Name :	**Dipangshu Roy**
Roll No :	**001811001014**
Department :	**Information Technology**
Class :	**4th Year 1st Sem**
Subject :	**Machine Learning Lab**

Assignment 5

1. Using Reinforcement Learning (RL), implement the following examples:

   a. Mountain Car trying to go top a hill

   b. Car Racing

   c. Roulette

> **Github Link of the Assignments :** https://github.com/Droyder7/ML-Lab-Assignments

A. Mountain Car trying to go top a hill

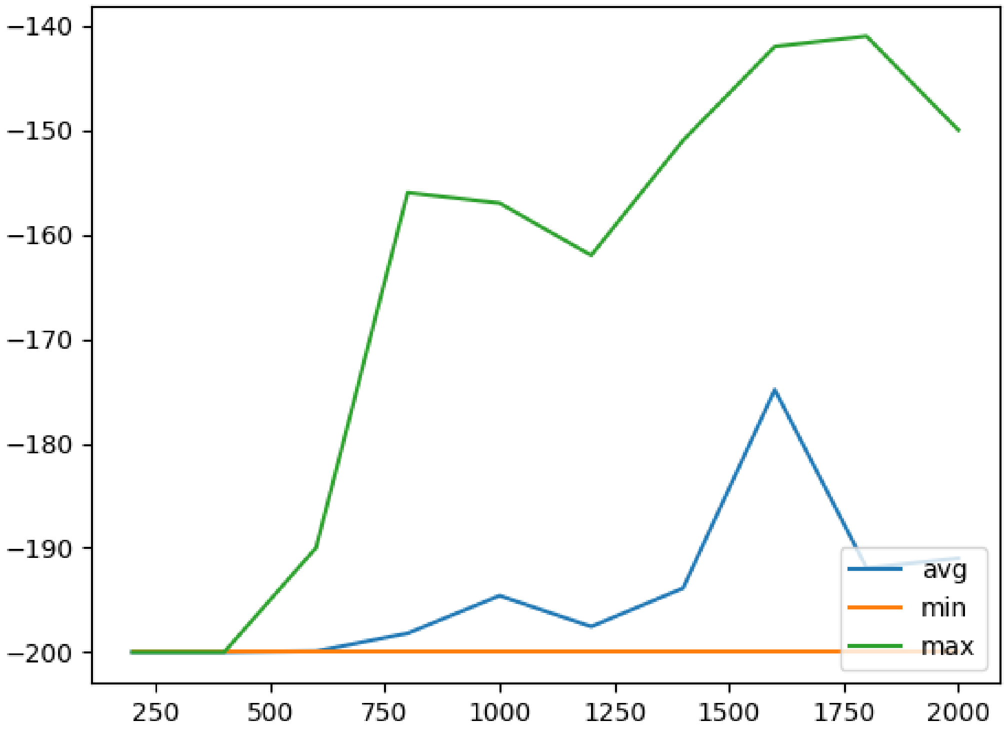
> **Code :**

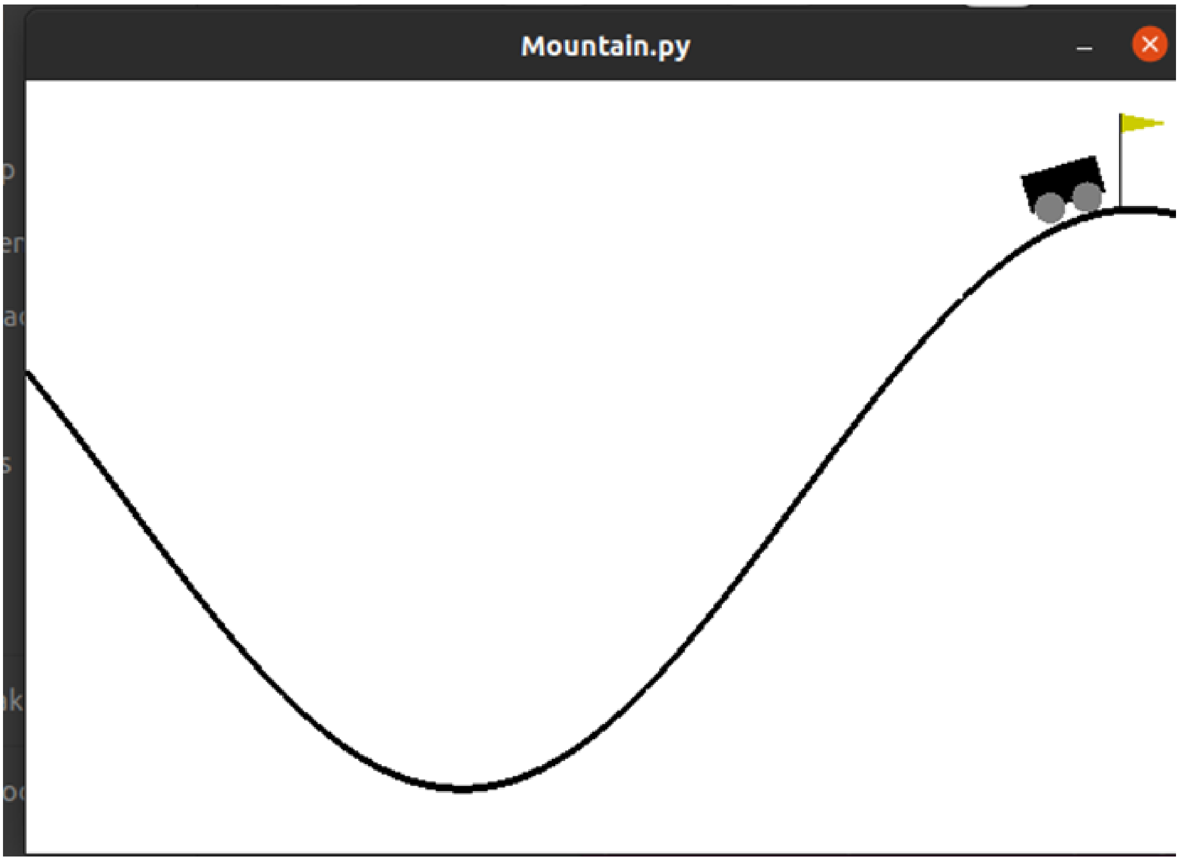
```
import gym
import matplotlib.pyplot as plt
import numpy as np

env = gym.make("MountainCar-v0")
env.reset()

LEARNING_RATE = 0.1
DISCOUNT = 0.95
EPISODES = 2000

SHOW_EVERY = 200
RENDER_EVERY = 100

DISCRETE_OS_SIZE = [20] * len(env.observation_space.high)
discrete_os_win_size = (env.observation_space.high - env.observation_space.low) /
DISCRETE_OS_SIZE

epsilon = 0.2
START_EPSILON_DECAYING = 1
END_EPSILON_DECAYING = EPISODES // 2

epsilon_decay_value = epsilon/(END_EPSILON_DECAYING - START_EPSILON_DECAYING)

q_table = np.random.uniform(low=-2, high=0, size=(DISCRETE_OS_SIZE +
[env.action_space.n]))
```

```
ep_rewards = []
aggr_ep_rewards = {'ep': [], 'avg': [], 'min': [], 'max': []}


def get_discrete_state(state):
    discrete_state = (state - env.observation_space.low) / discrete_os_win_size
    return tuple(discrete_state.astype(np.int))


best_episode_reward = -200
best_episode = -1
best_q_table = q_table
for episode in range(1, EPISODES + 1):
    episode_success = False
    episode_reward = 0
    if not episode % SHOW_EVERY:
        try:
            average_reward = sum(ep_rewards[-SHOW_EVERY:])/len(ep_rewards[-
SHOW_EVERY:])
            aggr_ep_rewards['ep'].append(episode)
            aggr_ep_rewards['avg'].append(average_reward)
            aggr_ep_rewards['min'].append(min(ep_rewards[-SHOW_EVERY:]))
            aggr_ep_rewards['max'].append(max(ep_rewards[-SHOW_EVERY:]))
            print(f"episode: {aggr_ep_rewards['ep'][-1]}, avg: {aggr_ep_rewards['avg']
[-1]}, "
                    + f"min: {aggr_ep_rewards['min'][-1]}, max:
{aggr_ep_rewards['max'][-1]}")
        except Exception as e:
            print(e)

    discrete_state = get_discrete_state(env.reset())
    done = False
    while not done:
        if np.random.random() > epsilon:
            action = np.argmax(q_table[discrete_state])
        else:
            action = np.random.randint(0, env.action_space.n)

        new_state, reward, done, _ = env.step(action)
        episode_reward += reward
        new_discrete_state = get_discrete_state(new_state)
        if RENDER_EVERY > 1:
            if not episode % RENDER_EVERY:
                env.render()
        if not done:
            max_future_q = np.max(q_table[new_discrete_state])
            current_q = q_table[discrete_state + (action, )]
            new_q = (1 - LEARNING_RATE) * current_q + LEARNING_RATE * (reward +
DISCOUNT * max_future_q)
            q_table[discrete_state + (action, )] = new_q
        elif new_state[0] >= env.goal_position:
            episode_success = True
            q_table[discrete_state + (action, )] = 0

        discrete_state = new_discrete_state

    if END_EPSILON_DECAYING >= episode >= START_EPSILON_DECAYING:
        epsilon -= epsilon_decay_value

    if episode_reward >= best_episode_reward:
        best_episode_reward = episode_reward
        best_episode = episode
        best_q_table = q_table
    ep_rewards.append(episode_reward)
    env.close()

print(f"best reward: {best_episode_reward}")
print(f"best episode: {best_episode}")

plt.plot(aggr_ep_rewards['ep'], aggr_ep_rewards['avg'], label="avg")
plt.plot(aggr_ep_rewards['ep'], aggr_ep_rewards['min'], label="min")
plt.plot(aggr_ep_rewards['ep'], aggr_ep_rewards['max'], label="max")
plt.legend(loc=4)
plt.show()
```

