```python
from utils import *
from constants import *
from agent import *
from task import *
import numpy as np
import matplotlib.pyplot as plt


def experiment_1():
    # Initializing task T at random position
    task = Task(task_capacity=1, task_radius=50)

    # Initializing agent R1 at random position
    r1 = Agent()

    # Starting simulation
    results = np.zeros((NUM_EPOCHS_A))
    completed_tasks = 0
    for i in range(NUM_EPOCHS_A):
        # Updating movement (velocity and position) of agent
        r1.update_velocity()
        r1.update_pos()

        # Checking if the agent is within the task radius, adding to
        # completed tasks and creating a new one if so
        if distance_euclid(task.pos, r1.pos) < task.task_radius:
            completed_tasks += 1
            task = Task(task_capacity=TASK_CAPACITY_A, task_radius=TASK_RADIUS_A)
        results[i] = completed_tasks

    # Plotting results
    x = np.linspace(1, NUM_EPOCHS_A, NUM_EPOCHS_A)
    y = results
    plt.plot(x, y)
    plt.title("TASK A)")
    plt.xlabel("Time (# of epochs)")
    plt.ylabel("# of tasks solved")
    plt.savefig(fname='figures/task_a')
    plt.close()


def experiment_2():

    # Performing experiment with different numbers of agents
    for agent_num in NUM_AGENTS_B:
        # Initializing agents at random positions
        agents = []
        for _ in range(agent_num):
            agent = Agent()
            agents.append(agent)

        # Initializing task T at random position
        task = Task(task_capacity=TASK_CAPACITY_B, task_radius=TASK_RADIUS_B)

        # Starting simulation
        results = np.zeros((NUM_EPOCHS_B))
        completed_tasks = 0
        for i in range(NUM_EPOCHS_B):
            # Updating movement (velocity and position) of agent
            for agent in agents:
                agent.update_velocity()
                agent.update_pos()

                # Checking if the agent is within the task radius, adding to
                # completed tasks and creating a new one if so
                if distance_euclid(task.pos, agent.pos) < task.task_radius:
                    completed_tasks += 1
```

```
                task = Task(task_capacity=TASK_CAPACITY_B, task_radius=TASK_RADIUS_B)
            results[i] = completed_tasks
        x = np.linspace(1, NUM_EPOCHS_B, NUM_EPOCHS_B)
        y = results
        plt.plot(x, y, label=f'R = {agent_num}')
        print(f"Simulations for R = {agent_num} complete.")

    # Plotting results
    plt.title("TASK B)")
    plt.xlabel("Time (# of epochs)")
    plt.ylabel("# of tasks solved")
    plt.legend()
    plt.savefig(fname='figures/task_b')
    plt.close()

def experiment_3():
    # Performing experiment with different numbers of agents
    for agent_num in NUM_AGENTS_C:
        # Initializing agents at random positions
        agents = []
        for _ in range(agent_num):
            agent = Agent()
            agents.append(agent)

        # Initializing task T at random position
        task = Task(task_capacity=TASK_CAPACITY_C, task_radius=TASK_RADIUS_C)

        # Starting simulation
        results = np.zeros((NUM_EPOCHS_C))
        completed_tasks = 0
        for i in range(NUM_EPOCHS_C):
            # Updating movement (velocity and position) of agent
            for agent in agents:
                agent.update_velocity()
                agent.update_pos()

            task_completed = task.sufficient_agents_in_radius(agents)
            if task_completed:
                completed_tasks += 1
                task = Task(task_capacity=TASK_CAPACITY_C, task_radius=TASK_RADIUS_C)
            results[i] = completed_tasks

        x = np.linspace(1, NUM_EPOCHS_C, NUM_EPOCHS_C)
        y = results
        plt.plot(x, y, label=f'R = {agent_num}')
        print(f"Simulations for R = {agent_num} complete.")

    # Plotting results
    plt.title("TASK C)")
    plt.xlabel("Time (# of epochs)")
    plt.ylabel("# of tasks solved")
    plt.legend()
    plt.savefig(fname='figures/task_c')
    plt.close()

def experiment_4():
    # Storing results across all episodes
    episodes = []

    # Performing experiment with different numbers of tasks
    for task_num in NUM_TASKS_D:
        for _ in range(NUM_EPISODES_D):
            # Initializing tasks at random positions
            tasks = []
            for _ in range(task_num):
                task = Task(task_capacity=TASK_CAPACITY_D, task_radius=TASK_RADIUS_D)
                tasks.append(task)
```

```
            # Initializing agents at random positions
            agents = []
            for _ in range(NUM_AGENTS_D): # For purpose of experiment, assuming R=30 agents
                agent = Agent()
                agents.append(agent)

            # Starting simulation
            results = np.zeros((NUM_EPOCHS_D))
            completed_tasks = 0
            for i in range(NUM_EPOCHS_D):
                # Updating movement (velocity and position) of agent
                for agent in agents:
                    agent.update_velocity()
                    agent.update_pos()

                for task_i in range(len(tasks)):
                    task = tasks[task_i]
                    task_completed = task.sufficient_agents_in_radius(agents)
                    if task_completed:
                        completed_tasks += 1
                        tasks[task_i] = Task(task_capacity=TASK_CAPACITY_D,
task_radius=TASK_RADIUS_D)
                results[i] = completed_tasks
            episodes.append(results)
        x = np.linspace(1, NUM_EPOCHS_D, NUM_EPOCHS_D)
        y = np.array(episodes).mean(axis=0)
        plt.plot(x, y, label=f'T = {task_num}')
        print(f"Simulations for T = {task_num} complete.")

