# Solution for Gymnasium Toy Games

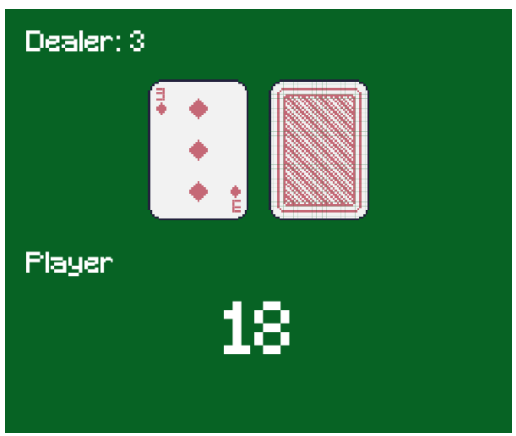
Cristian Curaba

September 2023

# Blackjack

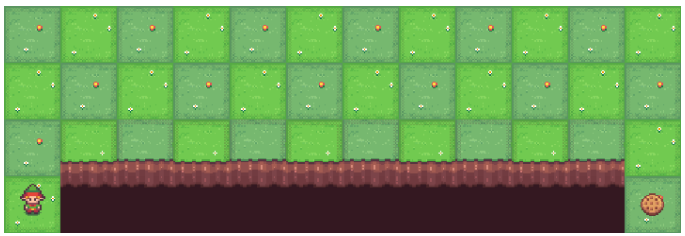
- Action space: 0 for stick, 1 for hit.
- Observation space: player current sum, dealer showing card value, usable ace
- Rewards: win +1, lose -1, draw 0, win with natural blackjack +1.5.
- Terminations: player hand exceeds 21, player sticks.

The dealer draws until it reaches 17 or more.



# Cliff Walking

- Action space: 0 move up, 1 move right, 2 move down, 3 move left.
- Observation space: $4 \times 12$ grid world, cliff at $[3, 1 \ldots 10]$.
- Rewards: -1 each step, -100 step into the cliff.
- Terminations: player enters $[3, 11]$.



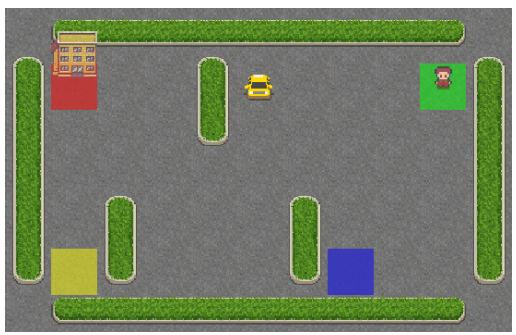
# Frozen Lake

- Action space: 0 move left, 1 move down, 2 move right, 3 move up. Slippery condition[1].
- Observation space: $4 \times 4$ grid world.
- Rewards: 1 reach goal, 0 reach hole, 0 reach frozen.
- Terminations: the player moves into a hole or the goal.

[1] 1/3 moves to the correct direction, 1/3 moves to an orthogonal direction

# Taxi

- Action space: 0 move down, 1 move up, 2 move right, 3 move left, 4 pickup passenger, 5 drop off passenger.
- Observation space: $5 \times 5$ grid world, 5 possible locations of the passenger, 4 destination locations.
- Rewards: -1 for each step, $+20$ delivering passenger, -10 executing "pickup" and "drop-off" illegally.
- Terminations: taxi drops off the passenger.
- Truncation: maximum length episode is 200



# General instruction

Implemented algorithms:

- on-policy methods: sarsa and expected sarsa;
- off-policy methods: q-learning and constant rate Monte Carlo.

Monte Carlo is the only one **not** bootstrapping.
For simplicity, I used a $\varepsilon$-greedy policy with a linear decaying function for the $\varepsilon$.
Each algorithm provides a Q-value function approximation.

# Q-learning

**Data:** Initialize Q(s, a) arbitrarily [all zeros];
**foreach** *episode* **do**
    Initialize the starting state, $s$;
    **repeat**
        Choose an action, $a$ [$\varepsilon$-greedy strategy]);
        Take action $a$, observe reward $r$ and next state $s'$;
        Update the Q-value for the current state-action pair;
        $Q(s,a) \leftarrow Q(s,a) + \lambda \cdot (r + \gamma \cdot \max_{a'}(Q(s',a')) - Q(s,a))$;
        Set the current state, $s$, to the next state, $s'$;
    **until** *episode ends*;
    Call $\varepsilon$ decay function [linear decay]

Hyperparameters tuned:

- Learning rate $\lambda$;
- Discount factor $\gamma$.

Training error: $r + \gamma \cdot \max_{a'}(Q(s',a')) - Q(s,a)$.

# SARSA

**Data:** Initialize Q(s, a) arbitrarily [all zeros];
**foreach** *episode* **do**
  Initialize the starting state, $s$;
  Choose an action, $a$ [$\epsilon$-greedy];
  **repeat**
    Take action $a$, observe reward $r$, and next state $s'$;
    Choose the next action, $a'$, using the same policy;
    Update the Q-value for the current state-action pair;
    $Q(s, a) \leftarrow Q(s, a) + \lambda \cdot (r + \gamma \cdot Q(s', a') - Q(s, a))$;
    Update action, $a$ and state, $s$ with $a', s'$;
  **until** *episode ends*;
  Call $\varepsilon$ decay function [linear decay]

Hyperparameters tuned:

- Learning rate $\lambda$;
- Discount factor $\gamma$.

Training error: $r + \gamma \cdot Q(s', a') - Q(s, a)$.

# Expected SARSA

---

**Data:** Initialize Q(s, a) arbitrarily;
**foreach** *episode* **do**
  Initialize the starting state, $s$;
  Choose an action, $a$ [$\epsilon$-greedy];
  **repeat**
    Take action $a$, observe reward $r$, and next state $s'$;
    Calculate the expected value of Q for the next state:
    expected $\leftarrow \sum_{a'} \pi(a'|s') \cdot Q(s', a')$;
    Update the Q-value for the current state-action pair:
    $Q(s, a) \leftarrow Q(s, a) + \lambda \cdot (r + \gamma \cdot \text{expected} - Q(s, a))$;
    Set the current state, $s$, to the next state, $s'$;
  **until** *episode ends*;
  Call $\varepsilon$ decay function [linear decay]

---

Hyperparameters tuned:

- Learning rate $\lambda$;
- Discount factor $\gamma$.

Training error: $r + \gamma \cdot \text{expected} - Q(s, a)$.

