**Deep reinforcement learning using Unity ML-Agents**

*Build a volleyball environment and train agents to play in it using deep RL.*

By:

Vachana Belgavi 002101078

Akshata Nanjappa 002100876

**Reinforcement learning**

Reinforcement Learning is a powerful technique that has emerged in modern machine learning, it allows a system to learn through a process of trial and error. It has been successfully applied in many domains, that learned to master the games of chess, Go, and Shogi. We have implemented the same technique in our project for a game of volleyball using Q-learning, Deep Q learning, Policy-Based methods, Markov Decision Processes (MDP), Bellman equation, epsilon greedy method, CNN, deep learning, double Q learning, Actro Critic Methods, MLP Policy, Prioritized replay, Replay memory, Random sampling methods, Deep deterministic Policy.

We will be setting up an environment, loading the dependencies, Tuning hyperparameters, exploration, and exploitation, bias and variance tradeoff, policy-based vs value-based methods.

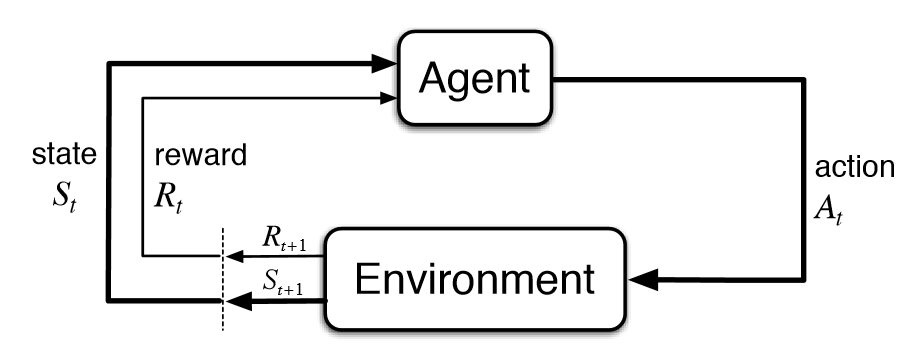
Reinforcement Learning (RL) is a machine learning technique that enables an agent to learn in an interactive environment using trial and error feedback from its own actions and experiences.

Diagram

Description automatically generated

Though both supervised and reinforcement learning use mapping between input and output, unlike supervised learning where the feedback provided to the agent is the correct set of actions for performing a task, reinforcement learning uses rewards and punishments as signals for positive and negative behavior.

As compared to unsupervised learning, reinforcement learning is different in terms of goals. While the goal in unsupervised learning is to find similarities and differences between data points, in the case of reinforcement learning the goal is to find a suitable action model that would maximize the total cumulative reward of the agent. The figure below illustrates the action-reward feedback loop of a generic RL model.



Some key terms that describe the basic elements of an RL problem are:

Environment — Physical world in which the agent operates

State — Current situation of the agent

Reward — Feedback from the environment

Policy — Method to map agent’s state to actions

Value — Future reward that an agent would receive by taking an action in a particular state

In order to build an optimal policy, the agent faces the dilemma of exploring new states while maximizing its overall reward at the same time. This is called Exploration vs Exploitation trade-off. To balance both, the best overall strategy may involve short-term sacrifices. Therefore, the agent should collect enough information to make the best overall decision in the future.

**Unity ML-Agents**

Unity ML-Agents is a powerful tool that makes it easy to train and experiment with reinforcement learning algorithms. However, what makes ML-Agents a powerful toolkit is the ability to leverage Unity's engine and create complex, physics- and graphics-rich 3D environments.

Unity ML-Agents is an open-source toolkit developed by Unity to enhance a game’s AI with machine learning. Typically, when developing an AI for a game, you’d check to see if a certain condition is true (i.e., can you see the player?) and then execute a certain action (i.e., attack). This form of AI works, but at the core of things, it can be predictable and limiting.

Machine learning allows agents (enemy, AI car, anything you want to have an AI) to automatically learn through reinforcement learning, imitation learning, and many other learning types. What this means, is that you’re not specifically telling the agent what to do. Instead, you’re developing their brain overtime for them to determine how to go about a certain task with a few given inputs.

**ML-Agents Package**

Right now, ML-Agents is still in development but it can be downloaded from the Unity GitHub page [here](https://github.com/Unity-Technologies/ml-agents). Furthermore, since it’s still in development and since the training is done through Python, there are additional things you need to download and set up, but the provided tutorials will go through that. ML-Agents also includes 15+ example environments, showing many different game types and how those are trained.

