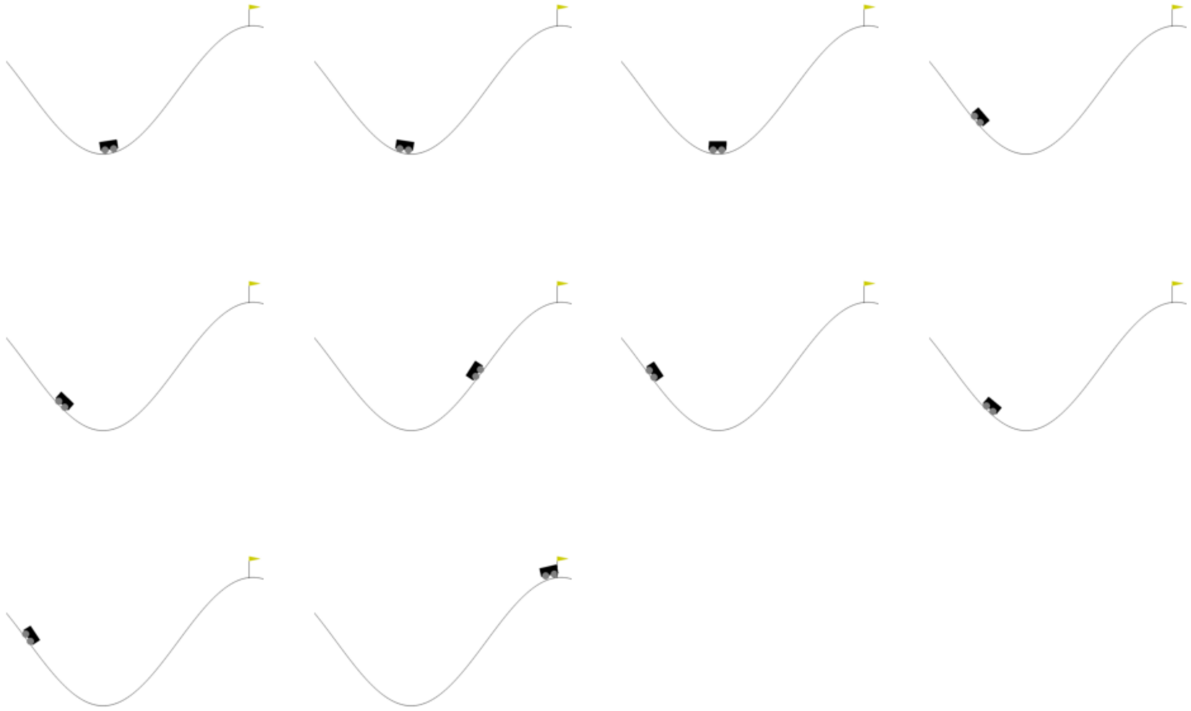
Assignment 5

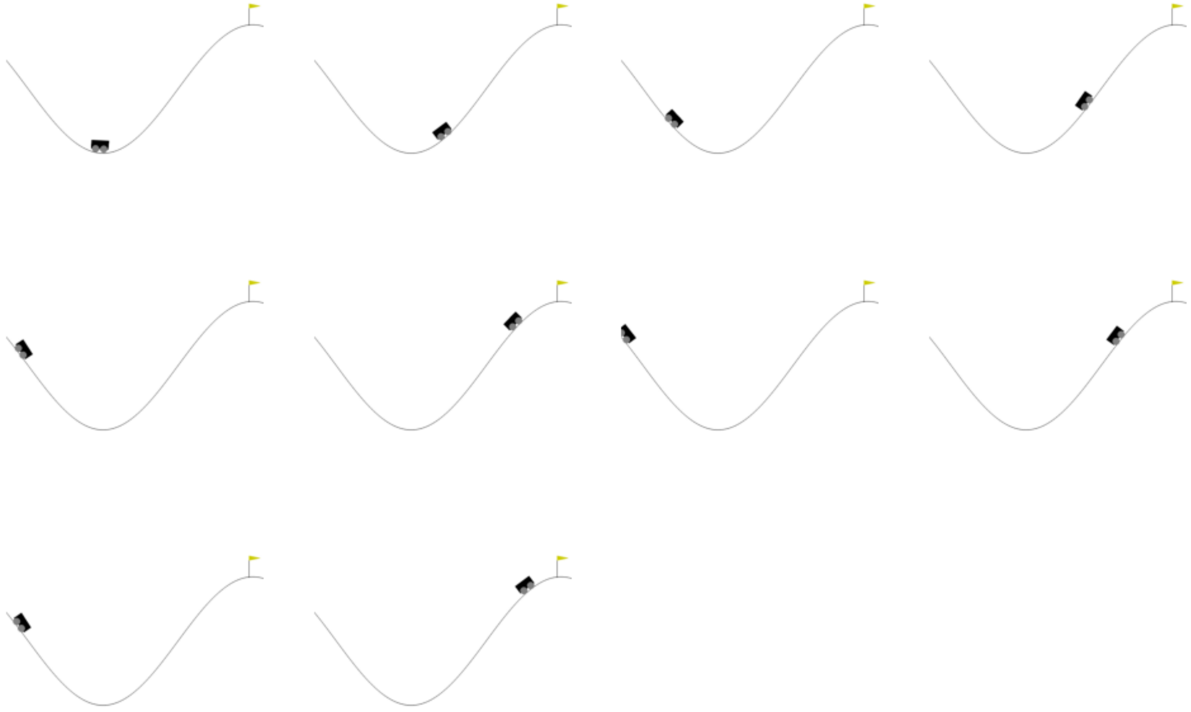
Mountain Car

```
def select_elements_at_equal_distances(arr, num_elements):
    if len(arr) <= num_elements:
        return arr  # Return the original array if it's smaller or equal to the number of elements you want to select.

    selected_elements = []
    interval = len(arr) // (num_elements - 1)  # Calculate the interval between selected elements.

    for i in range(0, len(arr), interval):
        selected_elements.append(arr[i])

    return selected_elements

def show_frames(image_array):
  # Create a figure and specify the number of columns for subplots
  image_array = select_elements_at_equal_distances(image_array, 10)
  num_images = len(image_array)
  num_columns = 4  # You can adjust the number of columns as needed
  num_rows = (num_images + num_columns - 1) // num_columns  # Calculate the number of rows

  # Create subplots
  fig, axes = plt.subplots(num_rows, num_columns, figsize=(12, 8))

  for i in range(num_images):
      # Determine the subplot position
      row = i // num_columns
      col = i % num_columns

      # Display the image in the corresponding subplot
      axes[row, col].imshow(image_array[i], cmap='gray')  # You can specify a colormap (e.g., 'gray' for grayscale)

      # Remove axis labels and ticks
      axes[row, col].axis('off')

  # Ensure that unused subplots are hidden
  for i in range(num_images, num_rows * num_columns):
      row = i // num_columns
      col = i % num_columns
      fig.delaxes(axes[row, col])

  # Show the plot
  plt.show()

import gym
import numpy as np
import matplotlib.pyplot as plt
env = gym.make("MountainCar-v0", render_mode='rgb_array')

#Environment values
print(env.observation_space.high) #[0.6 0.07]
print(env.observation_space.low) #[-1.2 -0.07]
print(env.action_space.n) #3

DISCRETE_BUCKETS = 20
EPISODES = 1001
DISCOUNT = 0.95
EPISODE_DISPLAY = 100
LEARNING_RATE = 0.1
EPSILON = 0.5
EPSILON_DECREMENTER = EPSILON/(EPISODES//4)

#Q-Table of size DISCRETE_BUCKETS*DISCRETE_BUCKETS*env.action_space.n
Q_TABLE =
np.random.randn(DISCRETE_BUCKETS,DISCRETE_BUCKETS,env.action_space.n)

# For stats
ep_rewards = []
ep_rewards_table = {'ep': [], 'avg': [], 'min': [], 'max': []}

def discretised_state(state):
    DISCRETE_WIN_SIZE = (env.observation_space.high-
env.observation_space.low)/[DISCRETE_BUCKETS]*len(env.observation_space
.high)
