# Game Design

1. **Goals and Rules**

The primary objective of the game is for players getting higher points by moving up, down, left, and right to find treasures and reach the terminal. The score is influenced by the number of treasures collected, the step taken to complete the game, and penalties incurred from stepping on traps. The shorter completion time and the more treasures obtained, the higher score. Treasures disappeared once you have obtained them, but the traps are permanent.

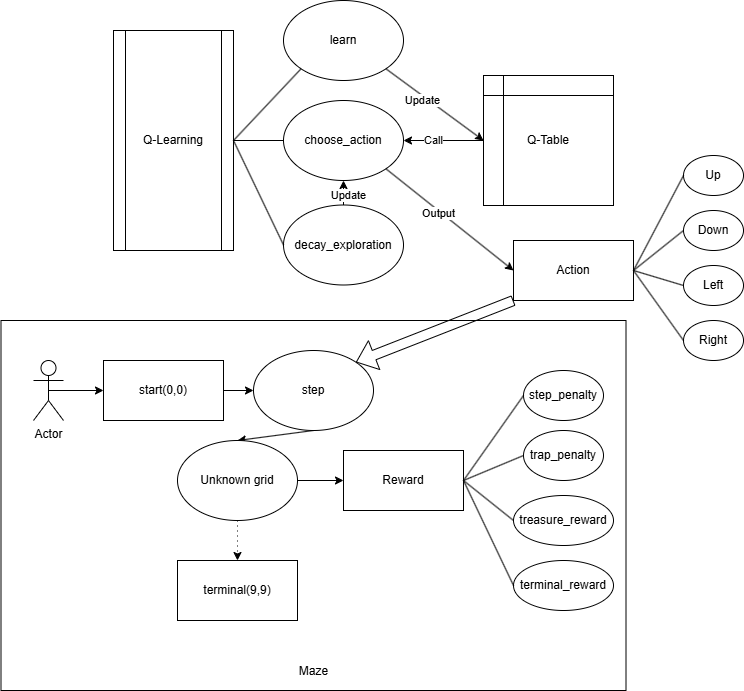
1. **Definition of State space and action space**

* **State space**: 100 states, represented as (x, y) tuples where x and y range from 0 to 9.
* **Action space**: 4 actions included up, down, left, right

1. **Reward Mechanism**

* **Trap penalty**: once you stepped on trap, you will lose 20 points.
* **Treasure reward**: once you stepped on treasure, you will obtain 20 points.
* **Step penalty**: moving from one state to another costs 1 point.

1. **Game Design**



The game employed the Q-learning by choosing the optimal solution for actions in the Q-table, updating the Q-table by learn Function, and gradually decrease the exploration rate in order to optimize the action selection by Q-table. After that, player choose the action, step to the next grid and get reward or penalty until he make it to the terminal.

# **Q-Learning Implementation**

def \_\_init\_\_(self, maze, learning\_rate=0.1, discount\_factor=0.9, exploration\_rate=1.0, exploration\_decay=0.99):

        self.maze = maze

        self.learning\_rate = learning\_rate

        self.discount\_factor = discount\_factor

        self.exploration\_rate = exploration\_rate

        self.exploration\_decay = exploration\_decay

1. **Parameters**

* **maze**: The maze environment, which includes the state space and Q-table;
* **learning\_rate**: Determines the step size for updating the Q-table (default: 0.1).
* **discount\_factor**: Discounts future rewards to emphasize immediate rewards (default: 0.9).
* **exploration\_rate**: Controls the balance between exploration (random actions) and exploitation (optimal actions) (default: 1.0).
* **exploration\_decay**: Gradually reduces the exploration rate to favor exploitation over time (default: 0.99).

def choose\_action(self, state):

        if random.uniform(0, 1) < self.exploration\_rate:

            return random.choice(actions)

        else:

            state\_index = self.maze.state\_to\_index(state)

            return actions[np.argmax(self.maze.q\_table[state\_index])]

1. **Choose\_action function**

If the random generated number is below the exploration\_rate, then randomly choose the action from up, down, left, right. Otherwise, choosing the action with highest value from q\_table by state\_index.

def learn(self, state, action, reward, next\_state):

        state\_index = self.maze.state\_to\_index(state)

        next\_state\_index = self.maze.state\_to\_index(next\_state)

        action\_index = actions.index(action)

        best\_next\_action = np.max(self.maze.q\_table[next\_state\_index])

        td\_target = reward + self.discount\_factor \* best\_next\_action

        td\_error = td\_target - self.maze.q\_table[state\_index][action\_index]

        self.maze.q\_table[state\_index][action\_index] += self.learning\_rate \* td\_error

1. **Learn function**

The function updates the Q-table based on the reward received and the estimated future reward.

