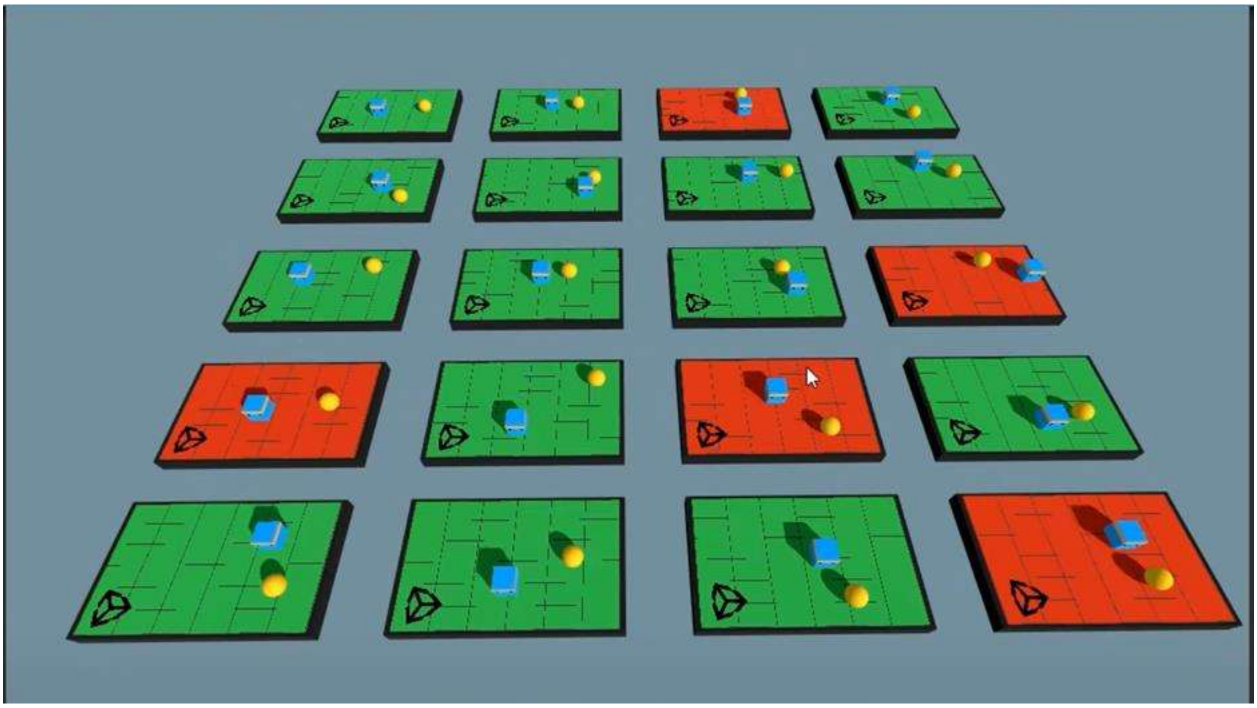
# 1. Introduction to ML-Agents

ML-Agents (Machine Learning Agents) is a Unity toolkit that allows developers to train intelligent agents using reinforcement learning, imitation learning, and heuristic methods.

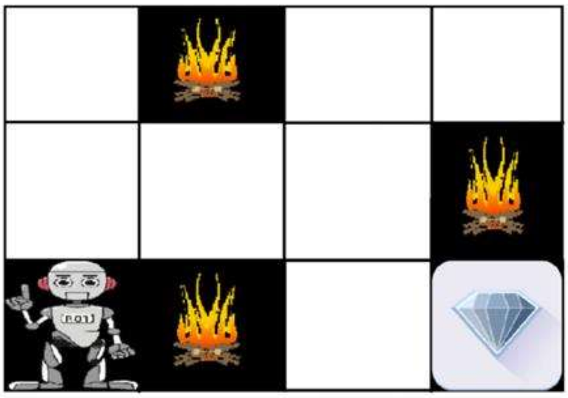


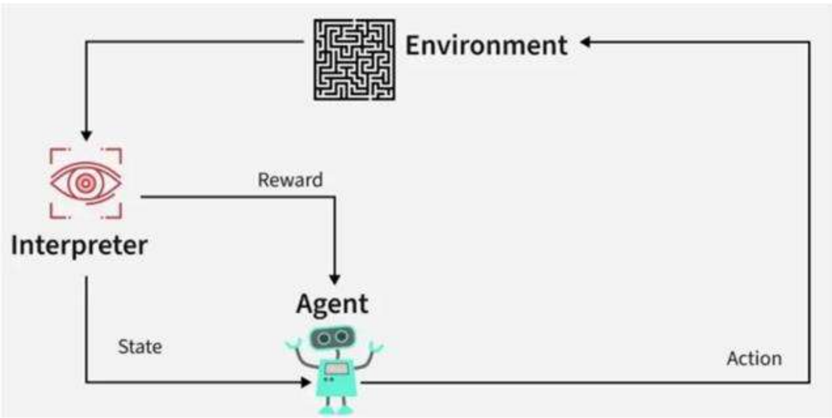
# ML-Agents Approaches or Learning Methods

ML-Agents is a Unity toolkit that allows developers to train intelligent agents using different learning approaches. The main methods include:

- **Reinforcement Learning (RL) -** Learning through trial and error.
- **Imitation Learning (IL) -** Learning by mimicking an expert.

- **Heuristic Methods -** Using manually coded behaviors.




# 1. Reinforcement Learning (RL)

## How It Works:

- The agent interacts with the environment and receives **rewards** for good actions and **penalties** for bad ones.
- Over time, the agent improves by maximizing the rewards.
- Uses **Proximal Policy Optimization (PPO)** to train.

### Example:

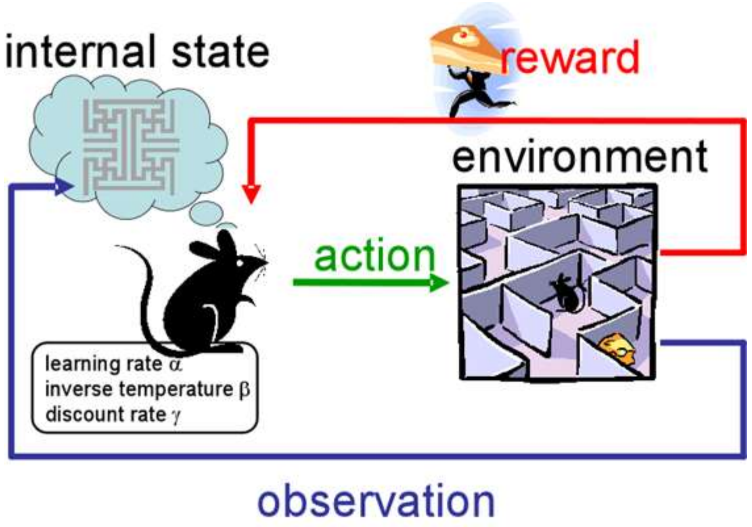
A self-driving car learns to stay on the road by receiving positive rewards for staying on track and negative rewards for going off-road.

### ML-Agents Setup:

- **Behavior Type:** Default (Agent trains when learning is active)
- **Training Command:**

```
mlagents-learn config.yaml --run-id=MyTraining --no-graphics
```

- **Saving & Using the Model:**
    - After training, the model is saved and can be used for inference.