# Constant Rate Monte Carlo

**Data:** Initialize Q(s, a) arbitrarily [all zeros];
**foreach** *episode* **do**
    Initialize the starting state, $s$;
    Initialize $G = 0$;
    Initialize empty state, action and reward trajectories;
    **repeat**
        Take an action, $a$ [$\epsilon$-greedy];
        Update state, action and reward trajectories;
    **until** *episode ends*;
    Call $\varepsilon$ decay function [linear decay]
    **foreach** *s,a,r in reversed(trajectories)* **do**
        $G+ = r$;
        Update Q values once for state-action pair:
        $Q(s, a) \leftarrow Q(s, a) + \lambda \cdot (G - Q(s, a))$;

Hyperparameters tuned:

- Learning rate $\lambda$;

Training error: $\sum_{(s,a)\in \text{episode}} |G - Q(s, a)|$

# Analysis plan

The following hyper-parameters are common:

- number of episodes: from 20000 to 50000 (depending on the game);
- start epsilon: 1.0;
- final epsilon: 0.001;
- epsilon decay: $\frac{\text{start epsilon}}{\text{number of episodes}}$ (last $20,50$ episodes with fixed final epsilon);
- max episode length: 200.

# Analysis plan

The following hyper-parameters are common:

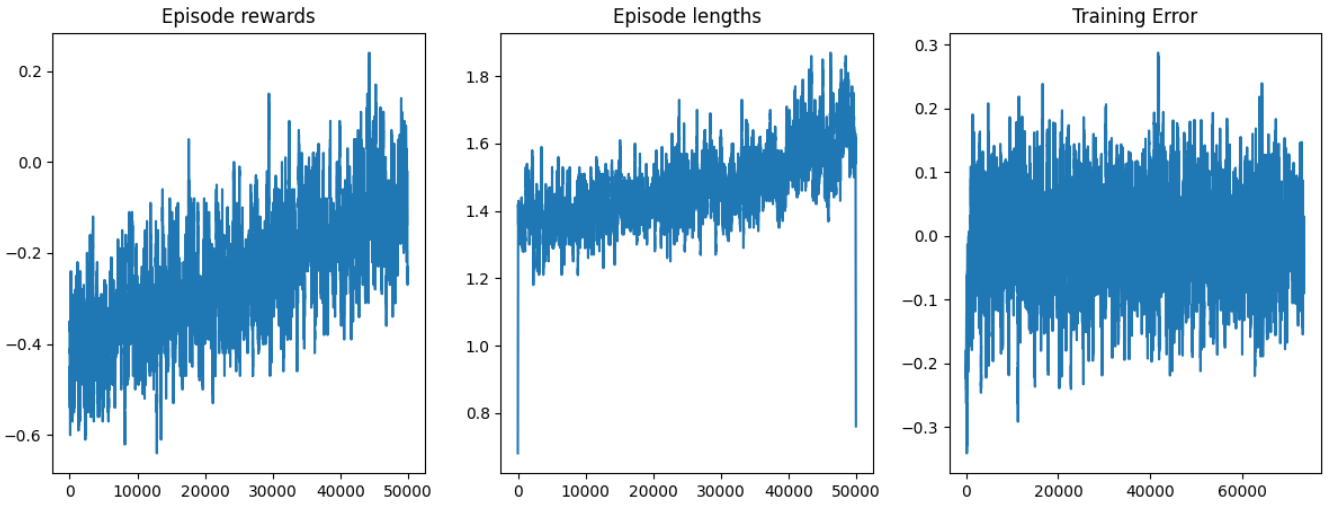
- number of episodes: from 20000 to 50000 (depending on the game);
- start epsilon: 1.0;
- final epsilon: 0.001;
- epsilon decay: $\frac{\text{start epsilon}}{\text{number of episodes}}$ (last $20, 50$ episodes with fixed final epsilon);
- max episode length: 200.

Grid tuning with `learning rate` in $\{0.5, 0.8, 0.9, 0.95, 0.99\}$ and `discount factor` (except for Monte Carlo) in $\{0.8, 0.95, 0.99\}$.

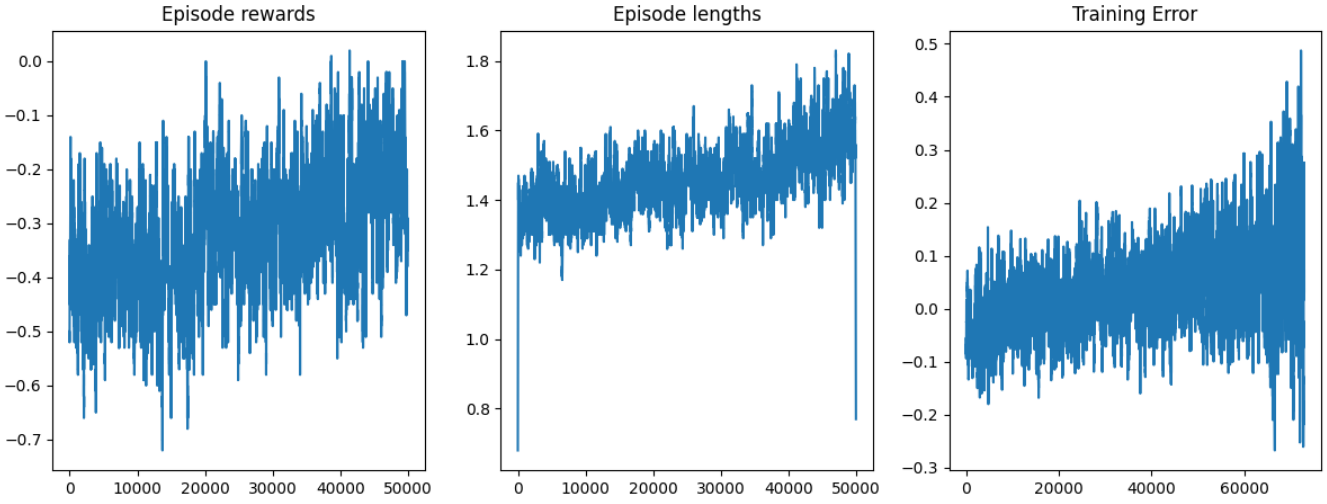
- Blackjack with 50000 episodes, convolution length is 100;
- Cliff Walking with 20000, convolution length is 1000;
- Frozen lake 50000, convolution length is 1000;
- Taxi 40000, convolution length is 1000.