* **best\_next\_action**: The action with the highest value in the Q table in the next state.
* **td\_target**: The target value for the Q-update.
* **td\_error**: The difference between the target value and the current Q-value.

The Q-value for the current state-action pair is updated using the learning rate.

def decay\_exploration(self):

        self.exploration\_rate \*= self.exploration\_decay

1. **Decay\_exploration function**

Each time it is called, the exploration\_rate(1.0) is multiplied by the exploration\_decay(0.99), so that the exploration rate gradually decreases.

This helps gradually reducing random exploration and increasing the exploitation of the best action during training.

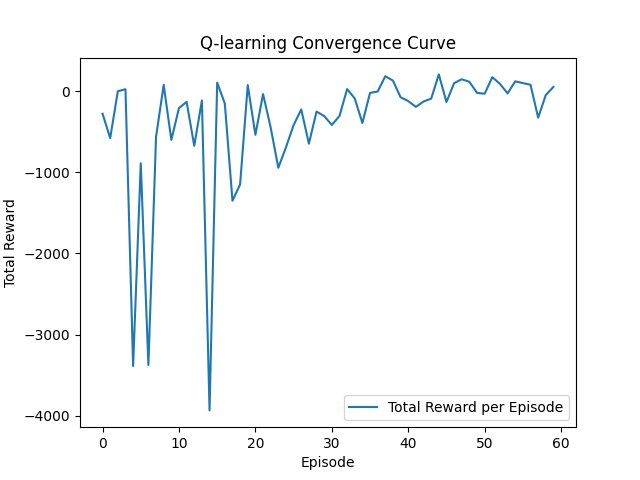
# **Evaluation**

The evaluation of the Q-learning based maze game is conducted through three approaches: performance metrics, parameter tuning, and environmental diversity tests.

1. **Performance metrics**

I think that score can be a vital performance metric.Since higher score means better Q-learning algorithm performance. By drawing the convergence curve can easily measure the performance of the Q-learning algorithm.

* **Training Setup**: I Set up a unified operating environment, same position of treasures and traps in training maze, same start state, end state and same size map. After that, I have trained the Q-learning algorithm 60 times and collected the score in each episode.



* **Results**: As you can see the performance of the Q-learning algorithm has fluctuated a lot at the begin, it’s getting more steady after 20 times training, then the total reward(score) gradually keeps steady almost at 0.

1. **Parameters tuning**

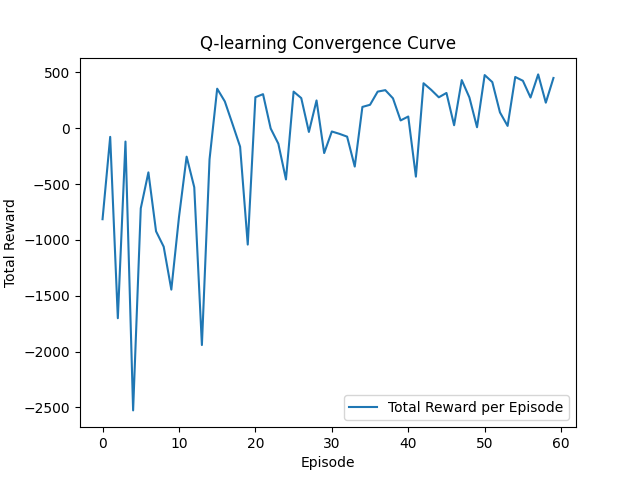
For parameters influencing the performance of the Q-

learning algorithm, I think that reward value and penalty value have influence on the performance, since they directly influence the score. Moreover, learning\_rate, discount\_factor, exploration\_rate, exploration\_decay also have influence on the performance, because they directly influence the Q-learning algorithm.

* **Reward and Penalty Adjustments:**

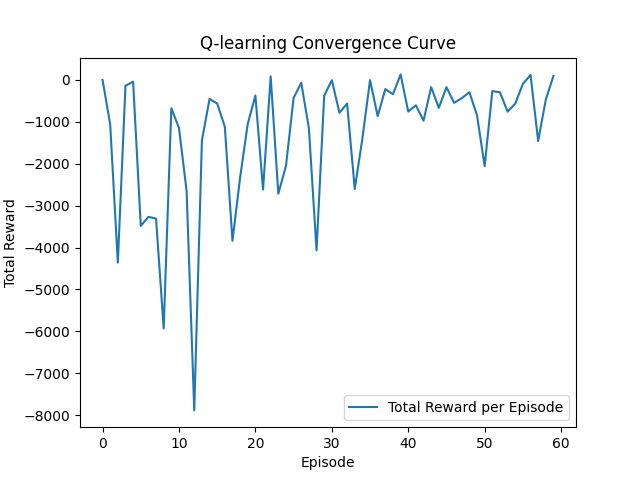
So I determine to alter these parameters, and see which one has a better convergence curve after 60 times training.