## 2. Imitation Learning (IL)

### How It Works:

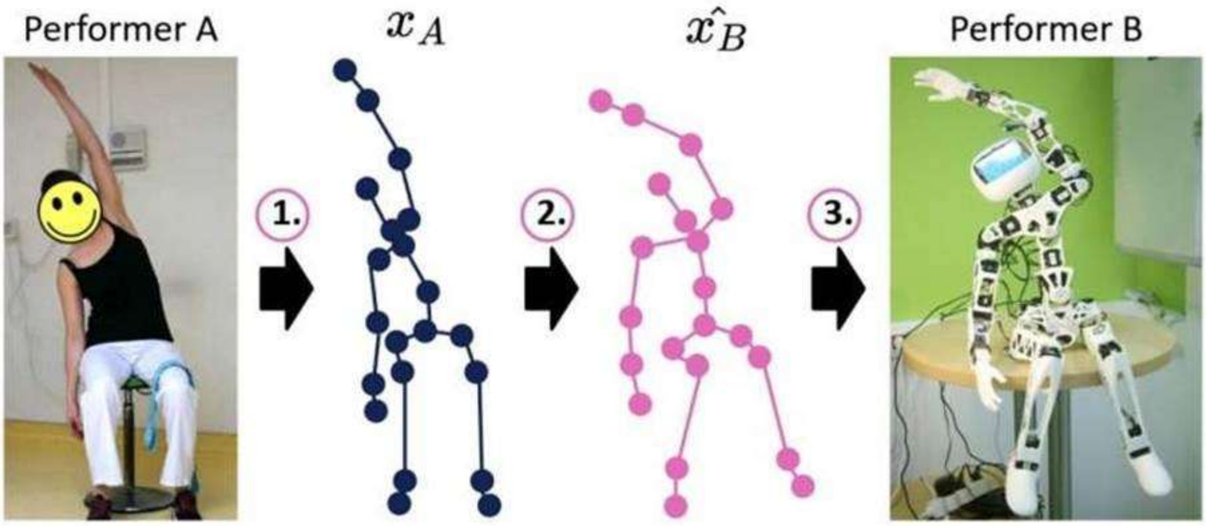
- The agent learns from a pre-recorded expert's actions instead of trial and error.
- Uses **Behavioral Cloning (BC)** or **Generative Adversarial Imitation Learning (GAIL)**.

### Example:

A robot arm learns to pick up objects by imitating a human's recorded actions.

### ML-Agents Setup:

- **Behavior Type:** Player (During recording) → Default (During training)
- **Steps:**
    - **Record Expert Actions** using the heuristic method.
    - **Train the agent** using the recorded data.




## 3. Heuristic Methods

### How It Works:

- The agent does **not learn** but follows **predefined rules** in the script.
- A **Heuristic () function** is manually written to control the agent's behavior.

### Example:

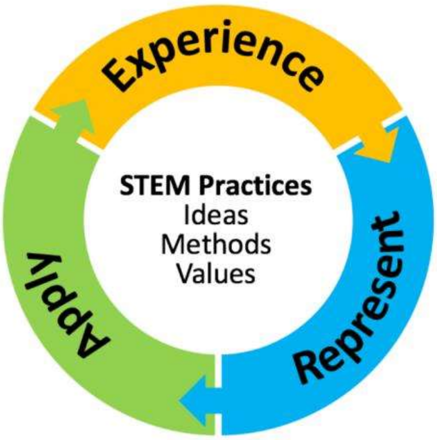
An NPC enemy follows a fixed patrol path without adapting to the player's movements.

### ML-Agents Setup:

- **Behavior Type:** Heuristic Only
- **Example Script:**

```
public override void Heuristic(in ActionBuffers actionsOut)
{
    var continuousActions = actionsOut.ContinuousActions;
    continuousActions[0] = Input.GetAxis("Horizontal");
    continuousActions[1] = Input.GetAxis("Vertical");
}
```

- **Heuristic learning** is mainly for **debugging and manual control**, not for AI-driven decision-making and it allows you to **manually define** how the agent behaves using the `Heuristic ()` method in the script.
- It does **not** use **machine learning** or training, and it is useful for **testing sensors, physics, and input controls** before switching to AI training.
- For training an AI model to make decisions and move toward a target, you need **Reinforcement Learning (RL) or Imitation Learning (IL)**.




# 4. Installation & Setup

## Installing ML-Agents

1. Install Python (3.8+ recommended).
2. Create a virtual environment:

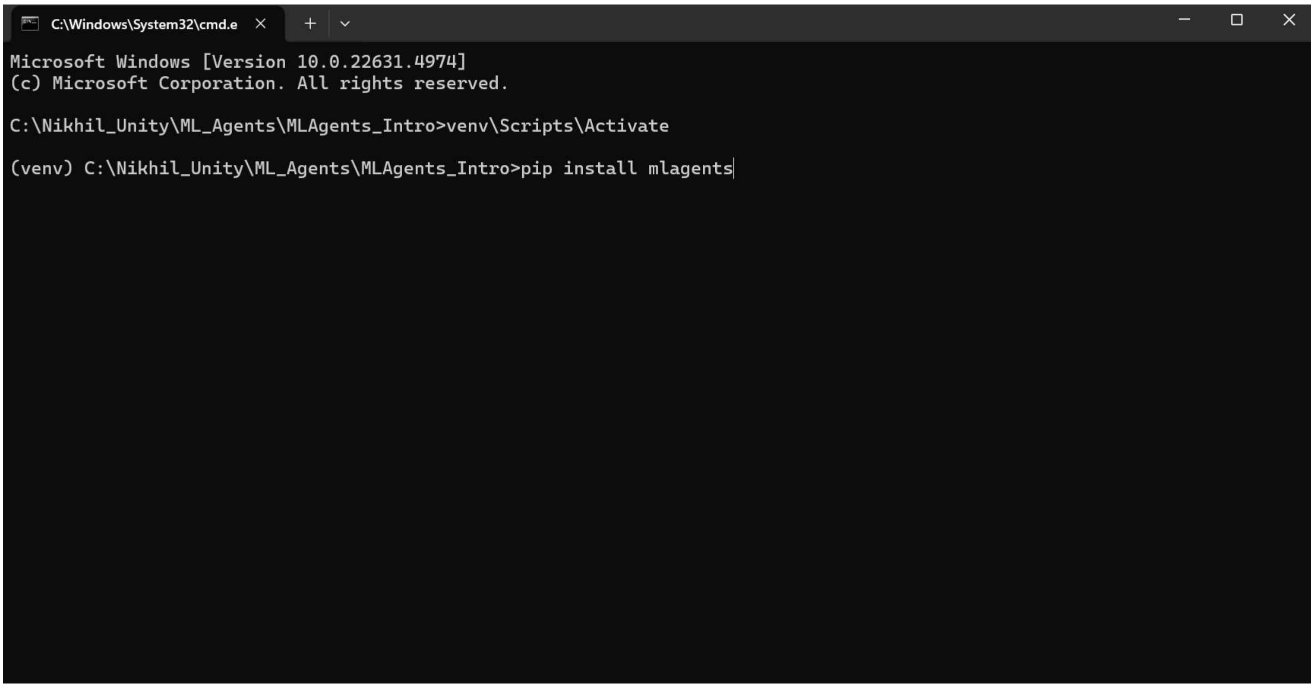
```
python -m venv venv
 source venv/bin/activate  (Linux/Mac)
 venv\Scripts\activate  (Windows)
```

