Mean -12 to 12 degrees).

| DQN | DDQN | Actor Critic | Rainbow |
|---|---|---|---|
| 46.34 | **200.01** | 123.24 | 176.09 |

## 5 Multi-Agent

After establishing multiple algorithms could solve the single-agent problem, we then proceeded to the two-agent problem. We began with the 'full' information, and applied the actor-critic algorithm. This was done simply by having a second actor and a second critic with both actors choosing one of the three available actions. We also tried a two agent DDQN [15] method in which again both DQN network were doubled and operated with 'full' information but two separate action spaces. As can be seen in Figure 6 below, both the Actor-Critic and DDQN two-agent algorithms show no learning.




(a) (b)

Figure 6: Two-Agent Results. (a) Actor-Critic training run. (b) DDQN training run.

We believe the two-agent environment is too complex for any of the algorithms we implemented on the single agent. In order to deal with this added complexity, we attempted a series of experiments and algorithmic changes to see if any synergistic [16] behavior could emerge. We began by using extensive reward shaping [17], intrinsic fear [14], and intrinsic motivation [16] in multiple combinations. The reward shaping was largely focused on giving the agents multiplicative and exponentially higher

rewards in more granular time steps, but this did not seem to have any impact. The intrinsic rewards utilized joint probability considerations where large errors in expected outcome vs actual outcome are rewarded when the two agents take two actions over only one of them taking an action. The intuition behind this is to reward the agent for surprising itself, assuming this generally is an indication of possible synergy. But many of these methods are targeted at sparse, binary reward environments, which ours is certainly not. Finally, we tried increasing the layers and complexity of our neural networks, along with adding non-linear combinations of our state space into the network inputs. We again saw no learning progress. Unsurprisingly, no progress was made in the 'partial' information state with all the above techniques. The two-agent cart-poles environment remains entirely unsolved and an exciting future challenge.

## 6 Future Work

The application space of Reinforcement Learning algorithms continues to grow into exciting new areas. This study offers another example of the capabilities of the latest algorithms to take on complex, non-linear control environments. At the same time, much work is left to be done. Only the DDQN algorithm was able to reach the max episode time every episode. All of the other algorithms could use additional research into stability and convergence to determine if only more training is needed to also achieve such performance. Furthermore, we are unsure if the current goal window is a limitation of the physics and time-step, or if performance could be increased to a wider window with the right techniques. Many of the cart-pole problems address a 'swing-up' version in which the pole starts down and is swung up using the right set of actions to vertical and then balanced [1, 2]. This could also be a natural extension of the basic random starting angle problem we addressed.

In regards to the multi-agent environment, much research remains. We spent extensive time exploring exciting research in multi-agent reinforcement learning addressing increasingly complex and non-linear environments. We are unsure if any of the existing algorithms alone, or in combination, could show promise in the two-agent environment with enough training. Problems like this are the next boundary for which reinforcement learning algorithms need to address to further establish their viability for such controls applications.

## 7 Conclusion

The algorithms we implemented showed varying degrees of success, which is not surprising considering the complexity of the problem. DQN was the only fully implemented algorithm that did not meet the goal criteria. It showed poor stability and did not indicate signs of convergence in reasonable time, often due to 'catastrophic' forgetting. Actor-Critic was the next successful algorithm which did meet the goal requirements. The algorithm showed much more stable learning, indicating possible convergence, but the training time was relatively extensive. The Rainbow algorithm was very successful in terms of the speed of training needed (only a few thousand episodes), but it was not 100% successful in the target range. Only the DDQN algorithm was 100% successful in the target angle range, and it required moderate training.

This paper provides an extensive study on the ability of current state of the art RL algorithms to address our novel take on the traditional, linear cart-pole problem. The success of multiple algorithms is a promising sign for control applications and the complexity these algorithms can handle. At the same time, the two-agent problem shows how much work there is left to do. The current vanguard of Reinforcement Learning is already well equipped for high complexity, but multi-agent problems and synergistic behavior will be an exciting research space in the coming years.

## References


[1] Swagat Kumar. Balancing a cartpole system with reinforcement learning - A tutorial. *CoRR*, abs/2006.04938, 2020.

[2] Fredrik Gustafsson. Control of inverted double pendulum using reinforcement learning. 2016.

[3] Tobias GlüCk, Andreas Eder, and Andreas Kugi. Swing-up control of a triple pendulum on a cart with experimental validation. *Automatica*, 49(3):801–808, March 2013.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[6] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[7] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017.

[8] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[10] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[11] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[12] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[13] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[14] Zachary C. Lipton, Jianfeng Gao, Lihong Li, Jianshu Chen, and Li Deng. Combating reinforcement learning's sisyphean curse with intrinsic fear. *CoRR*, abs/1611.01211, 2016.

[15] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. Deep q-network based multi-agent reinforcement learning with binary action agents. *CoRR*, abs/2008.04109, 2020.

[16] Rohan Chitnis, Shubham Tulsiani, Saurabh Gupta, and Abhinav Gupta. Intrinsic motivation for encouraging synergistic behavior. *CoRR*, abs/2002.05189, 2020.

[17] Yujing Hu, Weixun Wang, Hangtian Jia, Yixiang Wang, Yingfeng Chen, Jianye Hao, Feng Wu, and Changjie Fan. Learning to utilize shaping rewards: A new approach of reward shaping. *CoRR*, abs/2011.02669, 2020.

## A Code

All code and plots can be found at the following Github repository:
https://github.com/SunayBhat1/Double-CartPole-239AS-UCLA-Project.

## B Contributions

Sam: carts_pole environment, DQN, DDQN, policy gradient
Sunay: Actor Critic, carts_poles_2agent
Yi-Chun: Rainbow, policy gradient
Vahe: DQN, DDQN

We feel that everyone contributed equally, the above only indicates what parts everyone contributed primarily to.