    discrete_state = (state-
env.observation_space.low)//DISCRETE_WIN_SIZE
    return tuple(discrete_state.astype(int))


#integer tuple as we need to use it later on to extract Q table values
for episode in range(EPISODES):
    episode_reward = 0
    done = False
    curr_discrete_state = discretised_state(env.reset()[0])
    image_array = list()
    if episode % EPISODE_DISPLAY == 0:
        render_state = True
    else:
        render_state = False
    while not done:
        if np.random.random() > EPSILON:
            action = np.argmax(Q_TABLE[curr_discrete_state])
        else:
            action = np.random.randint(0, env.action_space.n)

        [new_state, reward, done, _, _] = env.step(action)
        new_discrete_state = discretised_state(new_state)

        if render_state:
            image_array.append(env.render())

        if not done:
            max_future_q = np.max(Q_TABLE[new_discrete_state])
            current_q = Q_TABLE[curr_discrete_state+(action,)]
            new_q = current_q + LEARNING_RATE*(reward
+DISCOUNT*max_future_q- current_q)
            Q_TABLE[curr_discrete_state+(action,)]=new_q
        elif new_state[0] >= env.goal_position:
            Q_TABLE[curr_discrete_state + (action,)] = 0

        curr_discrete_state = new_discrete_state
        episode_reward += reward
    EPSILON = EPSILON - EPSILON_DECREMENTER
    ep_rewards.append(episode_reward)

    if episode % EPISODE_DISPLAY == 0:
        avg_reward = sum(ep_rewards[-
EPISODE_DISPLAY:])/len(ep_rewards[-EPISODE_DISPLAY:])
        ep_rewards_table['ep'].append(episode)
        ep_rewards_table['avg'].append(avg_reward)
        ep_rewards_table['min'].append(min(ep_rewards[-
EPISODE_DISPLAY:]))
        ep_rewards_table['max'].append(max(ep_rewards[-
EPISODE_DISPLAY:]))
        print(f"Episode:{episode} avg:{avg_reward}
min:{min(ep_rewards[-EPISODE_DISPLAY:])} max:{max(ep_rewards[-
EPISODE_DISPLAY:])}")
        print(len(image_array))
        show_frames(image_array)
env.close()

plt.plot(ep_rewards_table['ep'], ep_rewards_table['avg'], label="avg")
plt.plot(ep_rewards_table['ep'], ep_rewards_table['min'], label="min")
plt.plot(ep_rewards_table['ep'], ep_rewards_table['max'], label="max")
plt.legend(loc=4) #bottom right
plt.title('Mountain Car Q-Learning')
plt.ylabel('Average reward/Episode')
plt.xlabel('Episodes')
plt.show()
```

```
[0.6  0.07]
[-1.2  -0.07]
3
Episode:0 avg:-2958.0 min:-2958.0 max:-2958.0
2958
```