A.For reward value and penalty value, I increased the treasure\_reward from 20 to 80. the terminal\_reward and traps\_penalty are respectively 200 and -20 without altering.



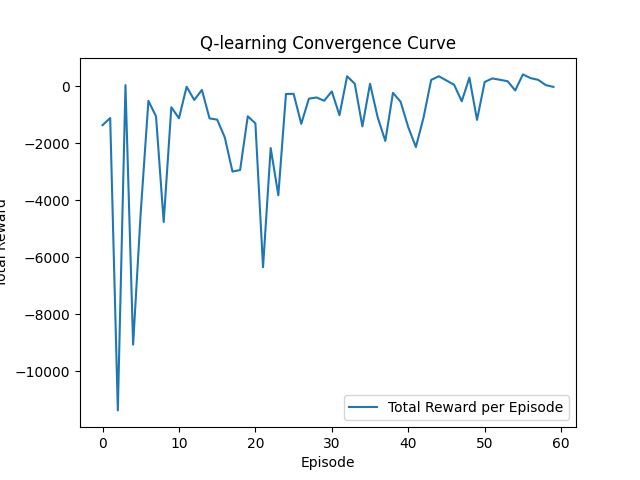
The convergence curve has slightly fluctuated at 300, despite the fluctuation at the begin, it shows continuously improving solutions made.

B.Then, I increase the traps\_penalty from -20 to -80. the terminal\_reward and treasure\_reward are respectively 200 and 20 without altering.



The convergence curve has fluctuated throughout the 60 times training even at the end. By increasing penalty value, it does not make ant improvement on Q-learning algorithm.

C.Moreover, I increase both of the traps\_penalty and treasure\_reward from -20,20 to -60,60.



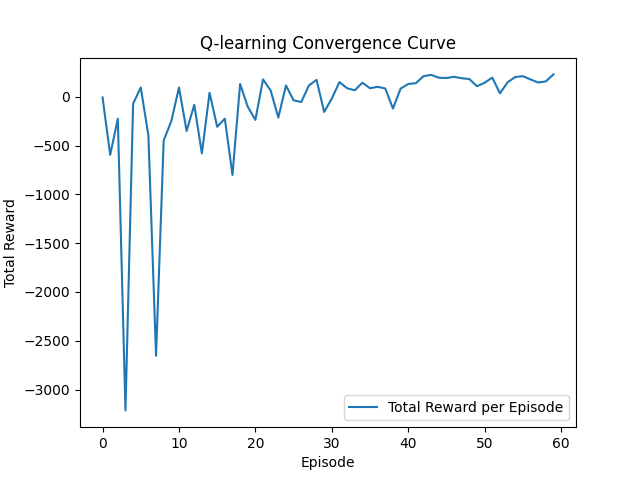
The result shows that Q-learning performed as similar as the reward value is 20 and penalty value is -20, the convergence curve has fluctuated during the first 20 times training and slightly fluctuates at -400.

In conclusion, by increasing the value of rewards can help the Q-learning algorithm find a more effective solution.

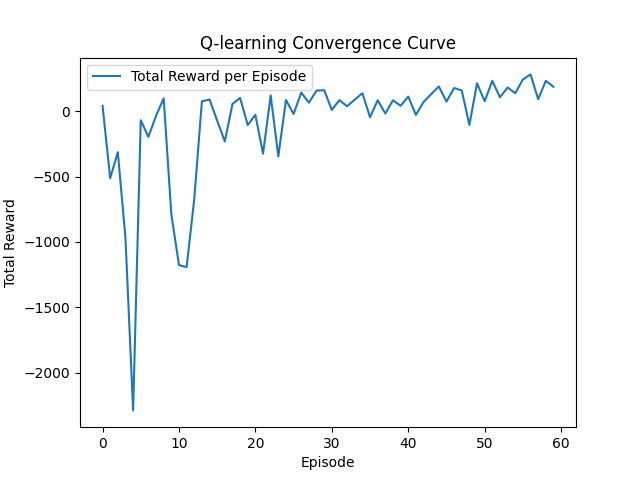
* **Learning Rate Adjustments:**

For Q-learning parameters, learning\_rate which determines the weight of new knowledge relative to old knowledge. As high learning\_rate means quicker learning, so I decide to increase the learning\_rate from 0.1(default) to 0.2 and 0.3, to see whether Q-learning convergence curve converges quicker and stable.