**Volleyball Game**

In Unity, scripts can be assigned to various game objects to control their behavior. Below is a brief overview of the four scripts used to define the environment.

*VolleyballAgent.cs*

This class is Attached to the agents.

This controls both the collection of observations and actions for the agents.

public override void CollectObservations(VectorSensor sensor)

{

// Agent rotation (1 float)

sensor.AddObservation(this.transform.rotation.y);

// Vector from agent to ball (direction to ball) (3 floats)

Vector3 toBall = new Vector3((ballRb.transform.position.x - this.transform.position.x)\*agentRot,

(ballRb.transform.position.y - this.transform.position.y),

(ballRb.transform.position.z - this.transform.position.z)\*agentRot);

sensor.AddObservation(toBall.normalized);

// Distance from the ball (1 float)

sensor.AddObservation(toBall.magnitude);

// Agent velocity (3 floats)

sensor.AddObservation(agentRb.velocity);

// Ball velocity (3 floats)

sensor.AddObservation(ballRb.velocity.y);

sensor.AddObservation(ballRb.velocity.z\*agentRot);

sensor.AddObservation(ballRb.velocity.x\*agentRot);

}

*VolleyballController.cs*

This class is Attached to the ball.

This script checks whether the ball has hit the floor. If so, it triggers the allocation of rewards in *VolleyballEnv.cs*

void OnTriggerEnter(Collider other)

{

if (other.gameObject.CompareTag("purpleGoal"))

{

envController.ResolveGoalEvent(GoalEvent.HitPurpleGoal);

}

else if (other.gameObject.CompareTag("blueGoal"))

{

envController.ResolveGoalEvent(GoalEvent.HitBlueGoal);

}

}

*VolleyballEnvController.cs*

This is Attached to the parent volleyball area which contains the agents, ball, etc.

This script contains all the logic for managing the starting or stopping of the episode, how the objects are spawned and how rewards should be allocated.

public void ResolveGoalEvent(GoalEvent goalEvent)

{

if (goalEvent == GoalEvent.HitPurpleGoal)

{

purpleAgent.AddReward(1f);

blueAgent.AddReward(-1f);

StartCoroutine(GoalScoredSwapGroundMaterial(volleyballSettings.purpleGoalMaterial, purpleGoalRenderer, .5f));

}

else if (goalEvent == GoalEvent.HitBlueGoal)

{

blueAgent.AddReward(1f);

purpleAgent.AddReward(-1f);

StartCoroutine(GoalScoredSwapGroundMaterial(volleyballSettings.blueGoalMaterial, blueGoalRenderer, .5f));

}

blueAgent.EndEpisode();

purpleAgent.EndEpisode();

ResetScene();

}

*VolleyballSettings.cs*

This is Attached to a separate ‘VolleyballSettings’ object.

This holds constants for essential environment settings, for example, agent run speed and jump height.

public class VolleyballSettings : MonoBehaviour

{

public float agentRunSpeed = 1.5f;

public float agentJumpHeight = 2.75f;

public float agentJumpVelocity = 777;

public float agentJumpVelocityMaxChange = 10;

}

**Letting our players make decisions:**

We want our agents to learn which actions to take given a certain state of the environment. If the ball is on our side of the court, our agent should get it before it hits the floor. While training, the agent will either take actions:

1. At random (to explore which actions lead to rewards and which don't)
2. From its current policy (the optimal action given the current state)

ML-Agents provides a convenient **Decision Requester** component that will handle the alternation between these for us during training.

To add a Decision Requester:

1. Select the **PurpleAgent** game object (within the **PurplePlayArea** parent).
2. Add Component > Decision Requester.
3. Leave decision period as 5.

