UNITY ENVIRONMENT REPORT

Autonomous Agents

Neil de la Fuente, Paula Feliu, Roger Garcia and Andrew Ng

March 2024

1 Introduction

This report details the development process of our project aimed at enabling the agent to execute various functions accurately. Our primary objective is to implement three essential classes: "Turn", which facilitates the agent's rotation to specified degrees on either side, "RandomRoam", allowing the agent to select states based on predefined probabilities, and "Avoid", enabling the agent to maneuver through obstacles without collisions.

The Unity environment serves as our testing ground, featuring a bounded space enclosed by four walls. Depending on the chosen environment, different obstacles are presented to the agent. For instance, in the "ForwardStop" environment, a single obstacle is strategically placed to verify the agent's ability to halt upon detection, preventing collisions.

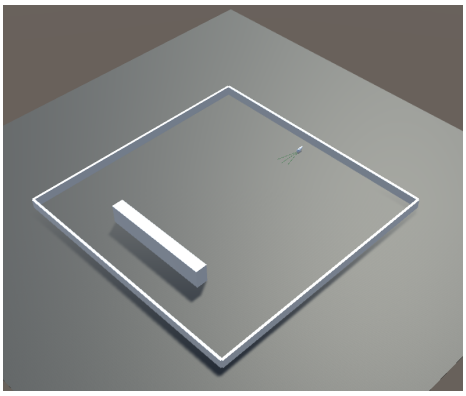

Figure 1: ForwardStop Scene

Conversely, the "Roaming" environment provides an obstacle-free space, ideal for testing the "Turn" and "RandomRoam" functions, as they focus solely on rotational movements and state selection without obstacle interference.

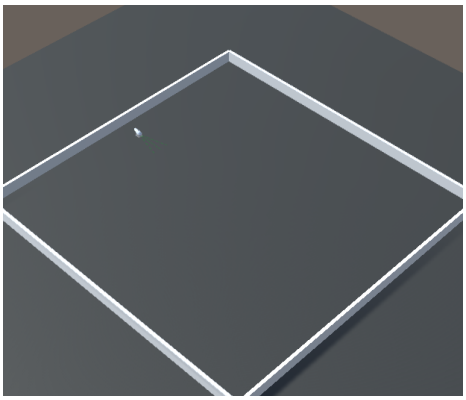

Figure 2: Roaming Scene

Lastly, the "Avoid" environment is tailored explicitly for testing the "Avoid" function. Here, the bounded space is populated with various obstacles, primarily cylindrical in nature, challenging the agent to navigate through them adeptly without any collisions.

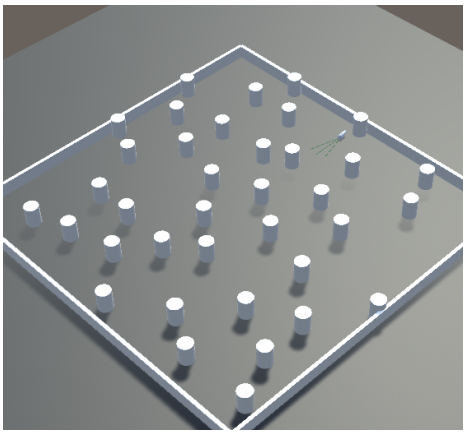

Figure 3: Avoid Scene

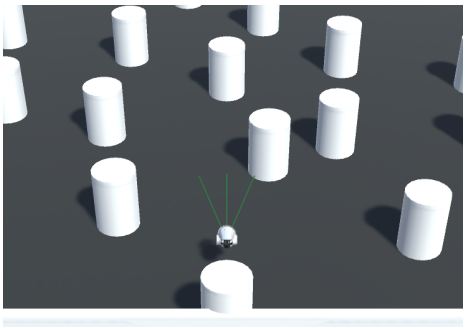

Figure 4: Avoid Scene

This project aims not only to implement these functionalities effectively but also to provide insights into the agent's adaptability and decision-making prowess within dynamic environments.

2 Turn

This class makes it possible for the agent to Turn in the randomly chosen direction a randomly chosen number of degrees. The functionality of turning is crucial for scenarios requiring the agent to change directions to explore its environment well or avoid obstacles efficiently.

2.1 Approach

To implement this feature, we first determine the direction and magnitude of the turn. The direction (left or right) is randomly chosen, with equal probability for each. Subsequently, the number of degrees the agent must rotate is also selected at random, ensuring a range from 1 to 360 degrees to cover all possible orientations within a full circle.