> **Output :**

```
$ python3 Mountain.py
Mountain.py:32: DeprecationWarning: `np.int` is a deprecated alias for the builtin
`int`. To silence this warning, use `int` by itself. Doing this will not modify any
behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or
`np.int32` to specify the precision. If you wish to review your current use, check the
release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  return tuple(discrete_state.astype(np.int))
episode: 200, avg: -200.0, min: -200.0, max: -200.0
episode: 400, avg: -200.0, min: -200.0, max: -200.0
episode: 600, avg: -199.905, min: -200.0, max: -190.0
episode: 800, avg: -198.19, min: -200.0, max: -156.0
episode: 1000, avg: -194.595, min: -200.0, max: -157.0
episode: 1200, avg: -197.55, min: -200.0, max: -162.0
episode: 1400, avg: -193.87, min: -200.0, max: -151.0
episode: 1600, avg: -174.86, min: -200.0, max: -142.0
episode: 1800, avg: -191.94, min: -200.0, max: -141.0
episode: 2000, avg: -191.0, min: -200.0, max: -150.0
best reward: -141.0
best episode: 1756
```





B. Car Racing

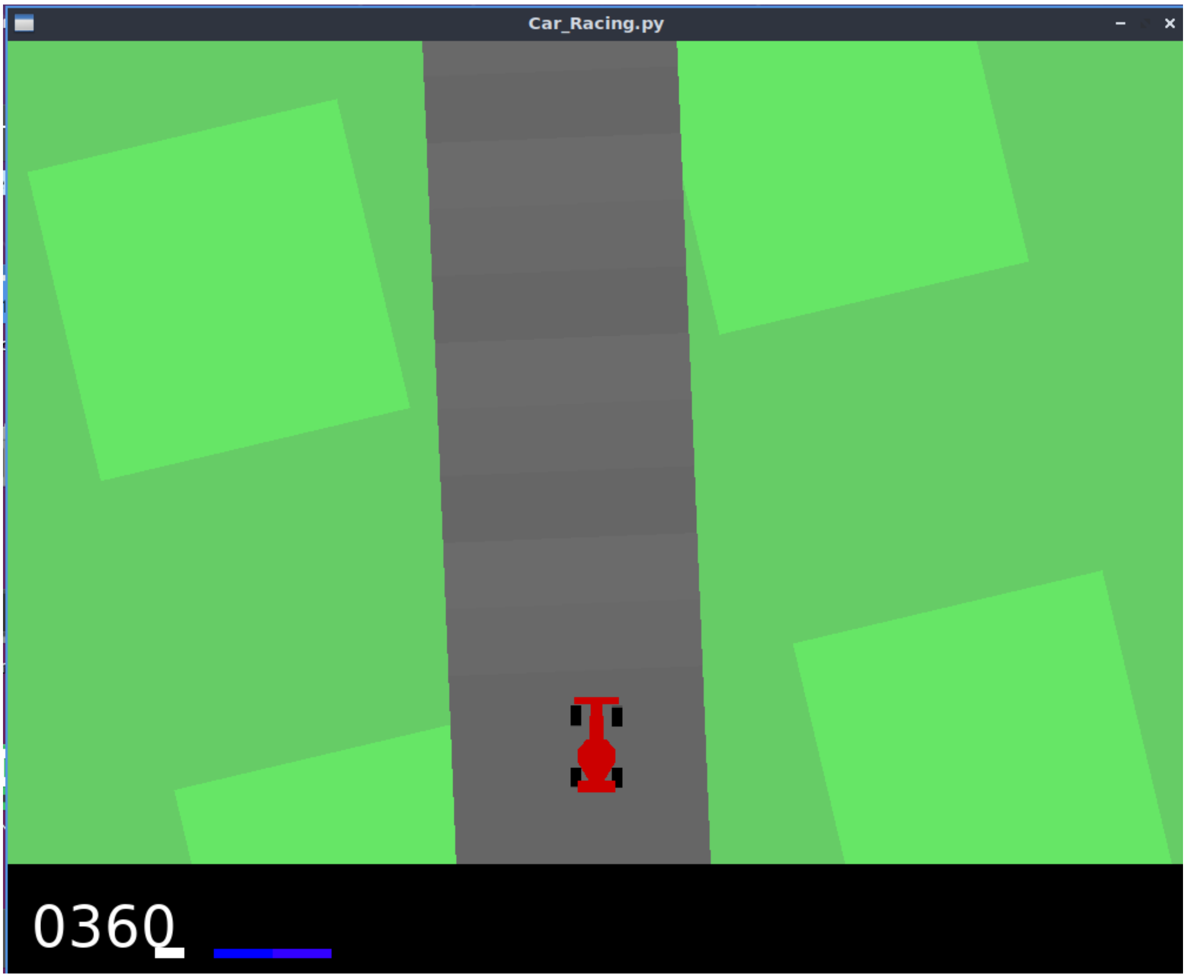
Code :

```python
import sys
import math
import numpy as np

import Box2D
from Box2D.b2 import fixtureDef
from Box2D.b2 import polygonShape
from Box2D.b2 import contactListener

import gym
from gym import spaces
from gym.envs.box2d.car_dynamics import Car
from gym.utils import seeding, EzPickle

import pyglet

pyglet.options["debug_gl"] = False
from pyglet import gl

STATE_W = 96  # less than Atari 160x192
STATE_H = 96
VIDEO_W = 600
VIDEO_H = 400
WINDOW_W = 1000
WINDOW_H = 800

SCALE = 6.0  # Track scale
TRACK_RAD = 900 / SCALE  # Track is heavily morphed circle with this radius
PLAYFIELD = 2000 / SCALE  # Game over boundary
FPS = 50  # Frames per second
ZOOM = 2.7  # Camera zoom
ZOOM_FOLLOW = True  # Set to False for fixed view (don't use zoom)


TRACK_DETAIL_STEP = 21 / SCALE
TRACK_TURN_RATE = 0.31
TRACK_WIDTH = 40 / SCALE
BORDER = 8 / SCALE
BORDER_MIN_COUNT = 4

ROAD_COLOR = [0.4, 0.4, 0.4]


class FrictionDetector(contactListener):
    def __init__(self, env):
        contactListener.__init__(self)
        self.env = env

    def BeginContact(self, contact):
        self._contact(contact, True)

    def EndContact(self, contact):
        self._contact(contact, False)

    def _contact(self, contact, begin):
        tile = None
        obj = None
        u1 = contact.fixtureA.body.userData
        u2 = contact.fixtureB.body.userData
        if u1 and "road_friction" in u1.__dict__:
            tile = u1
            obj = u2
        if u2 and "road_friction" in u2.__dict__:
            tile = u2
            obj = u1
        if not tile:
            return

        tile.color[0] = ROAD_COLOR[0]
        tile.color[1] = ROAD_COLOR[1]
        tile.color[2] = ROAD_COLOR[2]
        if not obj or "tiles" not in obj.__dict__:
            return
        if begin:
            obj.tiles.add(tile)
            if not tile.road_visited:
                tile.road_visited = True
                self.env.reward += 1000.0 / len(self.env.track)
                self.env.tile_visited_count += 1
        else:
            obj.tiles.remove(tile)


class CarRacing(gym.Env, EzPickle):
    metadata = {
        "render.modes": ["human", "rgb_array", "state_pixels"],
        "video.frames_per_second": FPS,
    }

    def __init__(self, verbose=1):
        EzPickle.__init__(self)
        self.seed()
        self.contactListener_keepref = FrictionDetector(self)
        self.world = Box2D.b2World((0, 0),
contactListener=self.contactListener_keepref)
        self.viewer = None
        self.invisible_state_window = None
        self.invisible_video_window = None
        self.road = None
        self.car = None
        self.reward = 0.0
        self.prev_reward = 0.0
        self.verbose = verbose
        self.fd_tile = fixtureDef(
            shape=polygonShape(vertices=[(0, 0), (1, 0), (1, -1), (0, -1)])
        )

        self.action_space = spaces.Box(
            np.array([-1, 0, 0]).astype(np.float32),
            np.array([+1, +1, +1]).astype(np.float32),
        )  # steer, gas, brake

        self.observation_space = spaces.Box(
            low=0, high=255, shape=(STATE_H, STATE_W, 3), dtype=np.uint8
        )

    def seed(self, seed=None):
        self.np_random, seed = seeding.np_random(seed)
        return [seed]

    def _destroy(self):
        if not self.road:
            return
        for t in self.road:
            self.world.DestroyBody(t)
        self.road = []
        self.car.destroy()

    def _create_track(self):
        CHECKPOINTS = 12

        # Create checkpoints
        checkpoints = []
        for c in range(CHECKPOINTS):
            noise = self.np_random.uniform(0, 2 * math.pi * 1 / CHECKPOINTS)
            alpha = 2 * math.pi * c / CHECKPOINTS + noise
            rad = self.np_random.uniform(TRACK_RAD / 3, TRACK_RAD)

            if c == 0:
                alpha = 0
                rad = 1.5 * TRACK_RAD
            if c == CHECKPOINTS - 1:
                alpha = 2 * math.pi * c / CHECKPOINTS
                self.start_alpha = 2 * math.pi * (-0.5) / CHECKPOINTS
                rad = 1.5 * TRACK_RAD

            checkpoints.append((alpha, rad * math.cos(alpha), rad * math.sin(alpha)))
        self.road = []

        # Go from one checkpoint to another to create track
        x, y, beta = 1.5 * TRACK_RAD, 0, 0
        dest_i = 0
        laps = 0
        track = []
        no_freeze = 2500
        visited_other_side = False
        while True:
            alpha = math.atan2(y, x)
            if visited_other_side and alpha > 0:
                laps += 1
                visited_other_side = False
            if alpha < 0:
                visited_other_side = True
                alpha += 2 * math.pi

            while True:  # Find destination from checkpoints
                failed = True

                while True:
                    dest_alpha, dest_x, dest_y = checkpoints[dest_i %
len(checkpoints)]
                    if alpha <= dest_alpha:
                        failed = False
                        break
                    dest_i += 1
                    if dest_i % len(checkpoints) == 0:
                        break

                if not failed:
                    break

                alpha -= 2 * math.pi
                continue
            r1x = math.cos(beta)
            r1y = math.sin(beta)
            p1x = -r1y
            p1y = r1x
            dest_dx = dest_x - x  # vector towards destination
            dest_dy = dest_y - y
            # destination vector projected on rad:
            proj = r1x * dest_dx + r1y * dest_dy
            while beta - alpha > 1.5 * math.pi:
                beta -= 2 * math.pi
            while beta - alpha < -1.5 * math.pi:
                beta += 2 * math.pi
            prev_beta = beta
            proj *= SCALE
            if proj > 0.3:
                beta -= min(TRACK_TURN_RATE, abs(0.001 * proj))
            if proj < -0.3:
                beta += min(TRACK_TURN_RATE, abs(0.001 * proj))
