# Cart Pole in Gymnasium

**1. Introduction to Cart Pole Problem**

Within the field of reinforcement learning, the Cart Pole problem stands as a foundational and widely recognized benchmark. The environment consists of a cart that can move along a frictionless horizontal track. A pole is attached to the cart by an un-actuated joint, allowing the pole to pivot freely in the vertical plane. The objective is to apply forces to the cart (by moving it left or right) in such a way that the pole remains balanced upright.

**1.1. Observation Space**

In the context of reinforcement learning, the observation at each timestep provides the agent with information about the current situation, which it can then use to decide on the next action.

The Cart Pole-v1 environment features an observation space that is represented as a Box space with a shape of (4,). These four components of the observation vector provide information about the state of the cart and the pole:

* **Cart Position**: The first element of the observation vector (index 0) represents the horizontal position of the cart along the frictionless track. The values for this component typically range from approximately -4.8 to 4.8. However, a critical aspect of the environment is that an episode will terminate if the cart's position exceeds the more restrictive range of -2.4 to 2.4.
* **Cart Velocity**: The second element (index 1) indicates the velocity of the cart along the track. This value represents how fast the cart is moving and in which direction. The velocity can theoretically range from negative infinity to positive infinity, although in practical scenarios, the observed values will be within a finite range.
* **Pole Angle**: The third element (index 2) represents the angle of the pole with respect to the vertical, measured in radians. A value of 0 indicates that the pole is perfectly upright. The pole angle typically ranges from approximately -0.418 to 0.418 radians, which corresponds to about ±24 degrees. Similar to the cart position, there is a termination condition based on the pole angle. If the angle goes outside the range of -0.2095 to 0.2095 radians (±12 degrees), the episode will end.
* **Pole Angular Velocity**: The fourth and final element (index 3) represents the rate of change of the pole's angle, also known as the angular velocity. It indicates how fast the pole is rotating and in which direction. Like the cart velocity, the pole angular velocity can theoretically range from negative infinity to positive infinity.

|  |  |  |  |
| --- | --- | --- | --- |
| Observation Index | Observation Description | Approximate Range | Termination Range |
| 0 | Cart Position | [-4.8, 4.8] | [-2.4, 2.4] |
| 1 | Cart Velocity | [-3.4e+38, 3.4e+38] (effectively unbounded) | N/A |
| 2 | Pole Angle | [-0.418, 0.418] radians (±24°) | [-0.2095, 0.2095] radians (±12°) |
| 3 | Pole Angular Velocity | [-3.4e+38, 3.4e+38] (effectively unbounded) | N/A |

**1.2. Action Space**

In Cart Pole-v1, the action space is a Discrete space with a size of 2. This means that at each timestep, the agent has two distinct actions it can choose from.

These two discrete actions are:

* **Action 0**: This action corresponds to pushing the cart to the left with a fixed amount of force.
* **Action 1**: This action corresponds to pushing the cart to the right with the same fixed amount of force

**2. Interacting with the Cart Pole Environment**

Once an instance of the Cart Pole environment has been created, the agent can begin to interact with it. This interaction primarily involves two key functions: reset() and step().

**2.1. Resetting the Environment**

The env.reset() function serves to initialize the environment to a starting state and is typically called at the beginning of each new episode of interaction. An episode represents a complete run of the environment, from an initial state until a termination condition is met.

**2.2. Performing a Step in the Environment**

After the environment has been reset to its initial state, the agent can take actions to interact with it. This is done using the env.step(action) function, which advances the environment by one timestep based on the action chosen by the agent.

The five returned values from step() function have the following meanings:

* **obs**: This is the new observation of the environment after the chosen action has been executed.
* **reward**: This is the reward received by the agent for performing the action in the previous state. In the Cart Pole-v1 environment, the reward is typically 1.0 for each timestep that the pole remains balanced (i.e., within the termination angle limits).
* **terminated**: This is a boolean flag that becomes True when the episode ends due to a terminal state of the underlying Markov Decision Process (MDP) being reached. In the Cart Pole environment, this occurs when the pole falls beyond the angle limit (±12 degrees) or the cart moves out of the track bounds (±2.4).
* **truncated**: This is a boolean flag that becomes True when the episode ends due to a truncation condition that is outside the scope of the MDP. In Cart Pole-v1, the default truncation limit is 500 steps.
* **info**: This is a dictionary that can contain additional diagnostic information about the environment's state or the step taken.

**3. Running Multiple Episodes with a Random Policy**

An episode in the Cart Pole environment begins with a call to the reset() function, which provides the initial observation. The agent then enters a loop where, at each state, it chooses an action that is passed to the step() function. The step() function returns a new observation and the reward resulting from that action. This process of selecting an action and observing its consequences repeats until the episode terminates, as indicated by the terminated flag returned by the step() function. The sequence of actions taken by the agent during an episode is referred to as the policy, and the sum of all the rewards received during the episode is called the value of that policy.