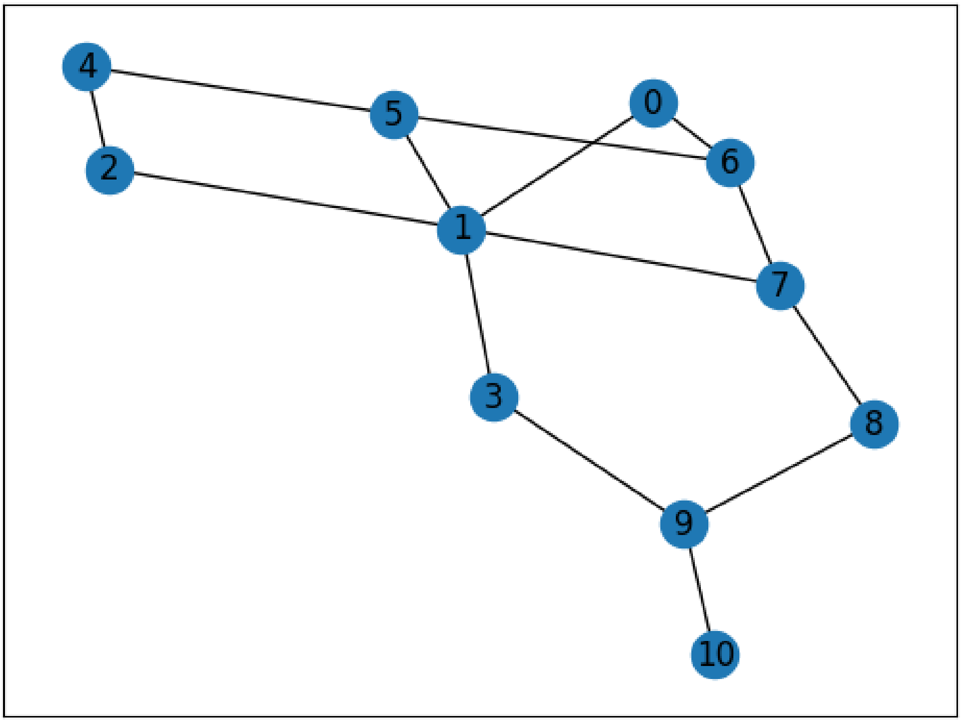


```
Episode:100 avg:-1073.83 min:-7674.0 max:-201.0
386
```