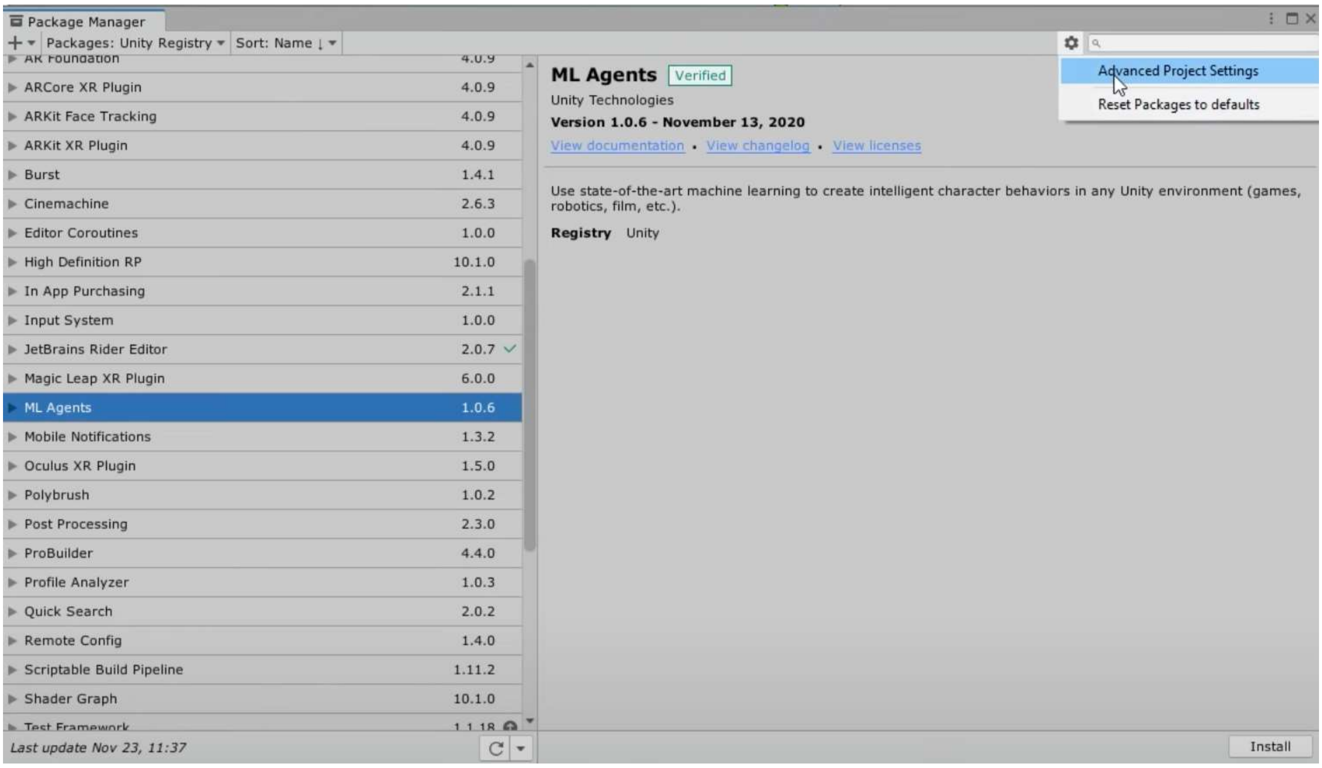
3. Install ML-Agents package:

```
pip install mlagents==0.29.0
```

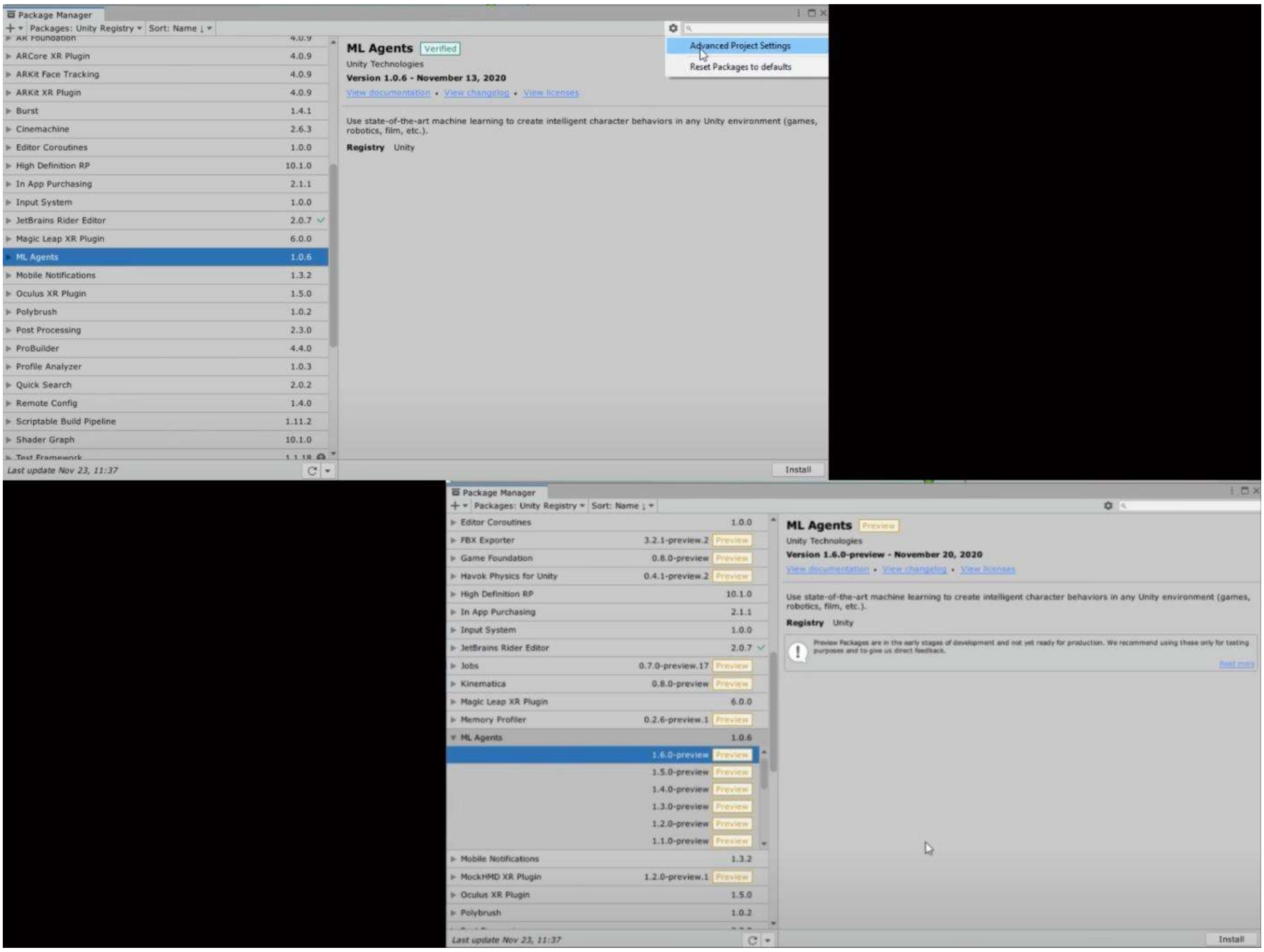
4. Verify installation:

```
mlagents-learn –help
```

