Execution Guide for Spaceship Agent Simulation in Unity

Required Software and Tools:

1. Unity (Version 2020.2 or Newer)

a. **Purpose:** Game engine for creating and running the simulation.
b. **Download:** **[Unity Download Portal](https://unity.com/download)**

2. Python (Version 3.6 or 3.7)

a. **Purpose:** Running the ML-Agents training environment.
b. **Download:** **[Python Official Site](https://www.python.org/downloads/)**

3. ML-Agents Toolkit

a. **Purpose:** Toolkit for integrating machine learning in Unity.
b. **Installation:** Install via pip with `pip install mlagents`.
c. **Documentation:** **[ML-Agents GitHub Repository](https://github.com/Unity-Technologies/ml-agents)**

Code Overview:

a. **MoveToGoalAgent` Script:** Controls the spaceship agent's behavior, learning through interactions within the environment.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;

public class MoveToGoalAgent : Agent
{
    // Reference to the environment, target, and background sprite renderer
    [SerializeField] private Transform Environment;
    [SerializeField] private Transform Target;
    [SerializeField] private SpriteRenderer backgroundSpriteRenderer;
    // This method is called at the beginning of each episode
    public override void OnEpisodeBegin()
    {
        // Randomly set the initial position of the agent within a specified range
        transform.localPosition = new Vector3(Random.Range(-3.5f, -1.5f),
Random.Range(-3.5f, 3.5f));
        // Randomly set the initial position of the target within a specified range
        Target.localPosition = new Vector3(Random.Range(1.5f, 3.5f),
Random.Range(-3.5f, 3.5f));

        // Randomly set the initial rotation of the environment
        Environment.localRotation = Quaternion.Euler(0, 0, Random.Range(0f, 360f));
        // Reset the rotation of the agent
        transform.rotation = Quaternion.identity;
    }

    // This method is responsible for collecting observations from the environment
    public override void CollectObservations(VectorSensor sensor)
    {
        // Add the position of the agent as an observation
        sensor.AddObservation((Vector2)transform.localPosition);
        // Add the position of the target as an observation
        sensor.AddObservation((Vector2)Target.localPosition);
    }

    // This method is called when the agent receives actions from its policy
    public override void OnActionReceived(ActionBuffers actions)
    {
        // Retrieve continuous actions from the provided actions
        float moveX = actions.ContinuousActions[0];
        float moveY = actions.ContinuousActions[1];

        Set the movement speed
        float movementSpeed = 5f;

        // Move the agent based on the received actions and the movement speed
        transform.position += new Vector3(moveX, moveY) * Time.deltaTime *
movementSpeed;
    }
    // This method is used for manual control of the agent
    public override void Heuristic(in ActionBuffers actionsOut)
    {
        Get continuous actions from user input(arrow keys or joystick)
        ActionSegment<float> continuousActions = actionsOut.ContinuousActions;

        continuousActions[0] = Input.GetAxisRaw("Horizontal");
        continuousActions[1] = Input.GetAxisRaw("Vertical");
    }

    // This method is called when the agent collides with other objects
    private void OnTriggerEnter2D(Collider2D collision)
    {
        // Check if the agent collided with the target
        if (collision.TryGetComponent(out Target target))
        {
            Reward the agent for reaching the target
            AddReward(10f);
            // Change background color to green
            backgroundSpriteRenderer.color = Color.green;
            // End the episode
            EndEpisode();
        }
        // Check if the agent collided with a wall
        else if (collision.TryGetComponent(out Wall wall))
        {
            Penalize the agent for hitting a wall
            AddReward(-2f);
            // Change background color to red
            backgroundSpriteRenderer.color = Color.red;
            // End the episode
            EndEpisode();
        }
    }
}
```

Setup and Execution Instructions:

1. Install and Configure Unity:

- Install Unity through Unity Hub and set up a new project or open an existing one.

2. Python and ML-Agents Setup:

- Install the required version of Python and use pip to install the ML-Agents package.

3. Project Configuration:

- Download your project files and open the project in Unity.
- Verify that the **`MoveToGoalAgent`** script is attached to the spaceship agent.

4. Running the Simulation:

- **Start ML-Agents Training:**

a. Open a terminal.
b. Activate any necessary Python virtual environment.
c. Navigate to your project's folder where the ML-Agents configuration file (`config/ppo.yaml`) is located.
d. Start the ML-Agents training with the command:

```bash
 mlagents-learn config/ppo.yaml --run-id=SpaceshipSimulation
```

- **Start Unity Simulation:**

  In Unity, press the **Play** button to begin the simulation. Unity will connect to the ML-Agents environment running in Python.

5. Monitoring Training:

a. Monitor the training progress using TensorBoard by running:

```bash
 tensorboard --logdir results
```

b. Access TensorBoard in a web browser at `localhost:6006` to view detailed training metrics.

6. Adjustments and Optimization:

- Adjust training parameters in the `ppo.yaml` file based on performance observed in TensorBoard and Unity.

7. Finalization:

- Once satisfactory results are achieved, stop both the Unity simulation and the Python training process.
- Save and document the trained model for further use or deployment.

Additional Considerations

a. **Backup and Documentation:** Regularly save your project and document all changes to maintain a reliable workflow and facilitate future improvements.

b. **Performance Tuning:** Continuously tweak agent parameters and training settings to optimize the agent's performance based on observed behaviors.

This guide provides all necessary steps to effectively set up, execute, and optimize your spaceship simulation project with Unity and ML-Agents, ensuring a smooth workflow from initial setup to final execution.