Shortest Path

```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import random
import pandas as pd
edges = [(0, 1), (1, 5), (5, 6), (5, 4), (1, 2), (1, 3), (9, 10), (2, 4), (0, 6), (6, 7),(8, 9), (7, 8), (1, 7), (3, 9)]
G=nx.Graph()
G.add_edges_from(edges)
pos=nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)
plt.show()
```



```
R=np.matrix(np.zeros(shape=(11, 11)))
for x in G[10]:
  R[x,10]=100
Q=np.matrix(np.zeros(shape=(11, 11)))
Q-=100
for node in G.nodes:
  for x in G[node]:
    Q[node,x]=0
    Q[x,node]=0
pd.DataFrame(R)
```

	0	1	2	3	4	5	6	7	8	9	10
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
pd.DataFrame(Q)
```

	0	1	2	3	4	5	6	7	8	9	10
0	-100.0	0.0	-100.0	-100.0	-100.0	-100.0	0.0	-100.0	-100.0	-100.0	-100.0
1	0.0	-100.0	0.0	0.0	-100.0	0.0	-100.0	0.0	-100.0	-100.0	-100.0
2	-100.0	0.0	-100.0	-100.0	0.0	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0
3	-100.0	0.0	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0	0.0	-100.0
4	-100.0	-100.0	0.0	-100.0	-100.0	0.0	-100.0	-100.0	-100.0	-100.0	-100.0
5	-100.0	0.0	-100.0	-100.0	0.0	-100.0	0.0	-100.0	-100.0	-100.0	-100.0
6	0.0	-100.0	-100.0	-100.0	-100.0	0.0	-100.0	0.0	-100.0	-100.0	-100.0
7	-100.0	0.0	-100.0	-100.0	-100.0	-100.0	0.0	-100.0	0.0	-100.0	-100.0
8	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0	0.0	-100.0	0.0	-100.0
9	-100.0	-100.0	-100.0	0.0	-100.0	-100.0	-100.0	-100.0	0.0	-100.0	0.0
10	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0	-100.0	0.0	-100.0

```
def next_number(start,er):
  random_value=random.uniform(0, 1)
  if random_value<er:
    sample=G[start]
  else:
    sample=np.where(Q[start,]== np.max(Q[start,]))[1]
  next_node=int(np.random.choice(sample,1))
  return next_node
def updateQ(node1,node2,lr,discount):
  max_index=np.where(Q[node2,]==np.max(Q[node2,]))[1]
  if max_index.shape[0]>1:
    max_index=int(np.random.choice(max_index,size=1))
  else:
    max_index=int(max_index)
  max_value=Q[node2,max_index]
  Q[node1,node2]=int(1-
lr)*Q[node1,node2]+lr*(R[node1,node2]+discount*max_value)
def learn(er,lr,discount):
  for i in range(50000):
    start=np.random.randint(0,11)
    next_node=next_number(start,er)
    updateQ(start,next_node,lr,discount)
learn(0.5,0.8,0.8)
pd.DataFrame(Q)
```

	0	1	2	3	4	5	6	7	8	9	10
0	-100.000000	35.520867	-100.000000	-100.000000	-100.000000	-100.000000	22.733355	-100.000000	-100.000000	-100.000000	-100.000000
1	22.733355	-100.000000	22.733355	55.501355	-100.000000	22.733355	-100.000000	35.520867	-100.000000	-100.000000	-100.000000
2	-100.000000	35.520867	-100.000000	-100.000000	14.549347	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000
3	-100.000000	35.520867	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	86.720867	-100.000000
4	-100.000000	-100.000000	22.733355	-100.000000	-100.000000	22.733355	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000
5	-100.000000	35.520867	-100.000000	-100.000000	14.549347	-100.000000	22.733355	-100.000000	-100.000000	-100.000000	-100.000000
6	22.733355	-100.000000	-100.000000	-100.000000	-100.000000	22.733355	-100.000000	35.520867	-100.000000	-100.000000	-100.000000
7	-100.000000	35.520867	-100.000000	-100.000000	-100.000000	-100.000000	22.733355	-100.000000	55.501355	-100.000000	-100.000000
8	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	35.520867	-100.000000	86.720867	-100.000000
9	-100.000000	-100.000000	-100.000000	55.501355	-100.000000	-100.000000	-100.000000	-100.000000	55.501355	-100.000000	135.501355

	0	1	2	3	4	5	6	7	8	9	10
10	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	-100.000000	86.720867	-100.000000

```
def shortest_path(begin,end):
  path=[begin]
  next_node=np.argmax(Q[begin,])
  path.append(next_node)
  while next_node!=end:
    next_node=np.argmax(Q[next_node,])
    path.append(next_node)
  return path
```

```
shortest_path(0,10)
```

```
[0, 1, 3, 9, 10]
```