            x += p1x * TRACK_DETAIL_STEP
            y += p1y * TRACK_DETAIL_STEP
            track.append((alpha, prev_beta * 0.5 + beta * 0.5, x, y))
            if laps > 4:
                break
            no_freeze -= 1
            if no_freeze == 0:
                break

        # Find closed loop range i1..i2, first loop should be ignored, second is OK
        i1, i2 = -1, -1
        i = len(track)
        while True:
            i -= 1
            if i == 0:
                return False  # Failed
            pass_through_start = (
                track[i][0] > self.start_alpha and track[i - 1][0] <= self.start_alpha
            )
            if pass_through_start and i2 == -1:
                i2 = i
            elif pass_through_start and i1 == -1:
                i1 = i
                break
        if self.verbose == 1:
            print("Track generation: %i..%i -> %i-tiles track" % (i1, i2, i2 - i1))
        assert i1 != -1
        assert i2 != -1

        track = track[i1 : i2 - 1]

        first_beta = track[0][1]
        first_perp_x = math.cos(first_beta)
        first_perp_y = math.sin(first_beta)
        # Length of perpendicular jump to put together head and tail
        well_glued_together = np.sqrt(
            np.square(first_perp_x * (track[0][2] - track[-1][2]))
            + np.square(first_perp_y * (track[0][3] - track[-1][3]))
        )
        if well_glued_together > TRACK_DETAIL_STEP:
            return False
        # Red-white border on hard turns
        border = [False] * len(track)
        for i in range(len(track)):
            good = True
            oneside = 0
            for neg in range(BORDER_MIN_COUNT):
                beta1 = track[i - neg - 0][1]
                beta2 = track[i - neg - 1][1]
                good &= abs(beta1 - beta2) > TRACK_TURN_RATE * 0.2
                oneside += np.sign(beta1 - beta2)
            good &= abs(oneside) == BORDER_MIN_COUNT
            border[i] = good
        for i in range(len(track)):
            for neg in range(BORDER_MIN_COUNT):
                border[i - neg] |= border[i]

        # Create tiles
        for i in range(len(track)):
            alpha1, beta1, x1, y1 = track[i]
            alpha2, beta2, x2, y2 = track[i - 1]
            road1_l = (
                x1 - TRACK_WIDTH * math.cos(beta1),
                y1 - TRACK_WIDTH * math.sin(beta1),
            )
            road1_r = (
                x1 + TRACK_WIDTH * math.cos(beta1),
                y1 + TRACK_WIDTH * math.sin(beta1),
            )
            road2_l = (
                x2 - TRACK_WIDTH * math.cos(beta2),
                y2 - TRACK_WIDTH * math.sin(beta2),
            )
            road2_r = (
                x2 + TRACK_WIDTH * math.cos(beta2),
                y2 + TRACK_WIDTH * math.sin(beta2),
            )
            vertices = [road1_l, road1_r, road2_r, road2_l]
            self.fd_tile.shape.vertices = vertices
            t = self.world.CreateStaticBody(fixtures=self.fd_tile)
            t.userData = t
            c = 0.01 * (i % 3)
            t.color = [ROAD_COLOR[0] + c, ROAD_COLOR[1] + c, ROAD_COLOR[2] + c]
            t.road_visited = False
            t.road_friction = 1.0
            t.fixtures[0].sensor = True
            self.road_poly.append(([road1_l, road1_r, road2_r, road2_l], t.color))
            self.road.append(t)
            if border[i]:
                side = np.sign(beta2 - beta1)
                b1_l = (
                    x1 + side * TRACK_WIDTH * math.cos(beta1),
                    y1 + side * TRACK_WIDTH * math.sin(beta1),
                )
                b1_r = (
                    x1 + side * (TRACK_WIDTH + BORDER) * math.cos(beta1),
                    y1 + side * (TRACK_WIDTH + BORDER) * math.sin(beta1),
                )
                b2_l = (
                    x2 + side * TRACK_WIDTH * math.cos(beta2),
                    y2 + side * TRACK_WIDTH * math.sin(beta2),
                )
                b2_r = (
                    x2 + side * (TRACK_WIDTH + BORDER) * math.cos(beta2),
                    y2 + side * (TRACK_WIDTH + BORDER) * math.sin(beta2),
                )
                self.road_poly.append(
                    ([b1_l, b1_r, b2_r, b2_l], (1, 1, 1) if i % 2 == 0 else (1, 0, 0))
                )
        self.track = track
        return True

    def reset(self):
        self._destroy()
        self.reward = 0.0
        self.prev_reward = 0.0
        self.tile_visited_count = 0
        self.t = 0.0
        self.road_poly = []

        while True:
            success = self._create_track()
            if success:
                break
            if self.verbose == 1:
                print(
                    "retry to generate track (normal if there are not many"
                    "instances of this message)"
                )
        self.car = Car(self.world, *self.track[0][1:4])

        return self.step(None)[0]

    def step(self, action):
        if action is not None:
            self.car.steer(-action[0])
            self.car.gas(action[1])
            self.car.brake(action[2])

        self.car.step(1.0 / FPS)
        self.world.Step(1.0 / FPS, 6 * 30, 2 * 30)
        self.t += 1.0 / FPS

        self.state = self.render("state_pixels")

        step_reward = 0
        done = False
        if action is not None:  # First step without action, called from reset()
            self.reward -= 0.1
            # We actually don't want to count fuel spent, we want car to be faster.
            # self.reward -=  10 * self.car.fuel_spent / ENGINE_POWER
            self.car.fuel_spent = 0.0
            step_reward = self.reward - self.prev_reward
            self.prev_reward = self.reward
            if self.tile_visited_count == len(self.track):
                done = True
            x, y = self.car.hull.position
            if abs(x) > PLAYFIELD or abs(y) > PLAYFIELD:
                done = True
                step_reward = -100

        return self.state, step_reward, done, {}
    def render(self, mode="human"):
        assert mode in ["human", "state_pixels", "rgb_array"]
        if self.viewer is None:
            from gym.envs.classic_control import rendering

            self.viewer = rendering.Viewer(WINDOW_W, WINDOW_H)
            self.score_label = pyglet.text.Label(
                "0000",
                font_size=36,
                x=20,
                y=WINDOW_H * 2.5 / 40.00,
                anchor_x="left",
                anchor_y="center",
                color=(255, 255, 255, 255),
            )
            self.transform = rendering.Transform()

        if "t" not in self.__dict__:
            return  # reset() not called yet

        # Animate zoom first second:
        zoom = 0.1 * SCALE * max(1 - self.t, 0) + ZOOM * SCALE * min(self.t, 1)
        scroll_x = self.car.hull.position[0]
        scroll_y = self.car.hull.position[1]
        angle = -self.car.hull.angle
        vel = self.car.hull.linearVelocity