![Graphical user interface, application

Description automatically generated](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RDoRXhpZgAATU0AKgAAAAgABAE7AAIAAAAKAAAISodpAAQAAAABAAAIVJydAAEAAAAUAAAQzOocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEpveSBaaGFuZwAABZADAAIAAAAUAAAQopAEAAIAAAAUAAAQtpKRAAIAAAADMzkAAJKSAAIAAAADMzkAAOocAAcAAAgMAAAIlgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADIwMjE6MDk6MDcgMjE6MDE6NTAAMjAyMTowOTowNyAyMTowMTo1MAAAAEoAbwB5ACAAWgBoAGEAbgBnAAAA/+ELHGh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iLz48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMjEtMDktMDdUMjE6MDE6NTAuMzk0PC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyI+PGRjOmNyZWF0b3I+PHJkZjpTZXEgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpPkpveSBaaGFuZzwvcmRmOmxpPjwvcmRmOlNlcT4NCgkJCTwvZGM6Y3JlYXRvcj48L3JkZjpEZXNjcmlwdGlvbj48L3JkZjpSREY+PC94OnhtcG1ldGE+DQogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgIDw/eHBhY2tldCBlbmQ9J3cnPz7/2wBDAAcFBQYFBAcGBQYIBwcIChELCgkJChUPEAwRGBUaGRgVGBcbHichGx0lHRcYIi4iJSgpKywrGiAvMy8qMicqKyr/2wBDAQcICAoJChQLCxQqHBgcKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKir/wAARCABbAh4DASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwDy+zBa1gA6lFHJ9q0W0m9VSzQ4AGSd68frWba/8ecP/XNf5VrS6o0mlpb87+jt6qOn+fagB3hyzh1HxTpVldKWgub2GGRQcZVnAIz24Nezar4Z+F2j6ktjeWMYnO3eBfv+7y6LhsyjHD7v91WP18f8IMF8caGzEBRqNuSSen7xa991Xwrqmo+MYr4X8y2qPGVbbDmMKJWIAKc4YxgZz1Oc4GIk2loZzckvdPE/iH4Vg8I+KPsdlP51rcQi5gzyVRmYbSe+Cp59MVy1elfHVgfHVoAQSunIDg9P3klea1ZoFXdI0qfWtSSytXjSRlZt0rbVAVSxJP0BqlW34RvbSw8RJNqE4t4DDNGZCjNtLRso4UE9SKAK2p6FdaXbxXDSW9zazMUS4tZRIhYdVJ7HnoapSWtxDKscsEqSPjarIQWz0wK6yy1fRtBh02yjuTqkcepJe3MqwsiqFXaFUNgk8k9B0ArSfxTp0eoWgkv4ZVBuCt1DHO7WzSJtV8ysW64JVRxjIJNAHDx6dM32oTf6O9tF5jJKrBjyBgcdee+BSQadcS3dtBIjW/2lgqSSowXk4zwCSPoDXTnWbWC3voLnXH1KR9OEEcxgbG7zlbYCwywABOWx6Vp3niXSZtV0u8m1PzpY9SS4lMImEQjA5Yxvna3ThCRjPtQBwLWk4SSRYneKJtrSqh2g/XHH40j2lzHF5r28qx8fOUIHIyOfpXc2niXTI4LSY6i0UFrb3EU+mCN8XTOXw3Tbg7hndgjbVe68UWs6X1s1/I9pJokVvFEQ+zz18vjGOCMNz096AMLTfDF1qenJeJd2UEcs5gjW4m2F3ABwOMfxDvWfLp15DcXED20nmWrFZgqk+WQcHJH0rqPD3iTTNL0PTra+t4LopqTzTJJEWaKMqgDqemcg8c9K0dL8UWFvFIr6nCLiLUZbl7mWOcC6DYwwWMrk8EbXwMH60AcFFa3E6loIJZFBCkohIBPQcUgtpyyAQyEuCygIfmA6kevQ/lXoXhPWLW5utCtrfUWsZIL2XzbNI2AufMfKkY+XgcfMeAOM1U0/WNGjOmXN3qPkT2FvdWz25gdixfzNrBgMY+f659uaAOINvMtuJ2hkELHAkKnaT6ZqaHTL2e6t7eO1l825YLCGUjfn0J7e9dnqHifTptBZLW4gQSWcNv8AZGjmd0ZducKWEQwVyGHJ9OTS6x4m0+W40+8TUmnuY9QSeQW3nLH5YHzEpJ91+2FJGM0AcXLpl7Dfy2T2spuYSQ8aqWIx9O3vUK207xGRIJGjXOXCEgY68/iPzrurXxBpsV/riHUoy13dpcQ3kn2hVaMZOwlMOCuRgdOPoazvEHiWO90WW3srsq8+ozzTRwo0SyIyoASCTwSCcZNAHJUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAdDovhGbVNLbUrm4a1tN/lxlIGleRgMnCjsMHnPaqWu6DPoc8AeRZ7e5iE1vOgIEiHvg8g+1dT4X8Y6fb+EpdD1XEWCfLkdHZHVmyVbZ83c/nWZ488UW3iPULZNOjZbS0QrGzLtLE4ycdhwMUAczbwPc3CwxmNWboZJFjX8WYgD8TWza+FLm50/ULj7VaK1ksTbRdRMjh2K/6wNtUjb0JyePUVjW7QLcKbqOSWH+JI5AjH6MQcfka6DUNS0E2dnDa212SbKJLsW1wIEeVc53KY23HPOc0Ac3VrTLW3vdQjgvLxbKFs7p2QuFwCRwPXpVWtTw3eLYa/b3D3zWCoGzcLCJSmVI+6euc4/GgDLPX1qvc9V/GrB5Y9+ar3PVfxoAbbO4tYsbfuDt7VJ5j+q/lUNt/x6xf7g/lUtAC+Y/qv5UeY/qv5VJBA07NhlVUXczMcBRnH8yKJLd0mEa4lLDKmM53CgCPzH9V/KjzH9V/KlEUjEgIx29eDxTmt5Fd1VS4QkFlU4460AM8x/Vfyo8x/Vfyp728scMcroQkn3W9ajoAXzH9V/KjzH9V/KkooAXzH9V/KjzH9V/KkooAXzH9V/KjzH9V/KkooAXzH9V/KjzH9V/KkooAek8sbh432spyGXII/Wm+Y56lfypKKAF8x/Vfyo8x/VfypKKAF8x/Vfyo8x/VfypKKAF8x/Vfyo8x/VfypKKAF8x/Vfyo8x/VfypKKAF8x/Vfyo8x/VfypKKAF8x/Vfyo8x/VfyqOU4jOPb+daGmeGbrVbI3cD28cQfZmVsc8e3vQBT8x/Vfyo8x/VfyqS70S5stSisplj8yYr5bKcq2TgHPpUM2nmKEyo8M0anaxjOdp7ZBANADvMf1X8qPMf1X8qrNCUVWeMqGGVJXGfpThat5hSRPLO0t86kZAGfT2oAn8x/Vfyo8x/VfyqCO1aRowU2LI21XZTj9BTRAxjLiMlAcFgvA/GgCz5j+q/lR5j+q/lVYwMqb2iIX+8V4q1c6U9sJfnhkMJxKqHlOcdwO/HFACeY/qv5UeY/qv5VWaBlCloiob7pK9fpStbOhw8LKeeCmO2f5UAWPMf1X8qPMf1X8qrLAzMoWIkt90Bev0oW3dmZViZmXqAvIoAs+Y/qv5UeY/qv5VC9nKkMUrQkJL9w4+9Tfs0nmGPyW3jquzkfhQBY8x/Vfyo8x/VfyqpsH90flSbV/uj8qALnmP6r+VHmP6r+VU9q/3R+VG1f7o/KgC55j+q/lR5j+q/lVPav90flRtX+6PyoAueY/qv5UeY/qv5VT2r/dH5UbV/uj8qALnmP6r+VHmP6r+VU9q/3R+VG1f7o/KgC55j+q/lR5j+q/lVPav90flRtX+6PyoAueY/qv5UeY/qv5VT2r/dH5UbV/uj8qALnmP6r+VHmP6r+VU9q/3R+VG1f7o/KgC55j+q/lR5j+q/lVPav90flRtX+6PyoAueY/qv5UeY/qv5VT2r/dH5UbV/uj8qALnmP6r+VHmP6r+VVVRc/dH5VeSwgEatcybGdd4RIt5C+p5GBQBH5j+q/lUcrFiu7HfoKLq2+ysQNpG3cjr0YdjSSdRQAlt/x6xf7g/lUtRW3/HrF/uD+VS0AWLJ9krHzxASuAWXcrezDB4/A1bM1sJJUilSJpIQplRWCbt2TgYyARx0/SsyigC891ttp4xctI7yR/NgjeoVgfw5HWrr3CMBMl35MZvpZAcN84+U9h1+vrWJRk4xnj0oAuXlxHPbReW+3azfujngEkj29qp0UUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAEc3+qP1H866fQ5rZ/DBt5Ly3hlFwzbZXAJ+UYPX1rmyARg8imeSnof8Avo0PUTV1Y6TxBeWY8QaXJBcxyw26R73jO4DDknp7Vi77e0t5lWdLl5ioIRWChQwY53Ac8AfnVXyU9D/30aPJT0P/AH0aBmzNqlt9pWTdE8bXKy7VWQsoGeTuOARnoKprOkMn72++0jypgCAxClkIHJGck1S8lPQ/99GjyU9D/wB9GgDXgvrZWtZJ7gbo5IciPeOF67lPHA9Khgu4Vhgc3G1YoXje3wfnJ3e2MHI6+n0rO8lPQ/8AfRo8lPQ/99GgC9NfJItyhmZka0ijRTnG5fLyMfg3P+NS3upW9w175axpvmDqyoR5ybvun07Ht0+lZnkp6H/vo0eSnof++jQBtSapbfafMEkRR5xINqyF4+DhvmOARkcD0piyxxWVsJrsSqXuIzKAxC7o1A6jOMnPSsjyU9D/AN9Gl8pduPmxnONxoA1Iru1hRIBNG5FqYvMYOFDeaWxxhsEHr/8AXpUv4mupfMnhWMmPLKJFztGMqwycj369ayfJT0P/AH0aPJT0P/fRoA1Le+t0+zkzEiMSpiQNld2drHH17HNNm1BFjmWOZA/2YRo0O/k+YGxluemfT0rN8lPQ/wDfRo8lPQ/99GgCKipfJT0P/fRo8lPQ/wDfRoAioqXyU9D/AN9GjyU9D/30aAIqKl8lPQ/99GjyU9D/AN9GgCKipfJT0P8A30aPJT0P/fRoAioqXyU9D/30aPJT0P8A30aAIqKl8lPQ/wDfRo8lPQ/99GgCKipfJT0P/fRo8lPQ/wDfRoAioqXyU9D/AN9GjyU9D/30aAIqKfIiRxlsE47bjVXzn/591/7/ADf4UAT0VB5z/wDPBP8Av83+FHnP/wA8E/7/ADf4UAWE61sW98ApeO6W2kaEQyh4ywZQMAjAPOAK5/zn/wCeC/8Af5v8KXz5P+eC/wDf9v8ACgDQvpklCpDu8uKPYpYct1JP5k1FJ1FVBcMGBeBdueSJScfgRVuTqKAEtv8Aj1i/3B/Kpaitv+PWL/cH8qloA1tDKRw6pcNDDK8FnvjE0YdVbzo1zg8dGP51ei0qHWlsLgJHZGZJzOI8KrCIA7lDEBc529QMjNYljqNxp0kjWpj/AHsflyLJEkisuQcFWBHVQfwqVtZv3vo7sz/vYl2IAihFXnKhANu05ORjByaANGbQrKAyXDXRa1itxK8cUkckisX2BMqSvfOfTtmo4LCyntJTHd+Xam+t4jNLCN0aushJJ68Y6A4OM+lVP7cvhc+cGhH7ryjGIEEZTOduzG3GeenXnrTTrN+0rSefhmljmOEUAMgITAxgAAkYHGO1AGpLoiww3ggWQA2quhlEcisTOifJIpIx83UYPanQ+HLK61I2dvdT7oL2O1uGdAAwZipZfTBHQ+v4VlT61fXETxNJGkTx+WY4oUjXbvD8BQAPmUHPXipH8Ram8kUnnqskcqz71hQF5F6M5A+cj/az1PqaAG31lapptte2MkzRyyyQssygEMgQ5GOxDjjtjvW5ewWsFmFuo9OjtjpkTx7NguDM0KkHC/Ny5yd3GM+1cu1zK1mlqX/cxyNIq4HDMACc9eij8qLi5mu5Fe4feyxpGDgDCqoVRx6AAUAbl34cgtdPZnu0FzHDHKQ00WHLbTsC7t+QG6kc4PTip5fD9s9zdadYzv8AutSjtHlmiXOf3uWGOcfJ0zz+WMWTWb2Wz+zPJGU2qhbyU3lVxtUvjcQMDjPYelIdXvzNLL9oYSTXIunZQATKN2GGBx99unHNAGjBoljewrd2s9wlpGZfPEiAuAib8rg4ORx7e9QaxBZxaZpktgrhZo5CxkUBiQ5GCR1x61C2vag08UqyxxmEsUWOBETLDDZUAA5HByORx0qC91K61ARC6dCsKlY0SNUVATkgBQB1oAq0UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAOiVHkAkfy1PVsZx+FWfs9n/z/f8AkE1UooAt/Z7P/n+/8gmj7PZ/8/3/AJBNVKKALf2ez/5/v/IJo+z2f/P9/wCQTVSigC39ns/+f7/yCa9M8NWNna6Dbm0CP5sYZ5NvLk9c/wAsV5RWhY67qemQmKyu3jjJztwGA+mRxQBreLdO0+215hFOtvvQO0axkgE59OnrWJ9ns/8An+/8gmq800lxM0s7tJI5yzMckmmUAW/s9n/z/f8AkE0fZ7P/AJ/v/IJqpRQBb+z2f/P9/wCQTR9ns/8An+/8gmqlFAFv7PZ/8/3/AJBNH2ez/wCf7/yCaqUUAW/s9n/z/f8AkE0fZ7P/AJ/v/IJqpRQBb+z2f/P9/wCQTR9ns/8An+/8gmqlFAFv7PZ/8/3/AJBNH2ez/wCf7/