# Blackjack

TD method: n_episodes=50000, learning_rate=0.5, discount_factor=0.8

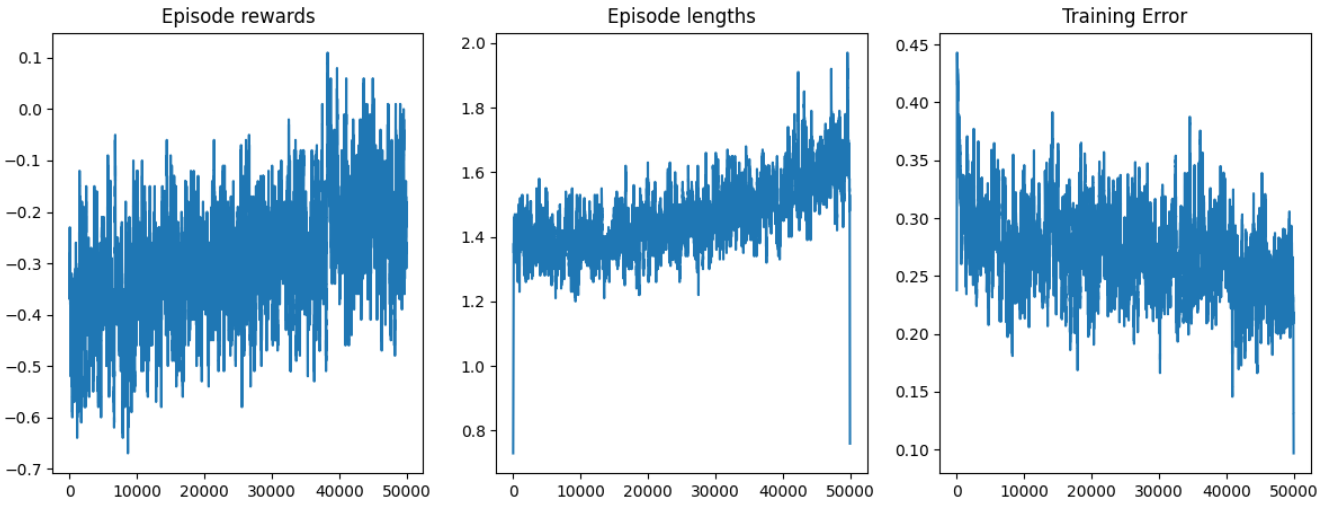



SARSA: learning_rate= 0.8, discount_factor= 0.99

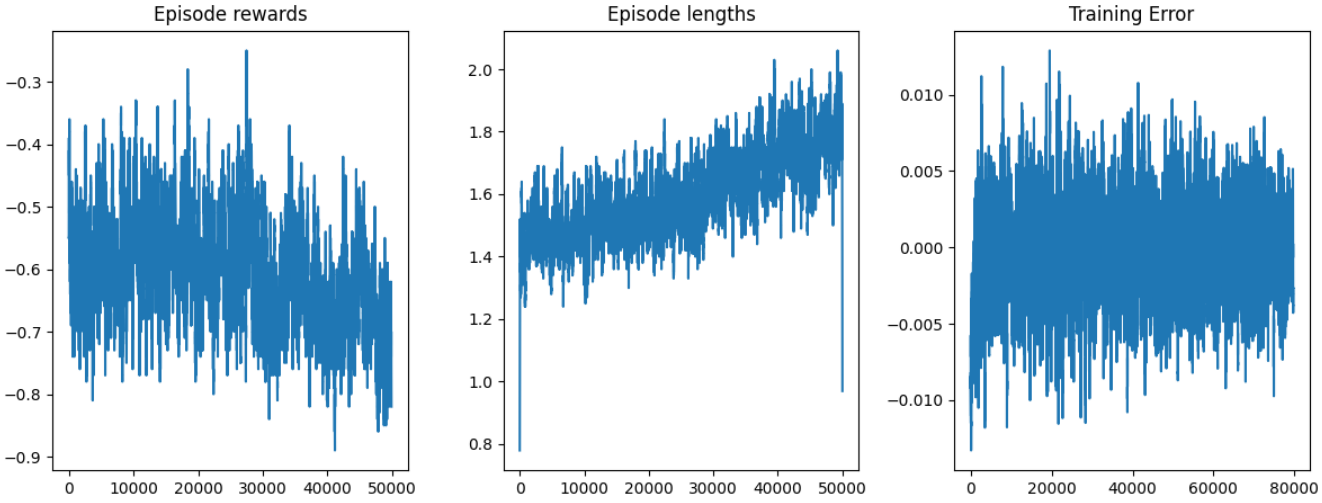


# Blackjack

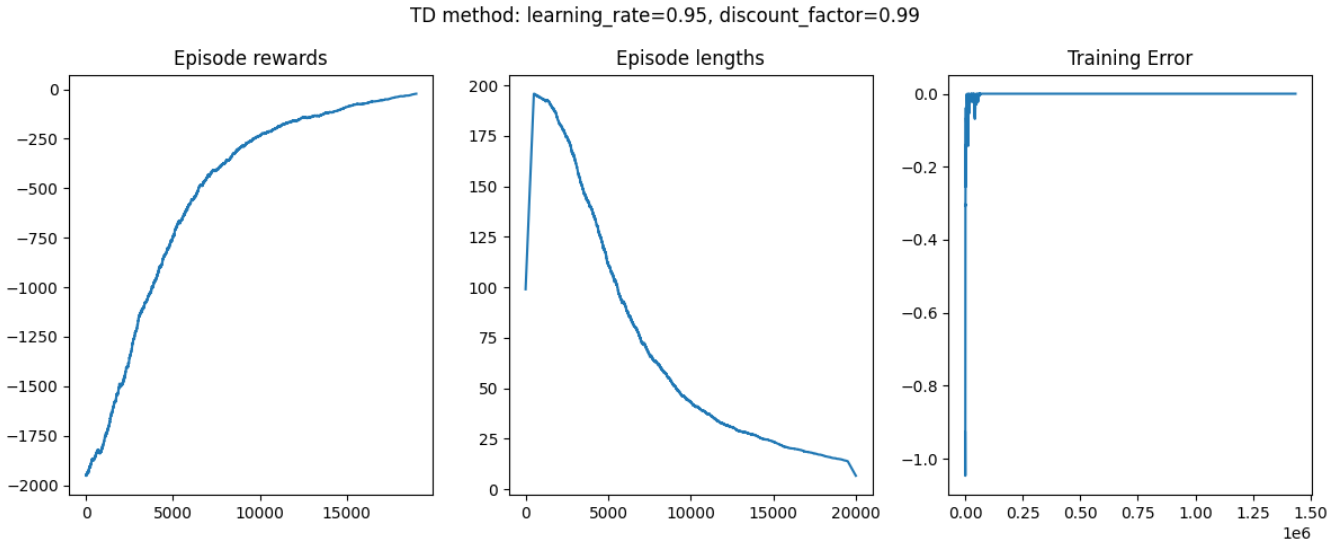
Constant Learning Rate Montecarlo: learning_rate= 0.5, discount_factor= 0.99