        if np.linalg.norm(vel) > 0.5:
            angle = math.atan2(vel[0], vel[1])
        self.transform.set_scale(zoom, zoom)
        self.transform.set_translation(
            WINDOW_W / 2
            - (scroll_x * zoom * math.cos(angle) - scroll_y * zoom * math.sin(angle)),
            WINDOW_H / 4
            - (scroll_x * zoom * math.sin(angle) + scroll_y * zoom * math.cos(angle)),
        )
        self.transform.set_rotation(angle)

        self.car.draw(self.viewer, mode != "state_pixels")

        arr = None
        win = self.viewer.window
        win.switch_to()
        win.dispatch_events()

        win.clear()
        t = self.transform
        if mode == "rgb_array":
            VP_W = VIDEO_W
            VP_H = VIDEO_H
        elif mode == "state_pixels":
            VP_W = STATE_W
            VP_H = STATE_H
        else:
            pixel_scale = 1
            if hasattr(win.context, "_nscontext"):
                pixel_scale = (
                    win.context._nscontext.view().backingScaleFactor()
                )  # pylint: disable=protected-access
            VP_W = int(pixel_scale * WINDOW_W)
            VP_H = int(pixel_scale * WINDOW_H)
        gl.glViewport(0, 0, VP_W, VP_H)
        t.enable()
        self.render_road()
        for geom in self.viewer.onetime_geoms:
            geom.render()
        self.viewer.onetime_geoms = []
        t.disable()
        self.render_indicators(WINDOW_W, WINDOW_H)

        if mode == "human":
            win.flip()
            return self.viewer.isopen

        image_data = (
            pyglet.image.get_buffer_manager().get_color_buffer().get_image_data()
        )
        arr = np.fromstring(image_data.get_data(), dtype=np.uint8, sep="")
        arr = arr.reshape(VP_H, VP_W, 4)
        arr = arr[::-1, :, 0:3]

        return arr

    def close(self):
        if self.viewer is not None:
            self.viewer.close()
            self.viewer = None

    def render_road(self):
        colors = [0.4, 0.8, 0.4, 1.0] * 4
        polygons_ = [
            +PLAYFIELD,
            +PLAYFIELD,
            0,
            +PLAYFIELD,
            -PLAYFIELD,
            0,
            -PLAYFIELD,
            -PLAYFIELD,
            0,
            -PLAYFIELD,
            +PLAYFIELD,
            0,
        ]

        k = PLAYFIELD / 20.0
        colors.extend([0.4, 0.9, 0.4, 1.0] * 4 * 20 * 20)
        for x in range(-20, 20, 2):
            for y in range(-20, 20, 2):
                polygons_.extend(
                    [
                        k * x + k,
                        k * y + 0,
                        0,
                        k * x + 0,
                        k * y + 0,
                        0,
                        k * x + 0,
                        k * y + k,
                        0,
                        k * x + k,
                        k * y + k,
                        0,
                    ]
                )

        for poly, color in self.road_poly:
            colors.extend([color[0], color[1], color[2], 1] * len(poly))
            for p in poly:
                polygons_.extend([p[0], p[1], 0])

        vl = pyglet.graphics.vertex_list(
            len(polygons_) // 3, ("v3f", polygons_), ("c4f", colors)
        )  # gl.GL_QUADS,
        vl.draw(gl.GL_QUADS)
        vl.delete()

    def render_indicators(self, W, H):
        s = W / 40.0
        h = H / 40.0
        colors = [0, 0, 0, 1] * 4
        polygons = [W, 0, 0, W, 5 * h, 0, 0, 5 * h, 0, 0, 0, 0]

        def vertical_ind(place, val, color):
            colors.extend([color[0], color[1], color[2], 1] * 4)
            polygons.extend(
                [
                    place * s,
                    h + h * val,
                    0,
                    (place + 1) * s,
                    h + h * val,
                    0,
                    (place + 1) * s,
                    h,
                    0,
                    (place + 0) * s,
                    h,
                    0,
                ]
            )

        def horiz_ind(place, val, color):
            colors.extend([color[0], color[1], color[2], 1] * 4)
            polygons.extend(
                [
                    (place + 0) * s,
                    4 * h,
                    0,
                    (place + val) * s,
                    4 * h,
                    0,
                    (place + val) * s,
                    2 * h,
                    0,
                    (place + 0) * s,
                    2 * h,
                    0,
                ]
            )

        true_speed = np.sqrt(
            np.square(self.car.hull.linearVelocity[0])
            + np.square(self.car.hull.linearVelocity[1])
        )

        vertical_ind(5, 0.02 * true_speed, (1, 1, 1))
        vertical_ind(7, 0.01 * self.car.wheels[0].omega, (0.0, 0, 1))  # ABS sensors
        vertical_ind(8, 0.01 * self.car.wheels[1].omega, (0.0, 0, 1))
        vertical_ind(9, 0.01 * self.car.wheels[2].omega, (0.2, 0, 1))
        vertical_ind(10, 0.01 * self.car.wheels[3].omega, (0.2, 0, 1))
        horiz_ind(20, -10.0 * self.car.wheels[0].joint.angle, (0, 1, 0))
        horiz_ind(30, -0.8 * self.car.hull.angularVelocity, (1, 0, 0))
        vl = pyglet.graphics.vertex_list(
            len(polygons) // 3, ("v3f", polygons), ("c4f", colors)
        )  # gl.GL_QUADS,
        vl.draw(gl.GL_QUADS)
        vl.delete()
        self.score_label.text = "%04i" % self.reward
        self.score_label.draw()


if __name__ == "__main__":
    from pyglet.window import key

    a = np.array([0.0, 0.0, 0.0])

    def key_press(k, mod):
        global restart
        if k == 0xFF0D:
            restart = True
        if k == key.LEFT:
            a[0] = -1.0
        if k == key.RIGHT:
            a[0] = +1.0
        if k == key.UP:
            a[1] = +1.0
        if k == key.DOWN:
            a[2] = +0.8  # set 1.0 for wheels to block to zero rotation

    def key_release(k, mod):
        if k == key.LEFT and a[0] == -1.0:
            a[0] = 0
        if k == key.RIGHT and a[0] == +1.0:
            a[0] = 0
        if k == key.UP:
            a[1] = 0
        if k == key.DOWN:
            a[2] = 0

    env = CarRacing()
    env.render()
    env.viewer.window.on_key_press = key_press
    env.viewer.window.on_key_release = key_release
    record_video = False
    if record_video:
        from gym.wrappers.monitor import Monitor

        env = Monitor(env, "/tmp/video-test", force=True)
    isopen = True
    while isopen:
        env.reset()
        total_reward = 0.0
        steps = 0
        restart = False
        while True:
            s, r, done, info = env.step(a)
            total_reward += r
            if steps % 200 == 0 or done:
                print("\naction " + str([f"{x:+0.2f}" for x in a]))
                print(f"step {steps} total_reward {total_reward:+0.2f}")
            steps += 1
            isopen = env.render()
            if done or restart or isopen == False:
                break
    env.close()
```