yCaqUUAW/s9n/z/f8AkE0fZ7P/AJ/v/IJqpRQBb+z2f/P9/wCQTR9ns/8An+/8gmqlFAFv7PZ/8/3/AJBNH2ez/wCf7/yCaqUUAW/s9n/z/f8AkE0fZ7P/AJ/v/IJqpRQBb+z2f/P9/wCQTR9ns/8An+/8gmqlFAEd4AI3CtuUMMNjGRmt7TVtLzQoLCxe0XUXhmZ1nsQ7SEFiAJSPlO0cY79xWDMhkhKr14I/OrEGs6tbaf8AY4WURBGRWMcZeNWzuVXI3KDk5AI6n1NAGXRTvJl/55n8x/jR5Mv/ADzP5j/GgBtFO8mX/nmfzH+NHky/88z+Y/xoAjb7jfSr0nUVV8iVuNmM8ZJHFWpOooAS2/49Yv8AcH8qlqK2/wCPWL/cH8qloAt6dYf2ldLbrcwQOxCp5xYBiTgAYBpbnTpIL0WsEsd5LzkWwZsEE5HIBzxTdLmjt9Xs5pm2xxzo7tjOAGBJrR0y6tF1LUJJrkRearCIu0ixvlwcP5Y3YwM49QKAMwWM5s5bnbhYpkhZT97cwYjj/gBpi207eZtgkPlf6zCH5Pr6V1zeItLjvL5w3mC4mtiksaOGi2wOjypu/iVmH3jk5yDnkQaRq1nZWcUIurVZLW7eUyyi4xMDtwyhCNx+UjD46+5oA5m0tZb68htYADJM4RcnAyT39qfcafcW+oyWJjLzxuUKxjduI9MdR3q5o99aWVzd3lym5xCyQQrlcs/ynnnGFLfjitM6rYXSyPDP/Z9xc2S27O29vLZHXGWAzhkUDIB5HPBoAwYtMvJrae4S3cxW7BZmx9wn1/KmyWU6T3EcaNMLdiHeNGKjBxnkZA474rW+3xm21S3l1QzPIIWSZ1cCbYMFehPsC2Mgc4rSfWbKbU2ng1P7HHFqs103yPm4jZgVIAHJwCMNjr7mgDkxbTmEzCGQxAZL7DtHOOv1BpGgmSFZnidYnOFcqQrfQ10Q1y3MtmiztFarp91DJCA21XczFVx3+9Hz9PTiPUr+1udBVXvFe5CQoiQ+auQq4xIrfJwOAVPPXHJoApQaBPcWttLHcWwkukZ4bdnIkcBmXjjGcqcDNZwgmaBplicxKcM4U7Qfc11emavZw2Gkia+s0jt4njuYWtWacgyuSEkCcEqwxhxgmo7bWrSPQ4Ejkto3htJYHhlE5Z2Yt0CkRnO4ctyMd8CgDn7rTLyyVGuraSNZEWRSV42npUDQypu3xuuwgNlSNpPTNdNb6rp8l3aDULovZzWSwXKbWLRsmGHGMHLIOQTwTnFPk8S2ct1pt7PmSeS6S41BQhwDGNiY9eNzfU0AcvJbTxSrHLDIkjY2oyEE56YFL9kufOaH7PL5q4ymw7h+H4j866xdds4Ly1WWe0mVfPIliFy3lM6bQSznf15IXp15NVrvW4Vt7qKG5h8z+zxBG9qJsE+erld0h3H5Q3JwO31AOftbKa61KGxRdk80qwqH4wxOOfTk1PdaRJb2rXMVxb3cKMEkeBydhPTIIBGcHnGKvjVLZvGGl6hJMTFCbNp5SpJyioHJ4yTlT9femCWx0yyu0jvI9QluyilYUdUVFcOcl1U5JUDgdCeaAMeSCaJEeWJ0WQZQspAYe3rUq2M3nGKZTAwjaQCVGGQFJ9O+PpXWXfiKw+3pcCS3mgkv0uNiJM0kajPJDnYCM4wuQcdgBWXHeRWtx/pOs/b1NvdqCEkKo0kRVeWAOWOM8YHHPXABjwWM00kAKtFHO4RJXRtuc+oBz+GajEEzQtMsTtEpw0gU7QfrXWWmsWCNps95frvgmtSRCJR8keAfMQgr8oHBTr6cmqtnqlrHaWcjX3lx21pLDLY7XzKzb+mBtIbcMkkEY9hQBzrW06QiV4ZFjJADlCByM9fpzSSQTQqhmieMOMoWUjcPUetdBc6zDN9uiN07QyabbQxIQxXzE8nIA7Y2vz9fWmeI7+1voFkS8We5edpHEBlEZBA+Yq/3WJ7KSP0oA5+iiigCK4ZlgYocHgA+nNRWVrqGo3sVpYtPPcTMEjjRFJYnt0qW5/492+o/nW2vjRLbQlt9M021sdUeEW0+pQfK7wgYAA6Kx6Mw5IA96AMbyLizvbm0vHDS27lHwysAQSDgrwRx1FPqracs5HTA5/OrVABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFRydRUlRydRQAlt/wAesX+4P5VLUVt/x6xf7g/lUtABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFRydRUlRydRQB//9k=)