To gain a basic understanding of the Cart Pole environment's behavior, it is useful to run multiple episodes where the agent selects actions randomly. This approach, known as a random policy, provides a baseline against which the performance of more sophisticated control strategies can be compared

This exploration with a random policy serves as a fundamental first step in understanding the environment's dynamics before attempting to implement more intelligent control strategies.

**4. Analyzing the Distribution of Observations from a Random Policy**

To gain a deeper understanding of the Cart Pole environment, it is beneficial to collect data on the observations generated while using a random policy over a significant number of episodes. By analyzing the distribution of these observations, we can infer typical ranges for the state variables and identify conditions that often lead to the termination of an episode.

The process involves running many episodes ( 100,000 episodes can provide a robust dataset) and recording the observation values at different points:

1. Initial observations after each reset,
2. All observations encountered during each episode
3. The observation at the moment the episode terminates.

By visualizing the distributions using histograms, we can gain the following insights into the behavior of the Cart Pole environment under a random policy:

* **Cart Position**: The distribution of all cart positions observed during random play is generally centered around 0, with most values falling within the interval of approximately [-0.5, 0.5].
* **Cart Velocity**: The distribution of cart velocities during random play tends to be wider, perhaps within the range of [-2.0, 2.0].
* **Pole Angle**: The distribution of pole angles for all observations is typically concentrated around 0, with most values falling within approximately [-0.28, 0.28] radians. In contrast, the distribution of pole angles at termination will likely show a strong concentration around the termination limits of [±0.2095] radians (±12 degrees). This clearly indicates that the pole falling beyond this critical angle is a primary reason for the episode to end.
* **Pole Angular Velocity**: The distribution of pole angular velocities during random play might be within the range of [-2.5, 2.5]. The terminal observations for angular velocity could show higher absolute values, implying that when the pole is falling, it often has a significant angular velocity.

The approximate ranges observed for each variable during random policy play can be summarized in the following table:

Approximate Observation Ranges from Random Policy

|  |  |
| --- | --- |
| Observation Description | Approximate Range |
| Cart Position | [-0.5, 0.5] |
| Cart Velocity | [-2.0, 2.0] |
| Pole Angle (radians) | [-0.28, 0.28] |
| Pole Angular Velocity | [-2.5, 2.5] |

It reveals that the system is inherently unstable and that the pole will quickly fall or the cart will move out of bounds if actions are chosen randomly. This underscores the need for a control policy that can react to the environment's state to maintain balance.

**5. Evaluating the Performance of Different Control Policies**

A control policy in reinforcement learning is a strategy that dictates the action an agent should take based on the current state of the environment.

**5.1. Random Action Policy**

A random action policy involves selecting actions (push left or push right) uniformly at random at each step. To evaluate the performance of such a policy, we can run a large number of episodes (e.g., 10,000) and record the length of each episode, measured by the number of steps taken before the episode terminates or is truncated. Since the agent receives a reward of +1 for every timestep the pole remains balanced, the episode length is equivalent to the total reward obtained in that episode.

Based on the give notebook, the typical performance of a random policy on Cart Pole-v1 is characterized by a minimum episode length of around 8 steps, a maximum length of approximately 114 steps, and an average length of about 22 steps. These metrics indicate that a random policy is generally ineffective at balancing the pole for a significant duration.

**5.2. Policy that Alternates Between Moving Left and Right**

Another simple control policy involves alternating between pushing the cart to the left (action 0) and pushing it to the right (action 1) at each step of the episode. To evaluate this policy, we can again run a large number of episodes and record the length of each one. According to the example in the given notebook, this alternating policy tends to perform better than the purely random policy. The typical performance metrics for the alternating policy are a minimum episode length of around 20 steps, a maximum length of approximately 163 steps, and an average length of about 37 steps.

Comparing these results to those of the random policy, we can see that the alternating policy achieves a higher average episode length (37 steps compared to 22). This improvement suggests that even a simple, deterministic strategy that introduces some structure to the actions taken can lead to better performance than completely random actions.

**6. Conclusion**

This report has provided a comprehensive introduction to getting started with the Cart Pole environment using the Gymnasium library. We have covered the essential steps, understanding its observation and action spaces and interacting with it using the reset() and step() functions. We have also explored the behavior of a random policy by analyzing the distribution of observations and the lengths of episodes. Furthermore, we have evaluated the performance of a simple alternating action policy and briefly touched upon the concept of more advanced, state-dependent policies.

#### References

1. Cart Pole - Gymnasium Documentation <https://gymnasium.farama.org/environments/classic_control/cart_pole/>
2. OpenAI Gym Example: CartPole - InterKI Machine Learning Library <https://cenos-interkiwwu.zivgitlabpages.uni-muenster.de/machine-learning-library/website/posts/cartpole_hillclimb/>
3. CartPole Balance using Python | OpenAI Gym | Reinforcement Learning Project Tutorial <https://www.hackersrealm.net/post/cartpole-balance-reinforcement-learning>