    # Plotting results
    plt.title("TASK D)")
    plt.xlabel("Time (# of epochs)")
    plt.ylabel("# of tasks solved")
    plt.legend()
    plt.savefig(fname='figures/task_d')
    plt.close()

def experiment_5():
    # Performing experiment with different communication distances
    for comm_dist in COMM_DISTANCES_E:
        # Initializing tasks at random positions
        tasks = []
        for _ in range(NUM_TASKS_E): # Assuming T=2 tasks
            task = Task(task_capacity=TASK_CAPACITY_E, task_radius=TASK_RADIUS_E)
            tasks.append(task)

        # Initializing agents at random positions
        agents = []
        for _ in range(NUM_AGENTS_E): # Assuming R=30 agents
            agent = Agent(comm_dist=comm_dist)
            agents.append(agent)

        # Starting simulation
        results = np.zeros((NUM_EPOCHS_E))
        completed_tasks = 0
        for i in range(NUM_EPOCHS_E):
            for task_i in range(len(tasks)):
                task = tasks[task_i]
                task_completed = task.sufficient_agents_in_radius(agents)
                if task_completed:
                    completed_tasks += 1
                    tasks[task_i] = Task(task_capacity=TASK_CAPACITY_E,
task_radius=TASK_RADIUS_E)

            # Updating movement (velocity and position) of agent
            for agent in agents:
```

```
                # Callout is performed before updating velocities, so that each
                # agent can evaluate whether conditions are met for it to follow
                # the target_pos it may receive from callout.
                if agent.inside_task_radius:
                    agent.callout(agents)
                agent.update_velocity()
                agent.update_pos()
            results[i] = completed_tasks
        x = np.linspace(1, NUM_EPOCHS_E, NUM_EPOCHS_E)
        y = results
        plt.plot(x, y, label=f'Rd = {comm_dist}')
        print(f"Simulations for Rd = {comm_dist} complete.")

    # Plotting results
    plt.title("TASK E)")
    plt.xlabel("Time (# of epochs)")
    plt.ylabel("# of tasks solved")
    plt.legend()
    plt.savefig(fname='figures/task_e')
    plt.close()

def experiment_6():
    # Performing experiment with different communication distances
    for comm_dist in COMM_DISTANCES_F:
        # Initializing tasks at random positions
        tasks = []
        for _ in range(NUM_TASKS_F): # Assuming T=2 tasks
            task = Task(task_capacity=TASK_CAPACITY_F, task_radius=TASK_RADIUS_F)
            tasks.append(task)

        # Initializing agents at random positions
        agents = []
        for _ in range(NUM_AGENTS_F): # Assuming R=30 agents
            agent = Agent(comm_dist=comm_dist)
            agents.append(agent)

        # Starting simulation
        results = np.zeros((NUM_EPOCHS_F))
        completed_tasks = 0
        for i in range(NUM_EPOCHS_F):
            for task_i in range(len(tasks)):
                task = tasks[task_i]
                task_completed = task.sufficient_agents_in_radius(agents, invoke_calloff=True)
                if task_completed:
                    completed_tasks += 1
                    tasks[task_i] = Task(task_capacity=TASK_CAPACITY_F,
task_radius=TASK_RADIUS_F)

            # Updating movement (velocity and position) of agent
            for agent in agents:
                # Callout is performed before updating velocities, so that each
                # agent can evaluate whether conditions are met for it to follow
                # the target_pos it may receive from callout.
                if agent.inside_task_radius:
                    agent.callout(agents)
                agent.update_velocity()
                agent.update_pos()
            results[i] = completed_tasks
        x = np.linspace(1, NUM_EPOCHS_F, NUM_EPOCHS_F)
        y = results
        plt.plot(x, y, label=f'Rd = {comm_dist}')
        print(f"Simulations for Rd = {comm_dist} complete.")

    # Plotting results
    plt.title("TASK F)")
    plt.xlabel("Time (# of epochs)")
    plt.ylabel("# of tasks solved")
```

```
    plt.legend()
    plt.savefig(fname='figures/task_f')
    plt.close()


if __name__ == "__main__":
    print("\n################### TASK A) ###################")
    experiment_1()
    print("\n################### TASK B) ###################")
    experiment_2()
    print("\n################### TASK C) ###################")
    experiment_3()
    print("\n################### TASK D) ###################")
    experiment_4()
    print("\n################### TASK E) ###################")
    experiment_5()
    print("\n################### TASK F) ###################")
    experiment_6()
```