Explaining the VolleyballAgent.cs. This script contains all the logic that defines the agents' actions and observations:

* Start() — called when the environment is first rendered. Grabs the parent Volleyball environment and saves it to a variable envController for easy reference to its methods later.
* Initialize() — called when the **agent** is first initialized. Grabs some useful constants and objects. Also sets agentRot to ensure symmetry so that the same policy can be shared between both agents.
* MoveTowards(), CheckIfGrounded() & Jump() — from ML-Agents sample projects. Used for jumping.
* OnCollisionEnter() — called when the Agent collides with something. Used to update lastHitter to decide which agent gets penalized if the ball is hit out of bounds or rewarded if hit over the net.

Adding an agent in Unity ML-Agents usually involves extending the base Agent class, and implementing the following methods:

* OnActionReceived()
* Heuristic()
* CollectObservations()
* OnEpisodeBegin() (**Note:** usually used for resetting starting conditions. We don't implement it here, because the reset logic is already defined at the environment level in VolleyballEnvController. This makes more sense for us since we also need to reset the ball in addition to the agents.)

At a high level, the Decision Requester will select an action for our agent to take and trigger OnActionReceived(). This in turn calls MoveAgent().

### MoveAgent()

### This method resolves the selected action.

Within the MoveAgent() method, start by declaring vector variables for our agents direction and rotation movements:

Based on the previous assignment order, this is how we'll map our actions to behaviors:

1. dirToGoForwardAction: Do nothing=0 | Move forward=1 | Move backward=2
2. rotateDirAction: Do nothing=0 | Rotate clockwise=1 | Rotate anti-clockwise=2
3. dirToGoSideAction: Do nothing=0 | Move left=1 | Move right=2
4. jumpAction: Don't jump=0 | Jump=1

### Heuristic()

To test that we've resolved the actions properly let’s implement the Heuristic() method. This will map actions to keyboard input so that we can playtest as a human controller.

In the Behavior Parameters component of the PurpleAgent:

1. Set Behavior Type to Heuristic Only. This will call the Heuristic() method.
2. Set up the Actions:
   1. Discrete Branches = 4
      1. Branch 0 Size = 3 [No movement, move forward, move backward]
      2. Branch 1 Size = 3 [No movement, move left, move right]
      3. Branch 2 Size = 3 [No rotation, rotate clockwise, rotate anti-clockwise]
      4. Branch 4 Size = 2 [No Jump, jump]

Graphical user interface

Description automatically generated

Press ▶️ in the editor and you'll be able to use the arrow keys (or WASD) and space bar to control your agent!