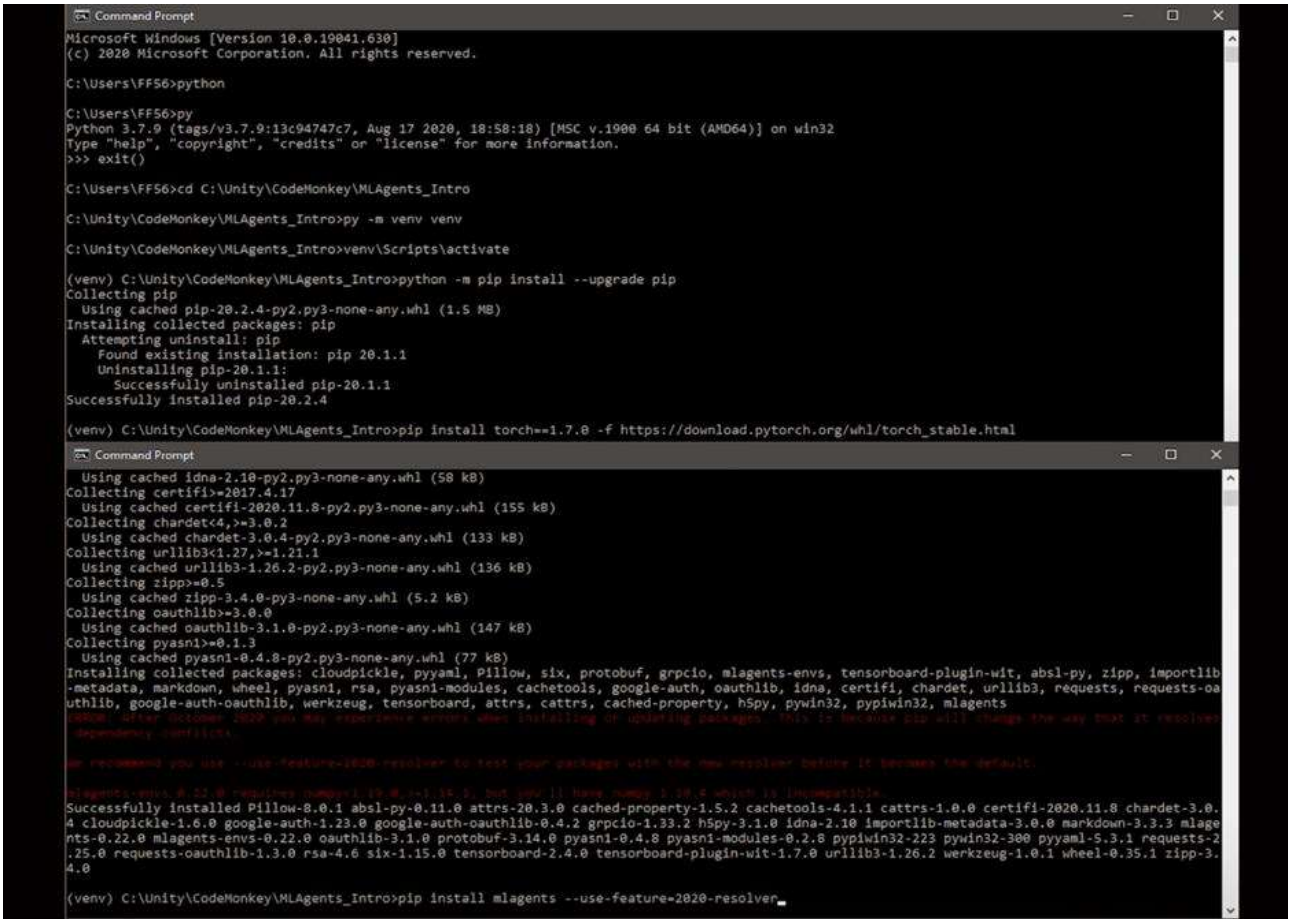





## 5.Additional Information to Preview Old Versions of Packages:

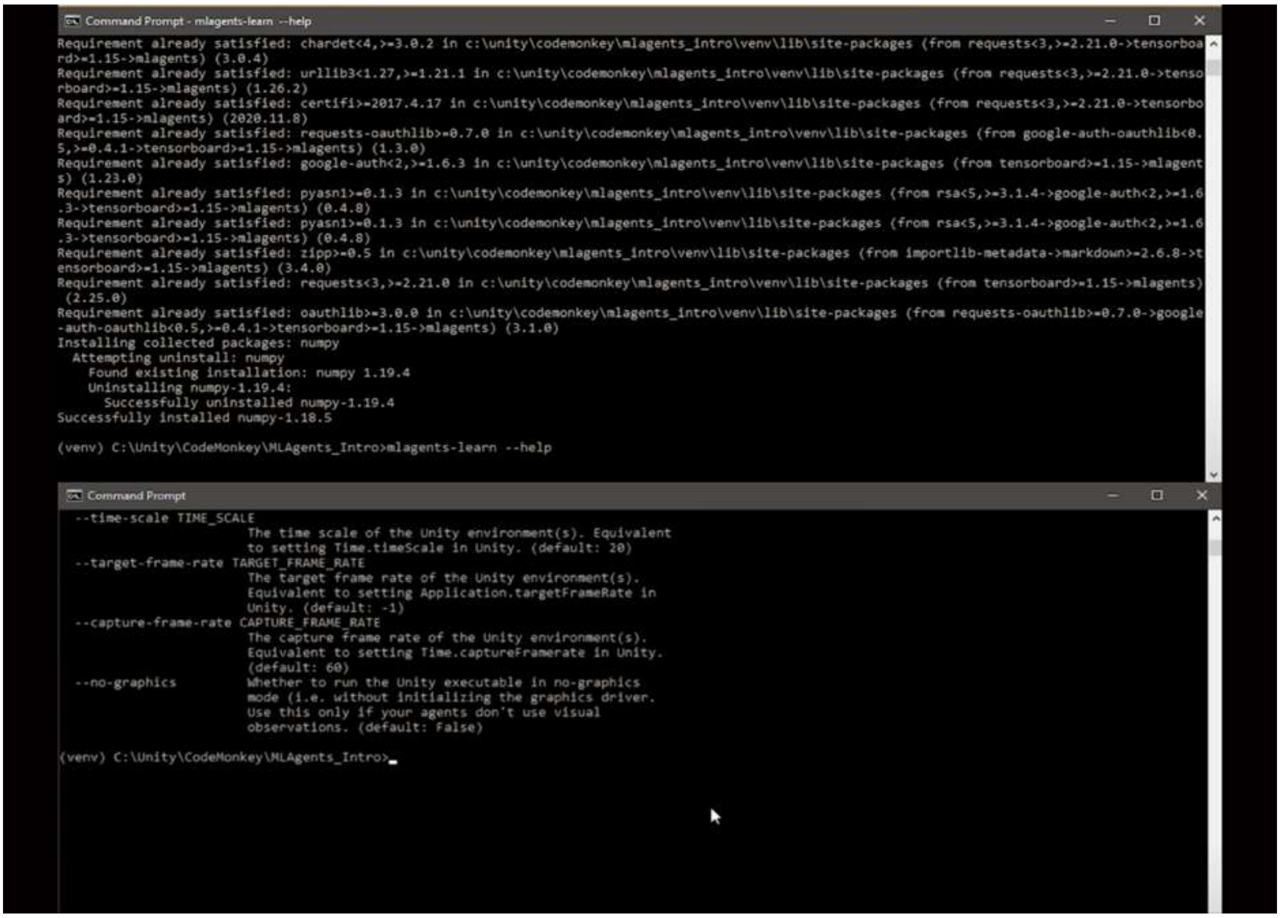



## 6. Pytorch Packages Installation & Setup

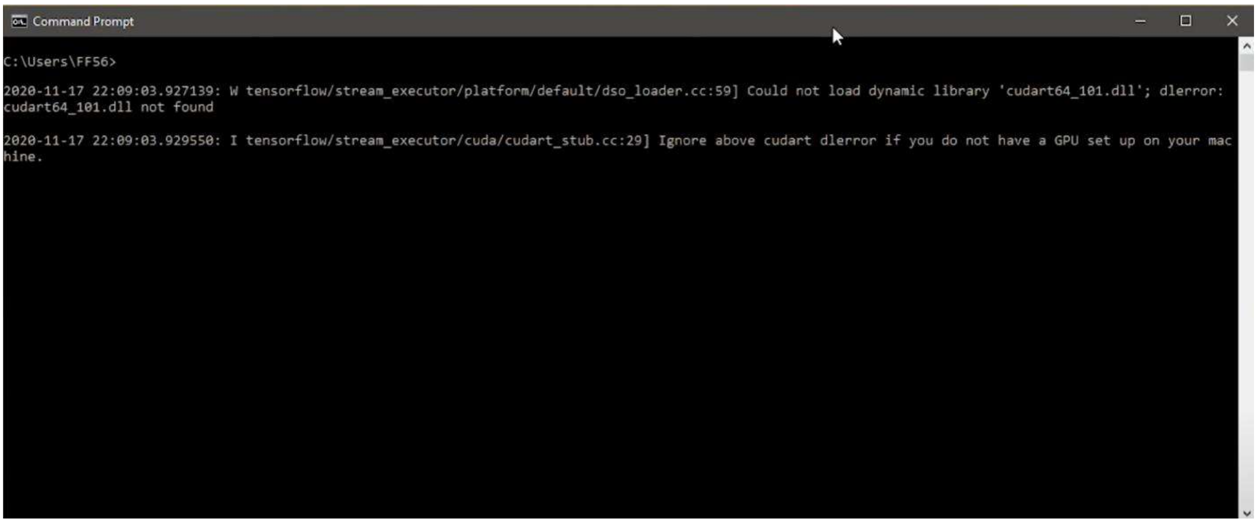


Command Prompt
Microsoft Windows [Version 10.0.19041.630]
(c) 2020 Microsoft Corporation. All rights reserved.
C:\Users\FF56>python
C:\Users\FF56>py
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()
C:\Users\FF56>cd C:\Unity\CodeMonkey\MLAgents_Intro
C:\Unity\CodeMonkey\MLAgents_Intro>py -m venv venv
C:\Unity\CodeMonkey\MLAgents_Intro>venv\Scripts\activate
(venv) C:\Unity\CodeMonkey\MLAgents_Intro>python -m pip install --upgrade pip
Collecting pip
Using cached pip-20.2.4-py2.py3-none-any.whl (1.5 MB)
Installing collected packages: pip
Attempting uninstall: pip
Found existing installation: pip 20.1.1
Uninstalling pip-20.1.1:
Successfully uninstalled pip-20.1.1
Successfully installed pip-20.2.4
(venv) C:\Unity\CodeMonkey\MLAgents_Intro>pip install torch==1.7.0 -f https://download.pytorch.org/whl/torch_stable.html
Command Prompt
Using cached idna-2.10-py2.py3-none-any.whl (58 kB)
Collecting certifi>=2017.4.17
Using cached certifi-2020.11.8-py2.py3-none-any.whl (155 kB)
Collecting chardet<4,>=3.0.2
Using cached chardet-3.0.4-py2.py3-none-any.whl (133 kB)
Collecting urllib3<1.27,>=1.21.1