learning\_rate is 0.2



learning\_rate is 0.3

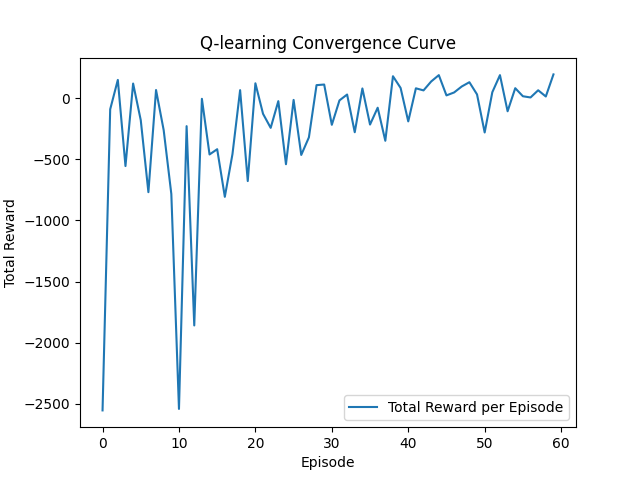


In conclusion, both plots show that as the number of training rounds increases, the total reward of the Q-learning algorithm gradually increases and tends to be stable. Compared with two plots, total reward increase with the increase of learning\_rate.

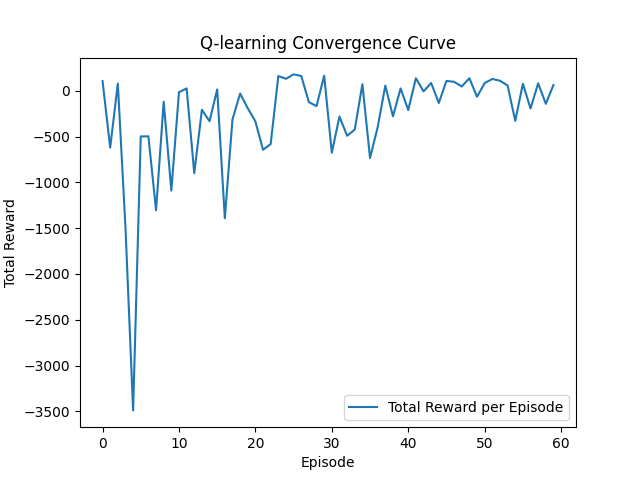
* **Discount Factor Adjustments:**

For discount\_factor, which determines the current value of future rewards. Since high discount\_factor makes the agent focus more on long-term rewards, while a low discount factor makes the agent focus more on immediate rewards. I decide to lower the discount\_factor value to 0.8 and 0.7 to find whether convergence curve will converge slowly or not.

discount\_factor value is 0.7



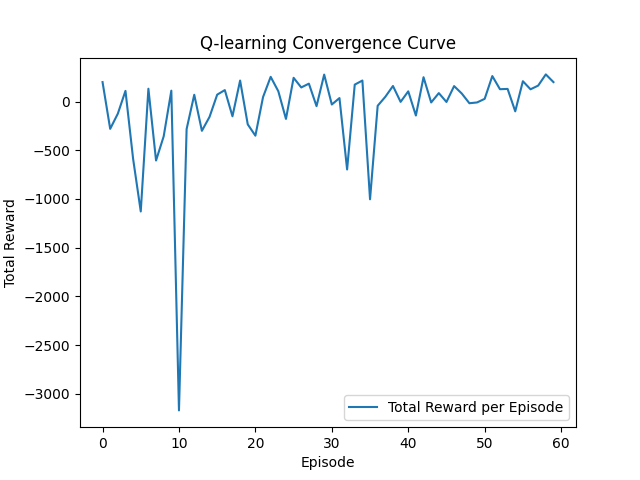
discount\_factor value is 0.8



Compared with two plots, they have almost same convergence rate and total reward. Decreasing the discount\_factor does not have obvious influence on the performance of Q-learning algorithm.

1. **Environmental diversity test**

Changing the static environment to dynamic environment, since we generate treasures and traps in fixed position, I decided to randomly generate them. By comparing the convergence curve, to see whether Q-learning algorithm perform better in the dynamic environment or not.



In the 60 times training, the total reward has fluctuated greatly in first 10 times training, and keeps steady at 0. This shows that the Q-learning is adapted to the dynamical environment of randomly generated treasures and traps, and the convergence curve has converged to a stable total reward value after the training.

# Summary

The game design leverages Q-learning to optimize the player's path through the maze. The Q-learning algorithm balances exploration and utilization, updates the Q-table based on rewards and penalties, and gradually reduces random exploration to enhance performance. This approach enables the player to efficiently navigate the maze, collect treasures, and avoid traps while minimizing steps token.