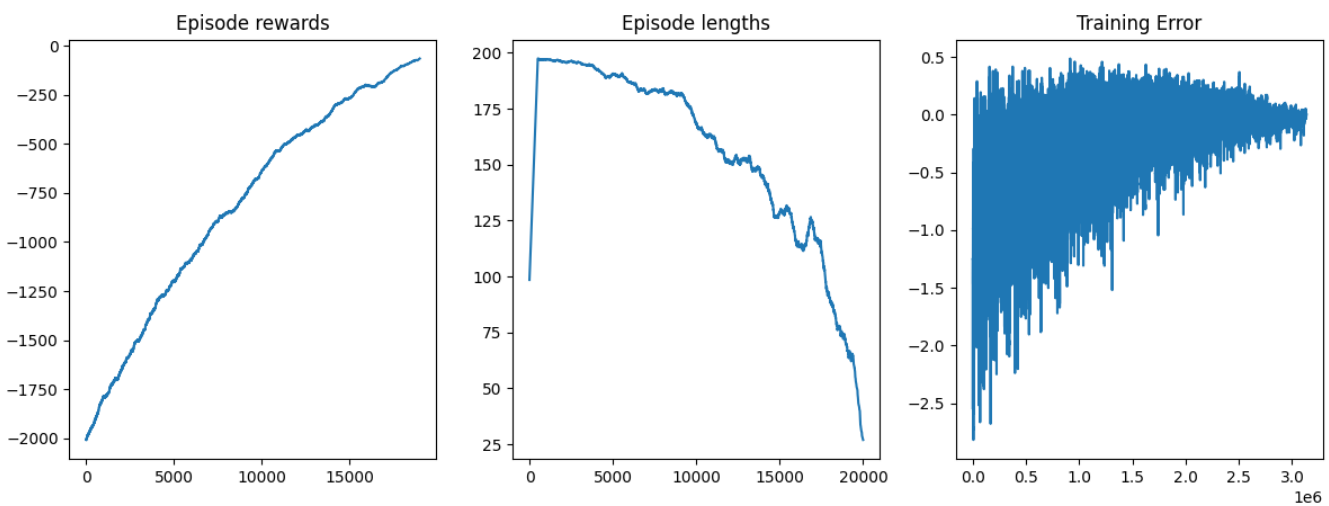



Expected SARSA: learning_rate= 0.95, discount_factor= 0.95

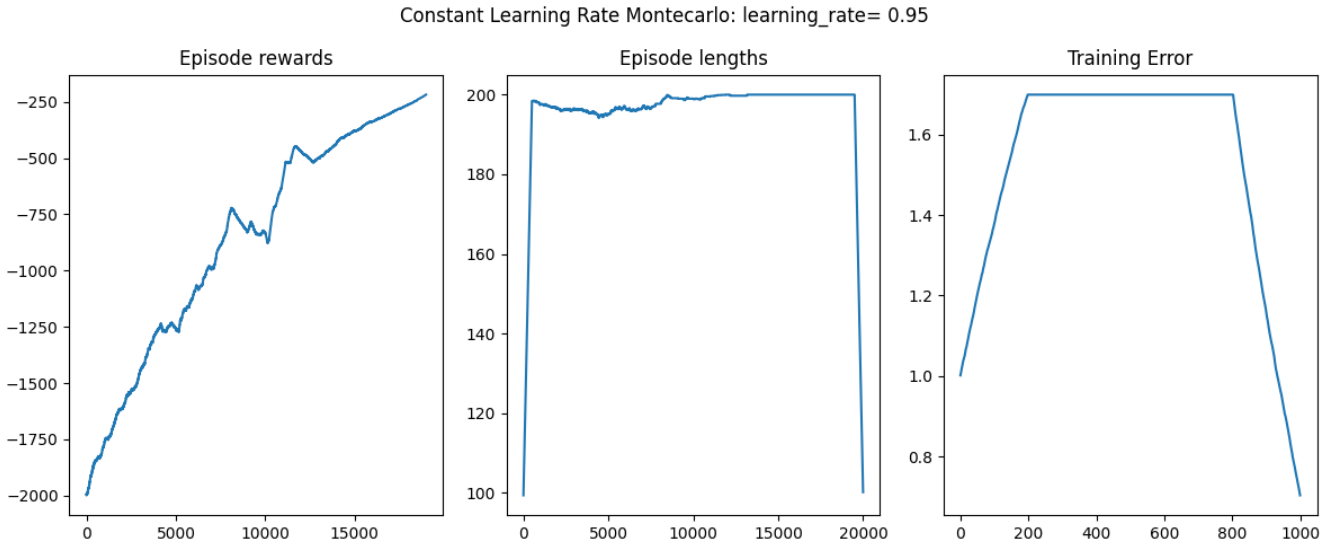


# Cliff Walking



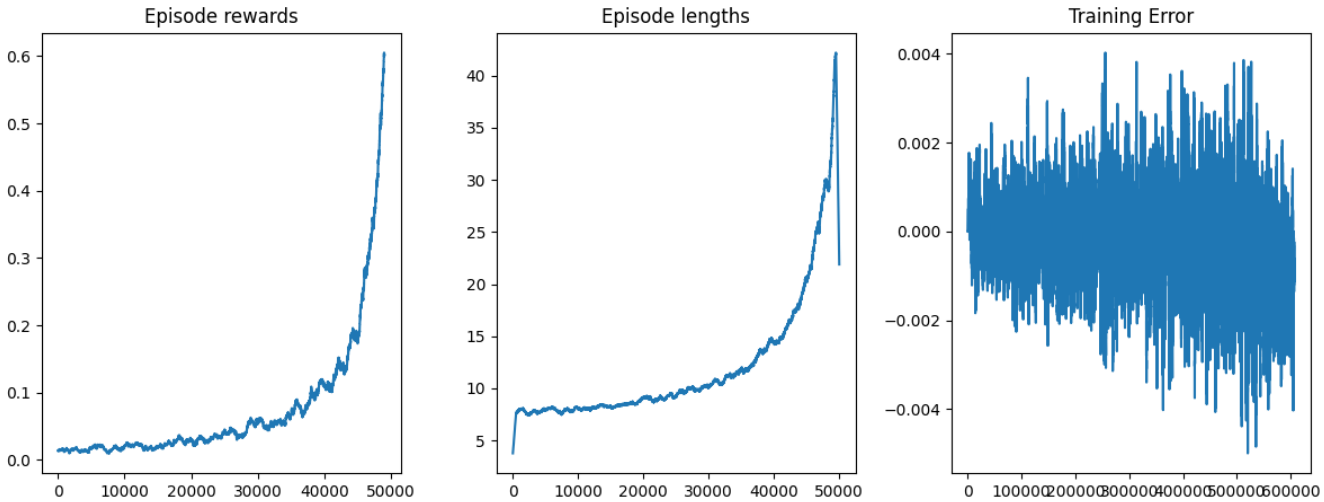