Using cached urllib3-1.26.2-py2.py3-none-any.whl (136 kB)
Collecting zipp>=0.5
Using cached zipp-3.4.0-py3-none-any.whl (5.2 kB)
Collecting oauthlib>=3.0.0
Using cached oauthlib-3.1.0-py2.py3-none-any.whl (147 kB)
Collecting pyasn1>=0.1.3
Using cached pyasn1-0.4.8-py2.py3-none-any.whl (77 kB)
Installing collected packages: cloudpickle, pyyaml, Pillow, six, protobuf, grpcio, mlagents-envs, tensorboard-plugin-wit, absl-py, zipp, importlib-metadata, markdown, wheel, pyasn1, rsa, pyasn1-modules, cachetools, google-auth, oauthlib, idna, certifi, chardet, urllib3, requests, requests-oauthlib, google-auth-oauthlib, werkzeug, tensorboard, attrs, cattrs, cached-property, h5py, pywin32, pypiwin32, mlagents
Successfully installed Pillow-8.0.1 absl-py-0.11.0 attrs-20.3.0 cached-property-1.5.2 cachetools-4.1.1 cattrs-1.0.0 certifi-2020.11.8 chardet-3.0.4 cloudpickle-1.6.0 google-auth-1.23.0 google-auth-oauthlib-0.4.2 grpcio-1.33.2 h5py-3.1.0 idna-2.10 importlib-metadata-3.0.0 markdown-3.3.3 mlagents-0.22.0 mlagents-envs-0.22.0 oauthlib-3.1.0 protobuf-3.14.0 pyasn1-0.4.8 pyasn1-modules-0.2.8 pypiwin32-223 pywin32-300 pyyaml-5.3.1 requests-2.25.0 requests-oauthlib-1.3.0 rsa-4.6 six-1.15.0 tensorboard-2.4.0 tensorboard-plugin-wit-1.7.0 urllib3-1.26.2 werkzeug-1.0.1 wheel-0.35.1 zipp-3.4.0
(venv) C:\Unity\CodeMonkey\MLAgents_Intro>pip install mlagents --use-feature=2020-resolver