Observations:

In ML-Agents, there are 3 types of observations we can use:

* **Vectors** — "direct" information about our environment (e.g., a list of floats containing the position, scale, velocity, etc. of objects)
* **Raycasts** — "beams" that shoot out from the agent and detect nearby objects
* **Visual/camera input**

Here is what we have used:

**The goal is to include only the observations that are relevant for making an informed decision about how to act.**

With some trial and error, here's what I decided to use for observations:

* Agent's y-rotation [1 float]
* Agent's x, y, z-velocity [3 floats]
* Agent's x, y, z-normalized vector to the ball (i.e., direction to the ball) [3 floats]
* Ball's x, y, z-velocity [3 floats]

**Now we'll finish setting up the Behavior Parameters:**

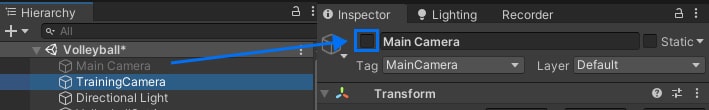
1. Set **Behavior Name** to 'Volleyball'. Later, this is how our trainer will know which agent to train.
2. Set Vector Observation:
   1. Space Size: 11
   2. Stacked Vectors: 1

Graphical user interface, application

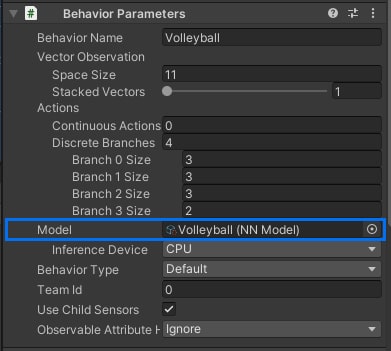
Description automatically generated

**Training:**

1. Make sure that Behavior Types are set to Default:
   1. Open Assets > Prefabs > VolleyballArea.prefab
   2. Select the PurpleAgent object
   3. Go to Inspector window > Behavior Parameters > Behavior Type > Set to Default
   4. Repeat for Blue Agent
2. (Optional) Set up a training camera so that you can view the whole scene while training.
   * **If using the pre-built repo**, select the Main Camera and turn it off in the Inspector.
   * **If using your own project,** create a camera object: right-click in Hierarchy > Camera.



1. Activate the virtual environment containing your installation of ml-agents.
2. Navigate to your working directory, and run in the terminal: mlagents-learn <path to config file> --run-id=VB\_1 --time-scale=1
3. When you see the message "Start training by pressing the Play button in the Unity Editor", click ▶ within the Unity GUI.
4. In another terminal window, run tensorboard --logdir results from your working directory to observe the training process.
5. You can pause training at any time by clicking the ▶ button in Unity. To see how the agents are performing:
   * Locate the results in results/VB\_1/Volleyball.onnx
   * Copy this .onnx model into the Unity project
   * Drag the model into the Model field of the Behavior Parameters component.
   * Click ▶ to watch the agents use this model for inference.



1. To resume training, add the --resume flag (e.g., mlagents-learn config/Volleyball.yaml --run-id=VB\_1 --time-scale=1 --resume)
2. Leave the agents to train. At about ~5M, you'll start to see the agents occasionally touching the ball. At ~10M the agents can start to volleyball.
3. At ~20M steps, the agents should be able to successfully volley the ball back-and-forth!

These agents are trained to keep the ball in the play.

**References**

* Unity3D – Hitting a Volleyball <https://www.youtube.com/watch?v=bsLFIPoBPEQ>
* Reinforcement Learning Project Ideas <https://towardsdatascience.com/8-reinforcement-learning-project-ideas-3521e0ccd313>
* A hands-on introduction to deep reinforcement learning using Unity ML-Agents <https://www.gocoder.one/blog/hands-on-introduction-to-deep-reinforcement-learning>
* Introduction to Reinforcement Learning with David Silver <https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver>
* Reinforcement Learning <https://deeplizard.com/learn/video/nyjbcRQ-uQ8>
* Unity Machine Learning Agents <https://unity.com/products/machine-learning-agents>
* ML-Agents GitHub <https://github.com/Unity-Technologies/ml-agents>
* Ultimate Volleyball: A multi-agent reinforcement learning environment built using Unity ML-Agents <https://towardsdatascience.com/ultimate-volleyball-a-3d-volleyball-environment-built-using-unity-ml-agents-c9d3213f3064>
* Coder One by Joy Zhang <https://www.gocoder.one/blog/introduction-to-unity-ml-agents>