# Cliff Walking

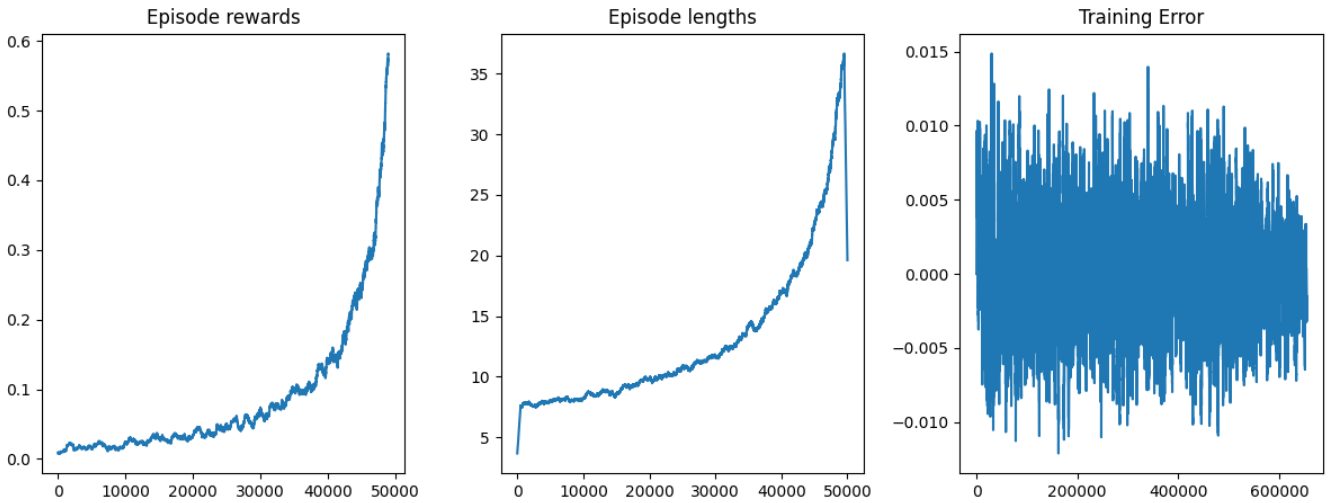



# Frozen Lake

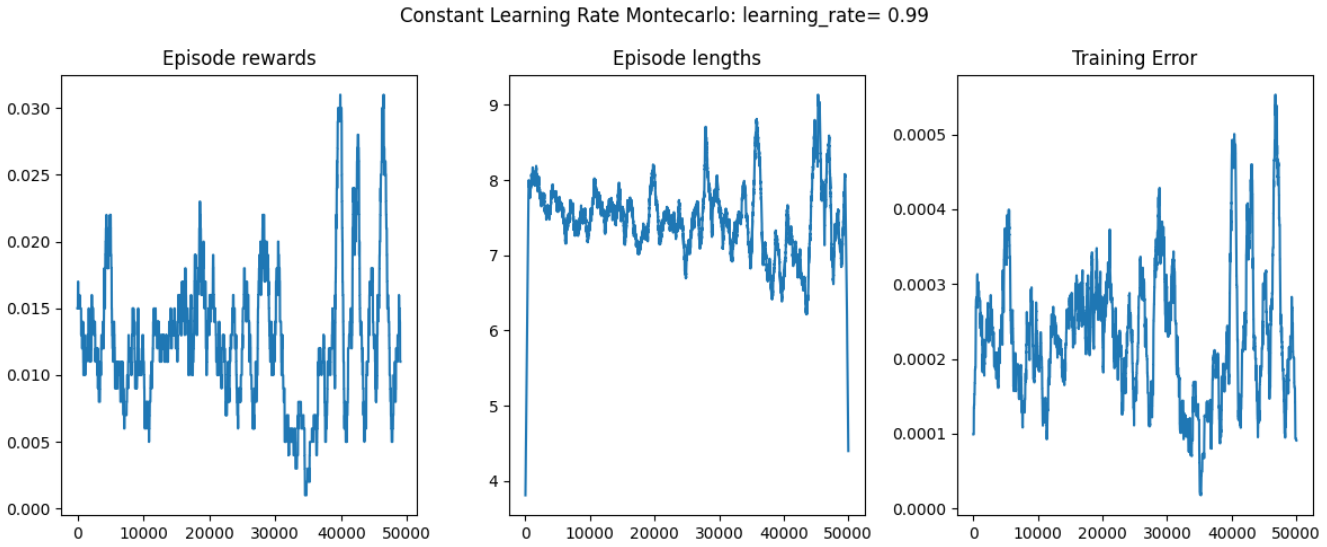
SARSA: learning_rate= 0.5, discount_factor= 0.99