# 7. Verification, Installation & Setup of mlagents package



## 8. mlagents Dependencies Not Necessary Otherwise) CUDA || Cudnn Libraries Shown In This Case:



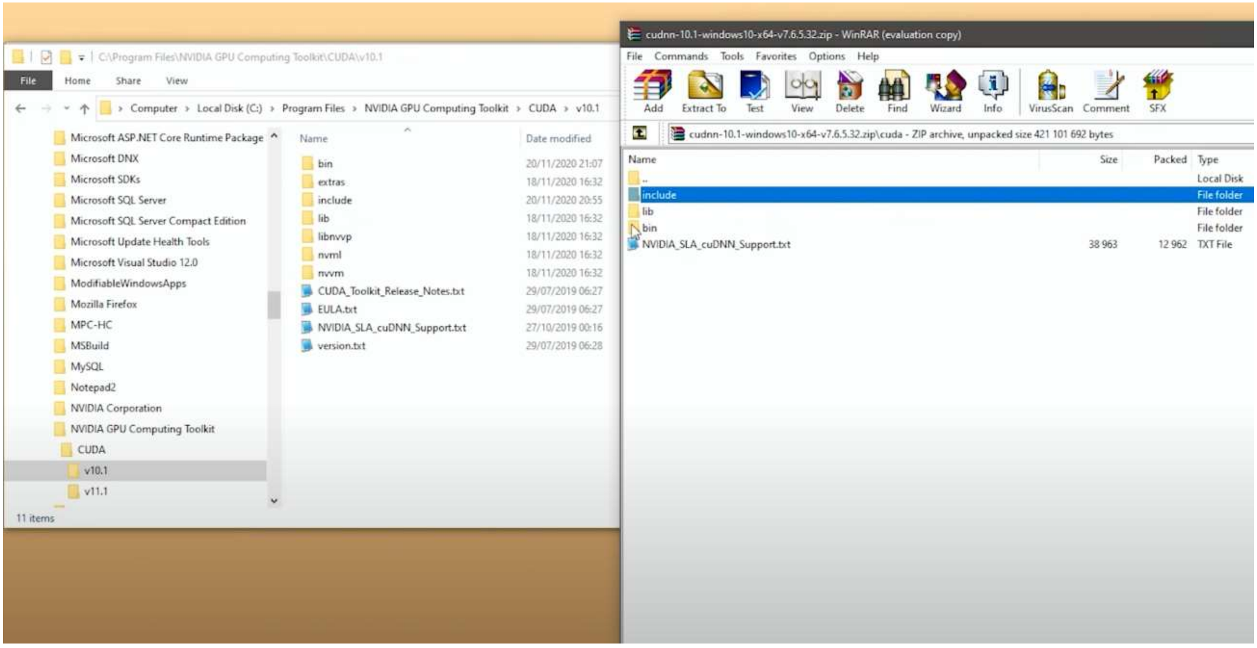

## 9. mlagents Dependencies Verification, Installation: CUDA || Cudnn Libraries As Shown Use These Content

### Install Recommended Versions Package To Not Facing Dependecies Conflicts

1. Install Python (3.8.10 recommended): Python 3.8.10
2. Install CUDA (11.8.0 recommended): CUDA 11.8.0
3. Install Cudnn (Cudnn v8.9.7 recommended): Cudnn v8.9.7

# Setup Cudnn into CUDA Package After Installation as Shown in Image:

When installing **cuDNN** for **CUDA**, you need to manually copy specific files from the **cuDNN package** into the **CUDA installation folder**. Follow these steps:




## 1️⃣ Locate Your CUDA and cuDNN Installation Paths

- **CUDA Installation Path (Default)**:

```makefile
makefile
CopyEdit
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vXX.X
```

*(Replace `XX.X` with your CUDA version, e.g., `v12.2`)*

- **cuDNN Extracted Folder**:

```makefile
CopyEdit
C:\Users\YourUsername\Downloads\cudnn-windows-x86_64-X.X.X.X_cudaXX.X
```

*(Replace X.X.X.X with your cuDNN version and XX.X with your CUDA version)*

## 2 Copy These Files From cuDNN to CUDA

| cuDNN Folder | Copy To (CUDA Folder) | Files to Copy |
|---|---|---|
| **bin** | C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vXX.X\bin | cudnn64_8.dll *(or latest version)* |
| **include** | C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vXX.X\include | cudnn.h |
| **lib\x64** | C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vXX.X\lib\x64 | cudnn.lib |

## 3 Verify Installation

After copying the files, verify that CUDA and cuDNN are correctly installed:

- **Check CUDA Version**

```cmd
CopyEdit
nvcc --version
```

- **Check If PyTorch Detects CUDA & cuDNN**

```cmd
CopyEdit
python -c "import torch; print(torch.cuda.is_available())"
```

```cmd
CopyEdit
```

```
python -c "import torch; print(torch.backends.cudnn.version())"
```

### 4️⃣ Set Environment Variables (If Needed)

If CUDA/cuDNN files are not detected, add these paths to the **Windows Environment Variables**:

1. **Press** `Win + R` → Type `sysdm.cpl` → Go to **Advanced > Environment Variables**.
2. Under **System Variables**, find `Path` → Click **Edit** → Add these:

```
vbnet
CopyEdit
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vXX.X\bin
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vXX.X\lib\x64
```

3. Click **OK** and **Restart Your PC**.

### 5️⃣ Final Verification

Run this command again to ensure everything is set up properly:

```
cmd
CopyEdit
python -c "import torch; print(torch.cuda.is_available(),
torch.backends.cudnn.version())"
```

4. Install ML-Agents (`0.29.0` recommended) :
   `Just by this command: pip install mlagents==0.29.0`

### Verify Installed Packages with Commands:

1. Python: python –version
2. CUDA: nvcc –version
3. Cudnn Didn't Need As We have to copy some of constraints from these into CUDA Package Folder

4. ML-Agents: pip show mlagents

## Behavior Type Options in Unity

| Behavior Type | Description | When to Use |
|---|---|---|
| Default | Trains during mlagents-learn, switches to inference when not training. | Recommended for training agents. |
| Inference Only | Uses a trained model, does not learn. | For testing a trained model in Unity. |
| Heuristic Only | Uses manually defined rules in the script. | For debugging or full manual control. |
| Player | Allows human control. | Useful for imitation learning setups. |

## Key Training Commands

| Command | Description |
|---|---|
| `mlagents-learn config.yaml --run-id=MyTraining --no-graphics` | Starts training in headless mode. |
| `mlagents-learn config.yaml --initialize-from=MyTraining --run-id=MyTraining2` | Resumes training from a previous run. |
| `mlagents-learn config.yaml --run-id=MyTraining --force` | Forces overwriting of previous training data. |
| `mlagents-learn config.yaml --run-id=MyTraining --inference` | Runs the trained model in inference mode. |

## Best Practices for Training ML-Agents

- **Longer training** improves performance but takes time.
- Use **reward shaping** to guide learning more efficiently.

- **Test different hyperparameters** (e.g., batch size, learning rate).
- Use **Curriculum Learning** for complex tasks.
- Save trained models periodically to avoid loss of progress.

## 10. Behavior Types in Unity

In the Unity Inspector, the **Behavior Type** setting determines how the agent acts:

- **Default**: Trains when learning is active; switches to inference when training stops.
- **Inference Only**: Uses a pre-trained model (no learning).
- **Heuristic Only**: Uses manual logic (custom script in `Heuristic ()` function).
- **Player**: Controlled by a human (useful for imitation learning).

## 11. Training an Agent

### Basic Command for Training

```
mlagents-learn config.yaml --run-id=MoveToGoal --no-graphics
```

- `--run-id=MoveToGoal`: Saves training data under this ID.
- `--no-graphics`: Runs Unity in headless mode (no rendering for faster training).

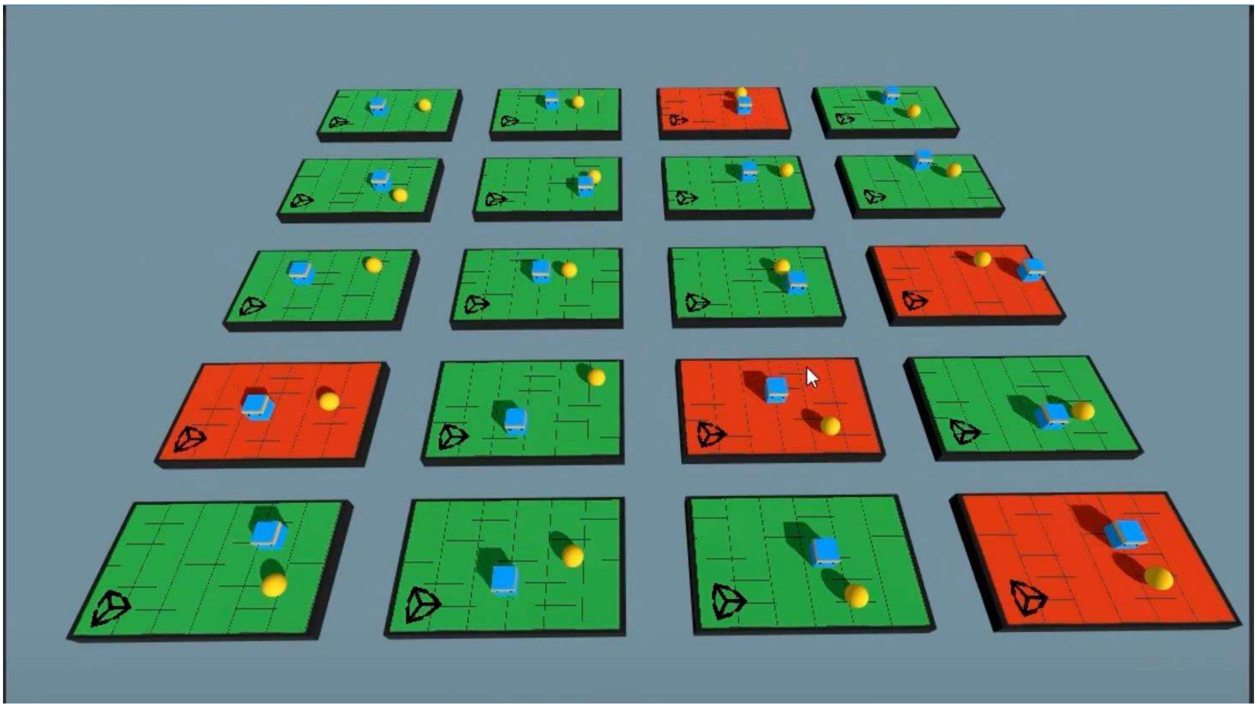
### Resume Training from Checkpoint

```
mlagents-learn config.yaml --run-id=MoveToGoal --resume --no-graphics
```