After establishing these parameters, the agent executes the turn in increments of 5 degrees. Each incremental rotation is followed by a brief pause of 0.3 seconds to give enough time to the agent to finish its turn successfully, while allowing us to see that one turn ended before the beginning of another.

To translate the chosen degree of rotation into actionable steps, the total turn-degrees is divided by 5, as each turn step corresponds to a 5-degree rotation. This calculation results in turns-needed, the total number of incremental turns the agent must complete to achieve the desired rotation. The division by 5 is based on the predefined rotation granularity, that according to the teachers will be changed in future versions for smooth turns.

The execution of these incremental turns is managed through the async for - in self.async-turns(turns-needed): loop, which iterates over the number of turns needed. Inside this loop, the agent is instructed to perform each incremental turn by appending the turn-direction to the requested-actions list and sending a corresponding action message to the agent (await self.a-agent.send-message("action", turn-direction)).

After completing the required number of turns, the method pauses for an additional 2 seconds (await asyncio.sleep(2)), providing a brief period for the simulation to stabilize and for any other necessary computational processes to catch up. This pause is crucial for ensuring that the agent's environment is updated correctly before proceeding with further actions, maintaining the integrity of the simulation's state.

3 RandomRoam

The following class, RandomRoam, allows the agent to move randomly in its environment, changing direction, deciding to stop and move again, all while following certain probabilities and maintaining the action for a predefined time.

3.1 Approach

To achieve this, we initialize the possible states of the agent, which are "STOPPED", "MOVING", and "TURNING". We establish that the default initial state is "STOPPED". Additionally, we assign predefined probabilities to each state: 55% for turning, 10% for being stopped, and 35% for moving forward.

Each time the agent enters a new state, it will execute the corresponding function and then randomly select the next state based on the assigned probabilities. For example, if the current state is "STOPPED", the next state is chosen based on the established probabilities.

In the "MOVING" state, the agent moves forward for the specified seconds while checking for any obstacles using its sensors. If it detects an obstacle, it stops moving and automatically changes to the "TURNING" state to rotate, then randomly selects the next state. This is done to avoid collisions with obstacles or walls.

In the "TURNING" state, the agent randomly chooses the direction and the number of degrees it will rotate. Each rotation is done in increments of 5 degrees, with a waiting time of 0.3 seconds between each rotation. Once the agent has rotated the necessary degrees, it randomly selects the next state. We

have observed that depending on the time between each rotation, the number of degrees corresponding varies. Therefore, in this case, we have established 0.3 seconds per rotation to achieve 5 degrees per rotation.

Finally, in the "STOP" state, the agent stops for 2 seconds and then randomly selects the next state.

This implementation ensures that the agent can move autonomously in its environment, making adaptive decisions based on its conditions.

4 Avoid

The challenge at hand revolves around an environment where an agent must constantly navigate while avoiding cylindrical obstacles that obstruct its path. The premise is that the agent, equipped with sensors, must be capable of detecting these obstacles and taking evasive measures to avoid collisions, thereby enabling it to continue exploring the space without incidents.

4.1 Approach

To address this environment, a class named "Avoid" has been developed to manage the behavior of the agent. This implementation is based on defining three main states: "moving", "turning", and "stopped", each with a specific purpose in the obstacle avoidance process.

When the agent is in the "stopped" state, it means it is halted and ready to resume movement, so if no sensor detects any obstacle, it will transition to

the "moving" state to resume motion. In the "moving" state, the agent continuously advances while monitoring the presence of obstacles using its sensors.

If an obstacle is detected during movement, the agent transitions to the "turning" state to take evasive action. During this turning phase, the direction of the turn is determined based on which sensor detected the obstacle. Specifically, if the agent detects an obstacle with its right sensor, a left turn will be assigned to avoid the obstacle more efficiently, and if the left sensor detects it, a right turn will be assigned. In the case where the central sensor detects an obstacle, a direction is randomly chosen between left or right for the turn. The agent performs a predefined number of turns, in this case, three turns, to avoid the obstacle, which has been considered sufficient for successfully resolving the problem.

Once the evasive maneuver is completed and the necessary turns executed, the agent returns to the "moving" state to continue advancing and exploring the environment. If during the turning process another obstacle is detected, the agent repeats the turning process until completing the three turns and can continue moving without collisions. That is, before moving forward, it checks that the sensors no longer detect an obstacle, whether it be a cylinder in the environment or the wall.

This implementation ensures that the agent can autonomously navigate the environment, efficiently avoiding obstacles, and ensuring continuous exploration without interruptions due to collisions.