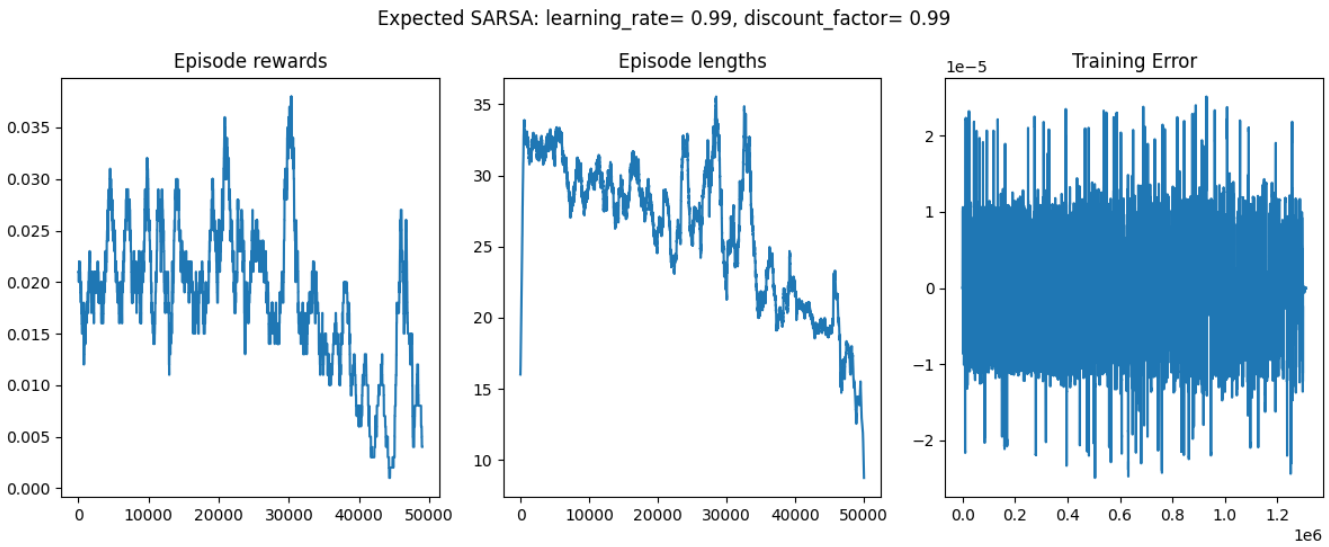



TD method: learning_rate=0.5, discount_factor=0.95



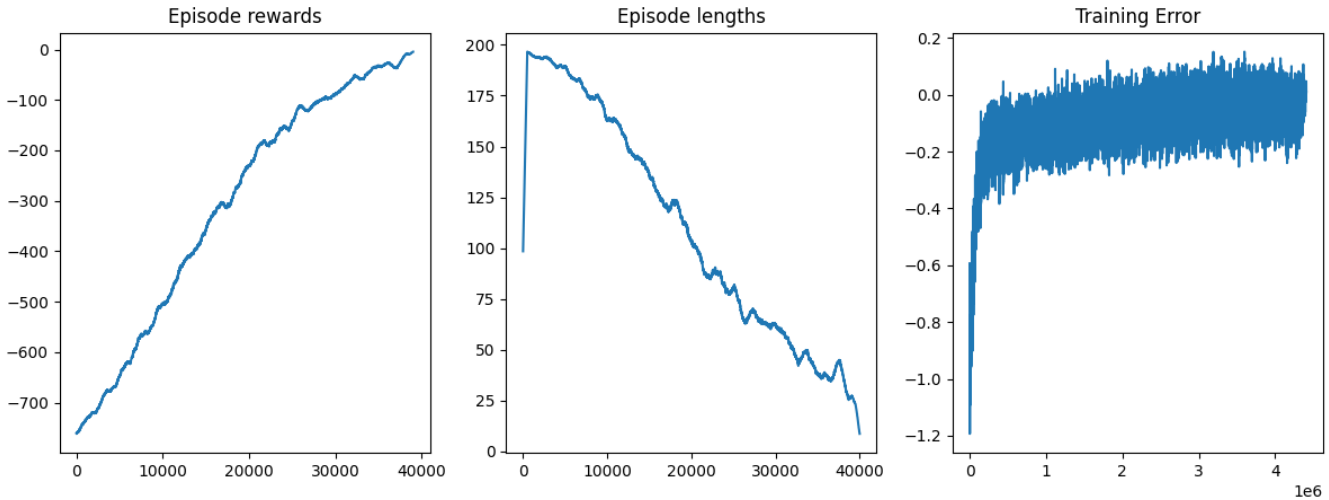
# Frozen Lake


Constant Learning Rate Montecarlo: learning_rate= 0.99
Episode rewards
Episode lengths
Training Error

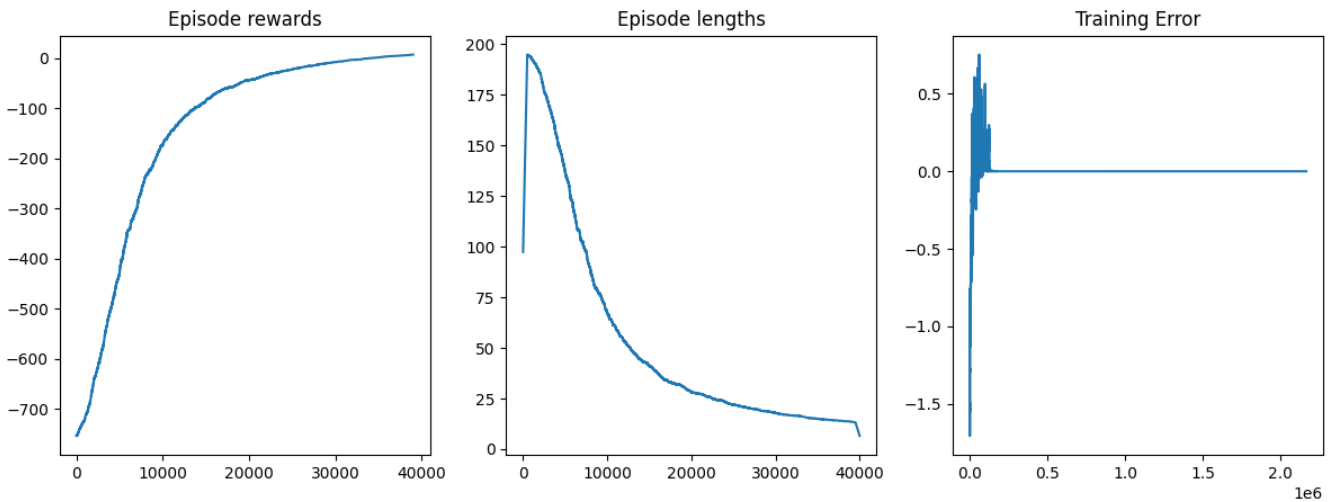

Expected SARSA: learning_rate= 0.99, discount_factor= 0.99
Episode rewards
Episode lengths
Training Error
1e−5
1e6