### Override Previous Training Data

```
mlagents-learn config.yaml --run-id=MoveToGoal --force --no-graphics
```

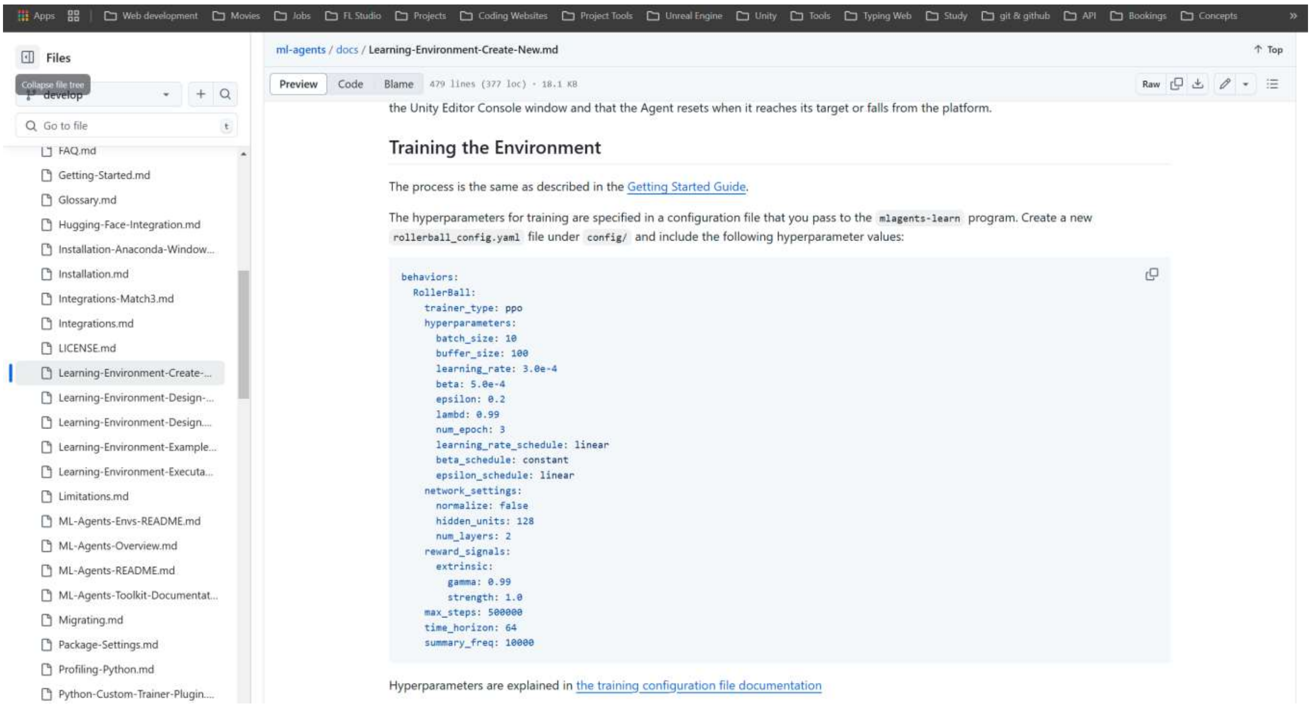

## While Training And Saving Model And Make him agent Like (Agent Vinod) :}



But before this we are facing issues like if we move the target object itself then the ai still having follow the previous training path which is not correct. So, the below recommendation point is a resolver for this issue, we have to update config,yaml that give like trainer to train specific model in given recommendations.

## 12. ML-Agents Configuration (config.yaml) File Update:

The below yaml file is taken from this that the developer has made and post it on to resolve previous issue and the link is: [Link: Go To Training the Environment and copy the below yaml file code]




Key parameters in YAML configuration file:
We have to replace "RollerBall" With "MoveToGoal" and We are done.

```
behaviors:
  MoveToGoal:
    trainer_type: ppo   # Proximal Policy Optimization (RL Algorithm)
    hyperparameters:
```

```
    batch_size: 1024
    buffer_size: 10240
    learning_rate: 0.0003
  max_steps: 500000  # Total training steps
  summary_freq: 50000  # How often to log progress
  checkpoint_interval: 500000  # When to save models
  network_settings:
    hidden_units: 128
    num_layers: 2
  reward_signals:
    extrinsic:
      gamma: 0.99
      strength: 1.0
```

Now we have another issue as we know about **ML-Agents Approaches or Learning Methods,** so the thing is if we make a focus on this references:

1.


2. Agent Class Script: Which we are inheriting that inherited monobehaviour with some interfaces and with constraints that needed to work with mlagents in unity to train AI based Models:

```
Assembly Unity.ML-Agents, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null

using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using Unity.Barracuda;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Demonstrations;
using Unity.MLAgents.Policies;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Sensors.Reflection;
using UnityEngine;
using UnityEngine.Serialization;

namespace Unity.MLAgents;

[Serializable]
[HelpURL("https://github.com/Unity-Technologies/ml-agents/blob/release_17_docs/docs/Learning-Environment-Design-Agents.md")]
[RequireComponent(typeof(BehaviorParameters))]
public class Agent : MonoBehaviour, ISerializationCallbackReceiver, IActionReceiver, IHeuristicProvider
{
    [Serializable]
    internal struct AgentParameters
    {
        public int maxStep;
    }

    private enum DoneReason
    {
        DoneCalled,
        MaxStepReached,
        Disabled
    }

    private IPolicy m_Brain;

    private BehaviorParameters m_PolicyFactory;

    [SerializeField]
    [HideInInspector]
    internal AgentParameters agentParameters;

    [SerializeField]
    [HideInInspector]
    internal bool hasUpgradedFromAgentParameters;

    [FormerlySerializedAs("maxStep")]
    [HideInInspector]
    public int MaxStep;

    private AgentInfo m_Info;

    private float m_Reward;

    private float m_GroupReward;

    private float m_CumulativeReward;

    private bool m_RequestAction;

    private bool m_RequestDecision;

    private int m_StepCount;
```