Output :

```
$ python3 Car_Racing.py
Track generation: 1028..1289 -> 261-tiles track
Car_Racing.py:474: DeprecationWarning: The binary mode of fromstring is deprecated, as
it behaves surprisingly on unicode inputs. Use frombuffer instead
  arr = np.fromstring(image_data.get_data(), dtype=np.uint8, sep="")

action ['+0.00', '+0.00', '+0.00']
step 0 total_reward +7.59

action ['+0.00', '+0.00', '+0.00']
step 200 total_reward +37.59

action ['+0.00', '+0.00', '+0.00']
step 400 total_reward +206.05

action ['+0.00', '+0.00', '+0.00']
step 600 total_reward +347.59

action ['+0.00', '+0.00', '+0.00']
step 800 total_reward +462.21

action ['+0.00', '+0.00', '+0.00']
step 1000 total_reward +599.90

action ['+0.00', '+0.00', '+0.00']
step 1200 total_reward +779.90

action ['+0.00', '+0.00', '+0.00']
step 1400 total_reward +840.67

action ['-1.00', '+0.00', '+0.00']
step 1571 total_reward +842.80
Track generation: 1063..1333 -> 270-tiles track

action ['-1.00', '+0.00', '+0.00']
step 0 total_reward +7.33

action ['+0.00', '+0.00', '+0.00']
step 200 total_reward +69.12

action ['+1.00', '+0.00', '+0.00']
step 400 total_reward +231.28

action ['-1.00', '+0.00', '+0.00']
step 600 total_reward +400.87
```

```
action ['+0.00', '+0.00', '+0.00']
step 800 total_reward +559.31

action ['+0.00', '+0.00', '+0.00']
step 1000 total_reward +710.31

action ['+0.00', '+0.00', '+0.00']
step 1200 total_reward +861.31

action ['+0.00', '+0.00', '+0.00']
step 1222 total_reward +877.70
Track generation: 1249..1565 -> 316-tiles track
```