# Taxi

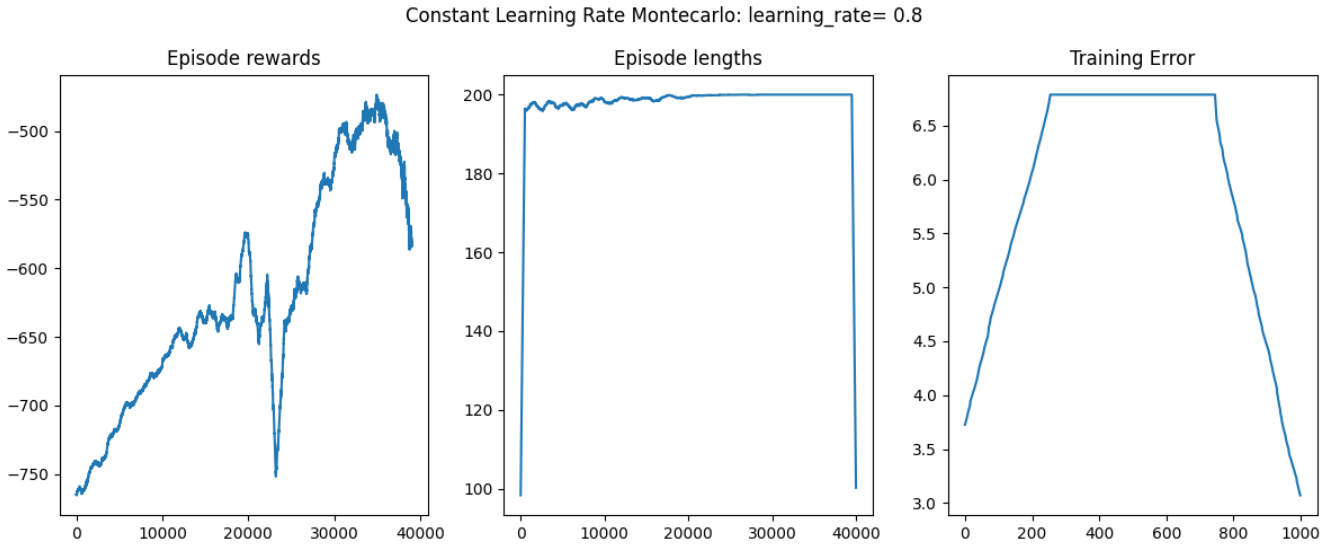
SARSA: learning_rate= 0.8, discount_factor= 0.95