C. Roulette

Code :

```
import gym
import gym_toytext
import numpy as np
from IPython.display import clear_output
from time import sleep
import matplotlib.pyplot as plt

env = gym.make('Roulette-v0')
```

```
action_space_size = env.action_space.n
state_space_size = env.observation_space.n
print(f"Number of actions available: {action_space_size}")
print(f"Number of states defined: {state_space_size}")
print(f"Therefore, Q-Table with {action_space_size} columns and {state_space_size}
rows will be created")

q_table = np.random.random([state_space_size, action_space_size])
#OR
# q_table = np.zeros([state_space_size, action_space_size])
print(f"Q-Table shape: {q_table.shape}")

sleep(5)

# Hyperparameters
TOTAL_EPISODES = 5_000 #Number of epsiodes to train the algorithm
MAX_STEPS = 150 #Max steps an agent can take during an episode

LEARNING_RATE = 0.1
GAMMA = 0.95 # Discount (close to 0 makes it greedy, close to 1 considers long term)

# Exploration Parameters
epsilon = 1
START_EPSILON_DECAYING = 1
END_EPSILON_DECAYING = TOTAL_EPISODES // 2
DECAY_RATE = epsilon/(END_EPSILON_DECAYING - START_EPSILON_DECAYING)

def print_frames(frames):
    total_reward = 0
    for i, frame in enumerate(frames):
        clear_output(wait=True)
        print('\n*********************')
        print(f"Episode: {frame['episode']}")
        print(f'Round: {i + 1}')
        print(f"Action: {frame['action']}")
        print(f"Reward: {frame['reward']}")
        total_reward += frame['reward']
        print(f"Total reward so far: {total_reward}")
        sleep(0.1)


def edit_reward(info):
    print(info)

env.reset()
history = {'steps': [], 'total_score': [], 'episode_number': []}
best_frames = []
high_score = 0

for episode in range(TOTAL_EPISODES):

    state = env.reset()
    step = 0
    frames = []
    current_score = 0
    done = False

    for step in range(MAX_STEPS):

        if np.random.random() > epsilon: #This is exploitation
            action = np.argmax(q_table[state, :]) #Current state, max value
        else: #This is exploration
            action = np.random.randint(0, action_space_size)

        new_state, reward, done, info = env.step(action)

        current_score += reward

        frames.append({
            'episode': episode,
            'action': action,
            'reward': reward
        })

        q_table[state, action] = (1 - LEARNING_RATE) * q_table[state, action] + LEARNING_RATE * (reward + GAMMA * np.max(q_table[new_state, :]))

        if done:

            history['steps'].append(step + 1)
            history['total_score'].append(current_score)
            history['episode_number'].append(episode)

            if current_score >= high_score:
                high_score = current_score
                best_frames = frames.copy()
            break

        state = new_state

    if END_EPSILON_DECAYING >= episode >= START_EPSILON_DECAYING:
        epsilon -= DECAY_RATE

env.close()
print_frames(best_frames)
print(f"\nQ-Table after {TOTAL_EPISODES} episodes")
print(q_table)
print(f'The best score after running {TOTAL_EPISODES} episodes: {high_score}')

# Total score
plt.subplot(2, 1, 1)
plt.scatter(history['episode_number'], history['total_score'], s = 5, label = 'score', color = 'blue')
plt.xlabel('Number of episodes')
plt.ylabel('Score')
plt.legend(loc = 4)

# Number of steps
plt.subplot(2, 1, 2)
plt.scatter(history['episode_number'], history['steps'], s = 5, label = 'steps', color = 'red')
plt.xlabel('Number of episodes')
plt.ylabel('Number of steps')
plt.legend(loc = 4)
plt.show()
```

Output :

```
$ python3 Roulette.py
Number of actions available: 38
Number of states defined: 1
Therefore, Q-Table with 38 columns and 1 rows will be created
Q-Table shape: (1, 38)

*********************
Episode: 2280
Round: 1
Action: 33
Reward: 1.0
Total reward so far: 1.0

*********************
Episode: 2280
Round: 2
Action: 33
Reward: 1.0
Total reward so far: 2.0

*********************
Episode: 2280
Round: 3
Action: 33
Reward: 1.0
Total reward so far: 3.0

*********************
Episode: 2280
Round: 4
Action: 33
Reward: 1.0
Total reward so far: 4.0

*********************
Episode: 2280
Round: 5
Action: 33
Reward: 1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 6
Action: 33
Reward: -1.0
Total reward so far: 4.0

*********************
Episode: 2280
Round: 7
Action: 33
Reward: -1.0
Total reward so far: 3.0

*********************
Episode: 2280
Round: 8
Action: 33
Reward: 1.0
Total reward so far: 4.0

*********************
```

```
Episode: 2280
Round: 9
Action: 33
Reward: 1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 10
Action: 33
Reward: 1.0
Total reward so far: 6.0

*********************
Episode: 2280
Round: 11
Action: 33
Reward: -1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 12
Action: 33
Reward: -1.0
Total reward so far: 4.0

*********************
Episode: 2280
Round: 13
Action: 16
Reward: 1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 14
Action: 33
Reward: -1.0
Total reward so far: 4.0

*********************
Episode: 2280
Round: 15
Action: 33
Reward: -1.0
Total reward so far: 3.0

*********************
Episode: 2280
Round: 16
Action: 33
Reward: -1.0
Total reward so far: 2.0

*********************
Episode: 2280
Round: 17
Action: 16
Reward: 1.0
Total reward so far: 3.0
```

```
********************
Episode: 2280
Round: 18
Action: 16
Reward: 1.0
Total reward so far: 4.0

********************
Episode: 2280
Round: 19
Action: 16
Reward: -1.0
Total reward so far: 3.0

********************
Episode: 2280
Round: 20
Action: 16
Reward: -1.0
Total reward so far: 2.0

********************
Episode: 2280
Round: 21
Action: 16
Reward: 1.0
Total reward so far: 3.0

********************
Episode: 2280
Round: 22
Action: 16
Reward: 1.0
Total reward so far: 4.0

********************
Episode: 2280
Round: 23
Action: 16
Reward: 1.0
Total reward so far: 5.0

********************
Episode: 2280
Round: 24
Action: 16
Reward: 1.0
Total reward so far: 6.0

********************
Episode: 2280
Round: 25
Action: 16
Reward: -1.0
Total reward so far: 5.0

********************
Episode: 2280
Round: 26
Action: 16
```

```
Reward: 1.0
Total reward so far: 6.0

*********************
Episode: 2280
Round: 27
Action: 16
Reward: -1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 28
Action: 16
Reward: 1.0
Total reward so far: 6.0

*********************
Episode: 2280
Round: 29
Action: 16
Reward: 1.0
Total reward so far: 7.0

*********************
Episode: 2280
Round: 30
Action: 16
Reward: 1.0
Total reward so far: 8.0

*********************
Episode: 2280
Round: 31
Action: 16
Reward: -1.0
Total reward so far: 7.0

*********************
Episode: 2280
Round: 32
Action: 16
Reward: -1.0
Total reward so far: 6.0

*********************
Episode: 2280
Round: 33
Action: 16
Reward: 1.0
Total reward so far: 7.0

*********************
Episode: 2280
Round: 34
Action: 16
Reward: -1.0
Total reward so far: 6.0

*********************
Episode: 2280
```

```
Round: 35
Action: 16
Reward: 1.0
Total reward so far: 7.0

*********************
Episode: 2280
Round: 36
Action: 16
Reward: 1.0
Total reward so far: 8.0

*********************
Episode: 2280
Round: 37
Action: 16
Reward: -1.0
Total reward so far: 7.0

*********************
Episode: 2280
Round: 38
Action: 16
Reward: -1.0
Total reward so far: 6.0

*********************
Episode: 2280
Round: 39
Action: 16
Reward: -1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 40
Action: 16
Reward: 1.0
Total reward so far: 6.0

*********************
Episode: 2280
Round: 41
Action: 16
Reward: -1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 42
Action: 16
Reward: 1.0
Total reward so far: 6.0

*********************
Episode: 2280
Round: 43
Action: 16
Reward: -1.0
Total reward so far: 5.0
```

```
*********************
Episode: 2280
Round: 44
Action: 16
Reward: -1.0
Total reward so far: 4.0

*********************
Episode: 2280
Round: 45
Action: 16
Reward: 1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 46
Action: 16
Reward: -1.0
Total reward so far: 4.0

*********************
Episode: 2280
Round: 47
Action: 16
Reward: 1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 48
Action: 31
Reward: -1.0
Total reward so far: 4.0

*********************
Episode: 2280
Round: 49
Action: 25
Reward: 1.0
Total reward so far: 5.0

*********************
Episode: 2280
Round: 50
Action: 16
Reward: 1.0
Total reward so far: 6.0

*********************
Episode: 2280
Round: 51
Action: 16
Reward: 1.0
Total reward so far: 7.0

*********************
Episode: 2280
Round: 52
Action: 16
Reward: 1.0
```

```
Total reward so far: 8.0

*********************
Episode: 2280
Round: 53
Action: 16
Reward: -1.0
Total reward so far: 7.0

*********************
Episode: 2280
Round: 54
Action: 0
Reward: 36.0
Total reward so far: 43.0

*********************
Episode: 2280
Round: 55
Action: 0
Reward: -1.0
Total reward so far: 42.0

*********************
Episode: 2280
Round: 56
Action: 20
Reward: -1.0
Total reward so far: 41.0

*********************
Episode: 2280
Round: 57
Action: 0
Reward: -1.0
Total reward so far: 40.0

*********************
Episode: 2280
Round: 58
Action: 0
Reward: -1.0
Total reward so far: 39.0

*********************
Episode: 2280
Round: 59
Action: 0
Reward: -1.0
Total reward so far: 38.0

*********************
Episode: 2280
Round: 60
Action: 0
Reward: -1.0
Total reward so far: 37.0

*********************
Episode: 2280
Round: 61
```