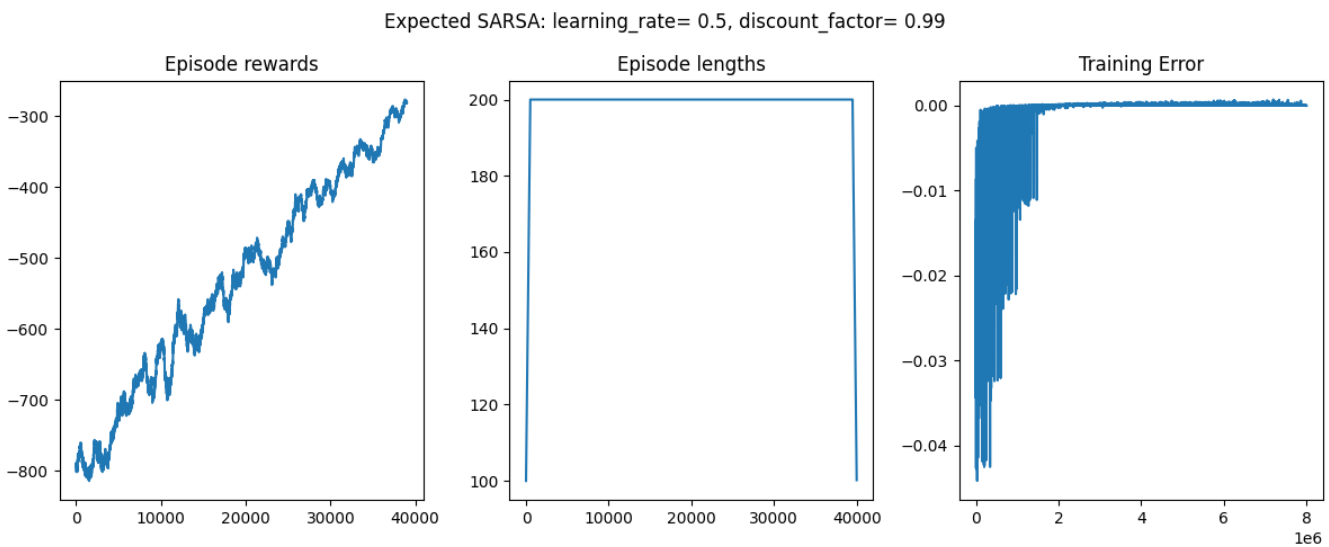



TD method: learning_rate=0.95, discount_factor=0.95

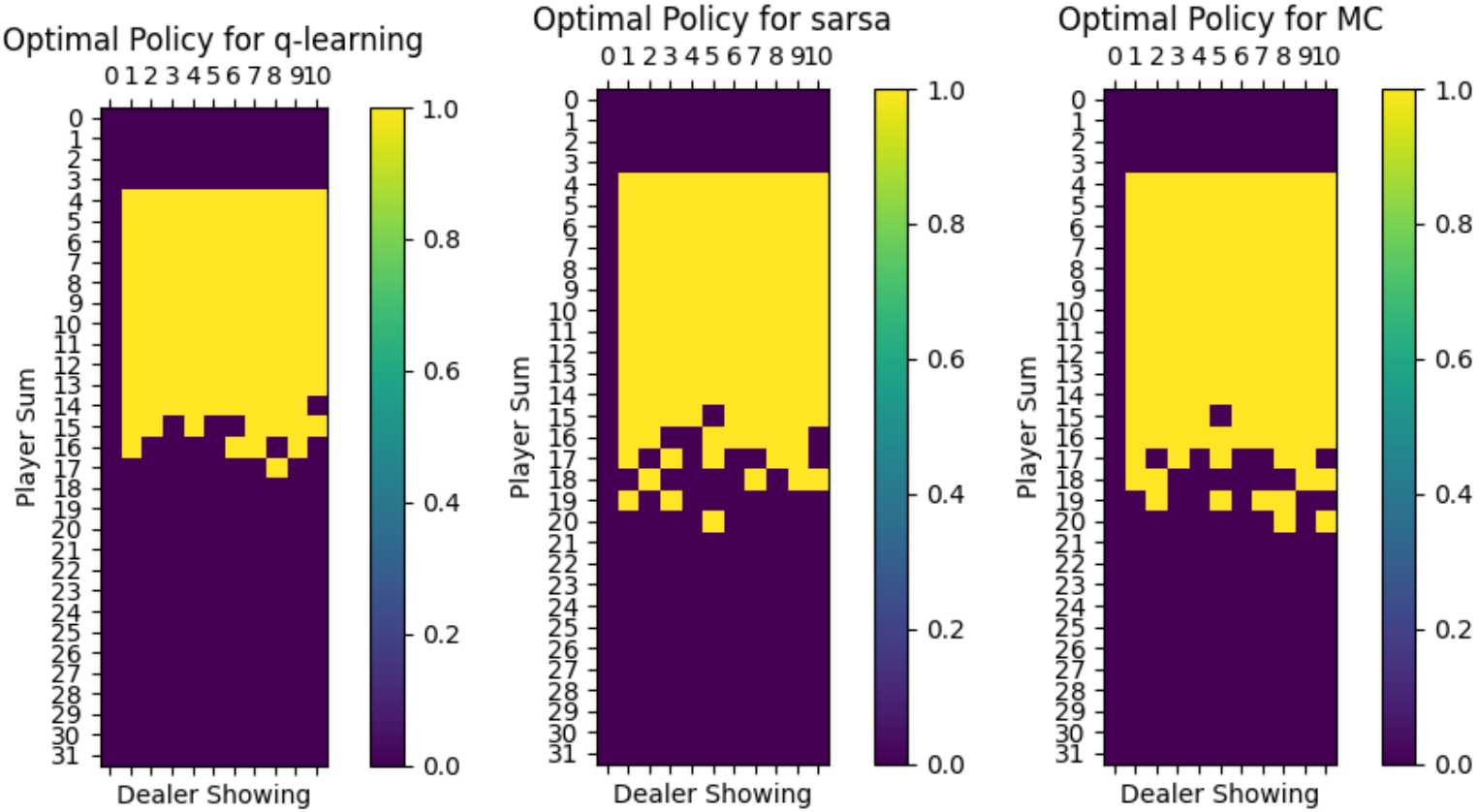


# Taxi

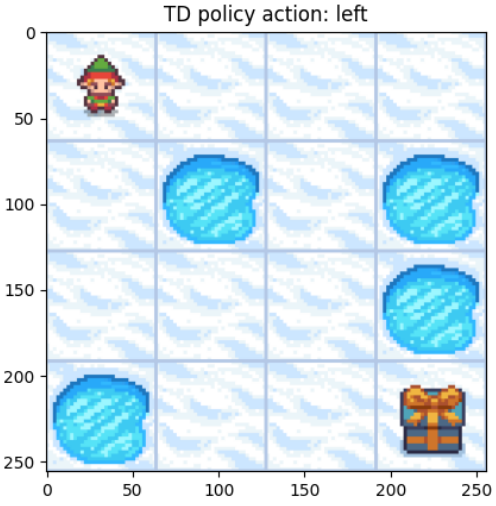

Constant Learning Rate Montecarlo: learning_rate= 0.8
Episode rewards
Episode lengths
Training Error

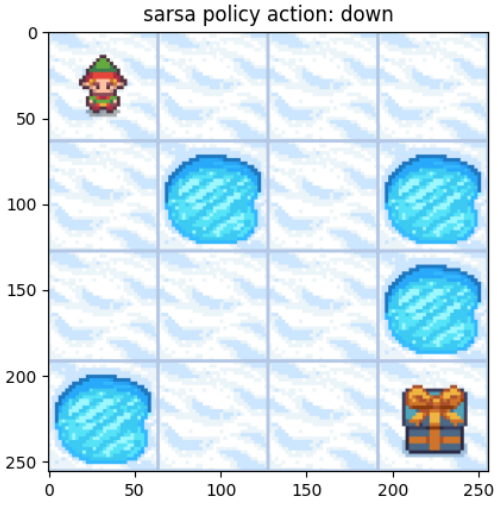

Expected SARSA: learning_rate= 0.5, discount_factor= 0.99
Episode rewards
Episode lengths
Training Error
1e6

# Blackjack policy

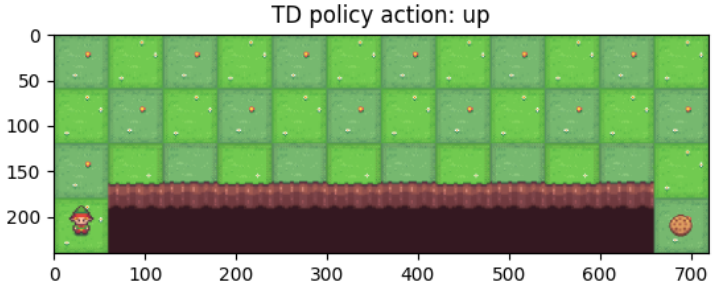


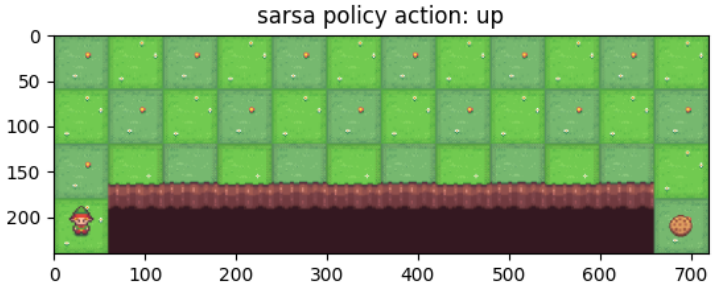
# Frozen Lake policy



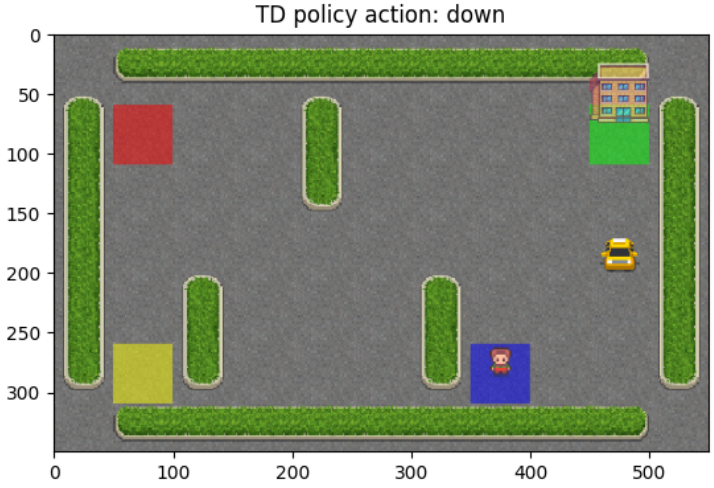



# Cliff policy



# Taxi policy


TD policy action: down
0
50
100
150
200
250
300
0
100
200
300
400
500


sarsa policy action: down
0
50
100
150
200
250
300
0
100
200
300
400
500