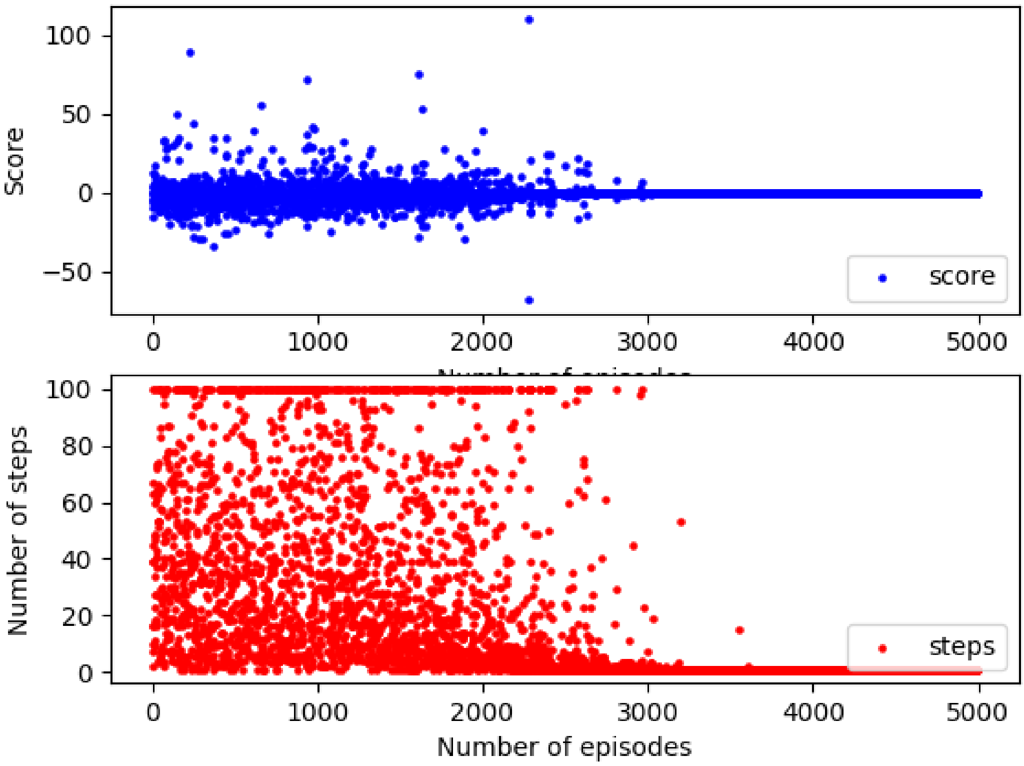
```
Action: 0
Reward: -1.0
Total reward so far: 36.0

*********************
Episode: 2280
Round: 62
Action: 0
Reward: -1.0
Total reward so far: 35.0

*********************
Episode: 2280
Round: 63
Action: 0
Reward: -1.0
Total reward so far: 34.0

*********************
Episode: 2280
Round: 64
Action: 0
Reward: -1.0
Total reward so far: 33.0

*********************
Episode: 2280
Round: 65
Action: 0
Reward: -1.0
Total reward so far: 32.0

*********************
Episode: 2280
Round: 66
Action: 0
Reward: -1.0
Total reward so far: 31.0

*********************
Episode: 2280
Round: 67
Action: 0
Reward: -1.0
Total reward so far: 30.0

*********************
Episode: 2280
Round: 68
Action: 0
Reward: -1.0
Total reward so far: 29.0

*********************
Episode: 2280
Round: 69
Action: 0
Reward: -1.0
Total reward so far: 28.0

*********************
```

```
Episode: 2280
Round: 70
Action: 0
Reward: -1.0
Total reward so far: 27.0

*********************
Episode: 2280
Round: 71
Action: 0
Reward: -1.0
Total reward so far: 26.0

*********************
Episode: 2280
Round: 72
Action: 0
Reward: -1.0
Total reward so far: 25.0

*********************
Episode: 2280
Round: 73
Action: 0
Reward: 36.0
Total reward so far: 61.0

*********************
Episode: 2280
Round: 74
Action: 0
Reward: -1.0
Total reward so far: 60.0

*********************
Episode: 2280
Round: 75
Action: 0
Reward: -1.0
Total reward so far: 59.0

*********************
Episode: 2280
Round: 76
Action: 0
Reward: -1.0
Total reward so far: 58.0

*********************
Episode: 2280
Round: 77
Action: 0
Reward: -1.0
Total reward so far: 57.0

*********************
Episode: 2280
Round: 78
Action: 0
Reward: -1.0
Total reward so far: 56.0
```

```
*********************
Episode: 2280
Round: 79
Action: 0
Reward: 36.0
Total reward so far: 92.0

*********************
Episode: 2280
Round: 80
Action: 0
Reward: -1.0
Total reward so far: 91.0

*********************
Episode: 2280
Round: 81
Action: 0
Reward: -1.0
Total reward so far: 90.0

*********************
Episode: 2280
Round: 82
Action: 0
Reward: -1.0
Total reward so far: 89.0

*********************
Episode: 2280
Round: 83
Action: 0
Reward: -1.0
Total reward so far: 88.0

*********************
Episode: 2280
Round: 84
Action: 0
Reward: -1.0
Total reward so far: 87.0

*********************
Episode: 2280
Round: 85
Action: 0
Reward: -1.0
Total reward so far: 86.0

*********************
Episode: 2280
Round: 86
Action: 0
Reward: -1.0
Total reward so far: 85.0

*********************
Episode: 2280
Round: 87
Action: 0
```

```
Reward: -1.0
Total reward so far: 84.0

*********************
Episode: 2280
Round: 88
Action: 5
Reward: 1.0
Total reward so far: 85.0

*********************
Episode: 2280
Round: 89
Action: 0
Reward: -1.0
Total reward so far: 84.0

*********************
Episode: 2280
Round: 90
Action: 0
Reward: -1.0
Total reward so far: 83.0

*********************
Episode: 2280
Round: 91
Action: 0
Reward: -1.0
Total reward so far: 82.0

*********************
Episode: 2280
Round: 92
Action: 0
Reward: -1.0
Total reward so far: 81.0

*********************
Episode: 2280
Round: 93
Action: 0
Reward: -1.0
Total reward so far: 80.0

*********************
Episode: 2280
Round: 94
Action: 0
Reward: -1.0
Total reward so far: 79.0

*********************
Episode: 2280
Round: 95
Action: 0
Reward: -1.0
Total reward so far: 78.0

*********************
Episode: 2280
```

```
Round: 96
Action: 0
Reward: -1.0
Total reward so far: 77.0

********************
Episode: 2280
Round: 97
Action: 25
Reward: -1.0
Total reward so far: 76.0

********************
Episode: 2280
Round: 98
Action: 0
Reward: 36.0
Total reward so far: 112.0

********************
Episode: 2280
Round: 99
Action: 0
Reward: -1.0
Total reward so far: 111.0

********************
Episode: 2280
Round: 100
Action: 0
Reward: -1.0
Total reward so far: 110.0

Q-Table after 5000 episodes
[[-3.07583358e-02 -4.19334032e-02 -5.99201427e-02 -3.76336086e-02
  -9.83211895e-02 -1.35208913e-02 -1.48916007e-02 -7.34094117e-02
  -2.78890312e-02 -2.38599619e-02 -8.43982923e-02 -2.09545907e-02
  -3.38959585e-02 -7.01535786e-02 -5.08198245e-02 -8.84553123e-03
  -7.96039430e-02 -7.43652260e-02 -2.86639392e-02 -6.85138461e-02
  -7.48503926e-02 -3.73022417e-02 -1.20240760e-02 -2.79969394e-02
  -1.20484291e-02 -6.39220388e-02 -4.02999360e-02 -3.19100504e-03
  -6.82564864e-02 -7.04195457e-02 -2.58737313e-04 -1.96667366e-02
  -4.47963458e-02 -4.82113761e-02 -6.84705007e-02 -3.36668727e-02
  -2.11633852e-02  1.57973762e-06]]
The best score after running 5000 episodes: 110.0
```


Score
100
50
0
−50
score
0
1000
2000
3000
4000
5000
Number of episodes
Number of steps
100
80
60
40
20
0
steps
0
1000
2000
3000
4000
5000
Number of episodes