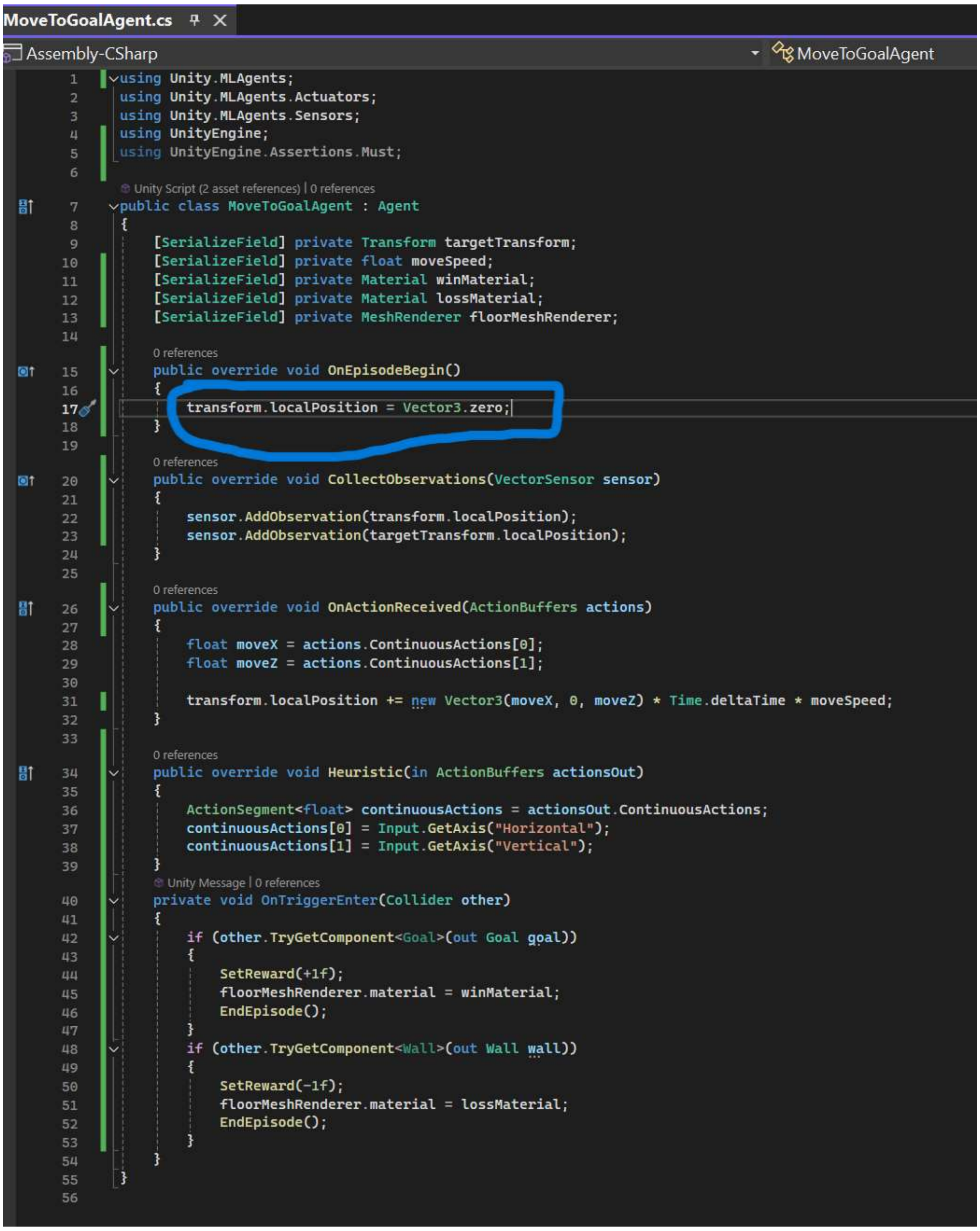
**3.** Custom Agent Script:

## Main Objectives of the Custom Agent Script

Your custom agent script does the following:

1. **Observing the Environment (`CollectObservations`)**
   a. The agent **perceives** the environment by collecting observations (e.g., position, velocity, surroundings).
   b. This data is passed to the neural network for decision-making.
2. **Taking Actions (`OnActionReceived`)**
   c. The agent receives actions from the ML model (or a heuristic function).
   d. It then applies those actions to control movement, rotation, jumping, etc.
3. **Defining Rewards (`SetReward & AddReward`)**
   e. Assign rewards for good behavior (e.g., moving towards a goal).
   f. Assign penalties for bad behavior (e.g., colliding with walls).
   g. This helps the agent learn optimal strategies.
4. **Handling Resets (`OnEpisodeBegin`)**
   h. Reset conditions at the start of each training episode.
   i. This ensures fair training by reinitializing positions, parameters, etc

## Custom Agent Script: MoveToGoalAgent:



MoveToGoalAgent.cs

Assembly-CSharp MoveToGoalAgent

```
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using UnityEngine;
using UnityEngine.Assertions.Must;

Unity Script (2 asset references) | 0 references
public class MoveToGoalAgent : Agent
{
    [SerializeField] private Transform targetTransform;
    [SerializeField] private float moveSpeed;
    [SerializeField] private Material winMaterial;
    [SerializeField] private Material lossMaterial;
    [SerializeField] private MeshRenderer floorMeshRenderer;

    0 references
    public override void OnEpisodeBegin()
    {
        transform.localPosition = Vector3.zero;
    }

    0 references
    public override void CollectObservations(VectorSensor sensor)
    {
        sensor.AddObservation(transform.localPosition);
        sensor.AddObservation(targetTransform.localPosition);
    }

    0 references
    public override void OnActionReceived(ActionBuffers actions)
    {
        float moveX = actions.ContinuousActions[0];
        float moveZ = actions.ContinuousActions[1];

        transform.localPosition += new Vector3(moveX, 0, moveZ) * Time.deltaTime * moveSpeed;
    }

    0 references
    public override void Heuristic(in ActionBuffers actionsOut)
    {
        ActionSegment<float> continuousActions = actionsOut.ContinuousActions;
        continuousActions[0] = Input.GetAxis("Horizontal");
        continuousActions[1] = Input.GetAxis("Vertical");
    }
    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        if (other.TryGetComponent<Goal>(out Goal goal))
        {
            SetReward(+1f);
            floorMeshRenderer.material = winMaterial;
            EndEpisode();
        }
        if (other.TryGetComponent<Wall>(out Wall wall))
        {
            SetReward(-1f);
            floorMeshRenderer.material = lossMaterial;
            EndEpisode();
        }
    }
}
```

Now, Remember the highlighted mark of blue color that will create issue like:
Let's assume I moved target that follow up by player or agent at runtime and that will not follow that path because the AI model is created or trained as we resetting target or agent or player position to vector zero, so it will not configuring all directions path or exploring more paths if object goes to other positions,
So, the thing is we have to give random positions to target or player or agent so it will take also random paths because as objects positions changes then paths exploring possibilities increases with respect to it and the complex model will be ready to

solve complex paths efficiently.

So, the given change we made to custom agent script is this to resolve this issue by giving random reset positions to both target or player or agent to eat as random observation and take actions and follow up life cycle approaches or methods AI Based algorithms as in given reference.

### Updated Custom Agent Script: MoveToGoalAgent:

MoveToGoalAgent.cs

Assembly-CSharp | MoveToGoalAgent

```csharp
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using UnityEngine;
using UnityEngine.Assertions.Must;

Unity Script (2 asset references) | 0 references
public class MoveToGoalAgent : Agent
{
    [SerializeField] private Transform targetTransform;
    [SerializeField] private float moveSpeed;
    [SerializeField] private Material winMaterial;
    [SerializeField] private Material lossMaterial;
    [SerializeField] private MeshRenderer floorMeshRenderer;

    0 references
    public override void OnEpisodeBegin()
    {
        transform.localPosition = new Vector3(Random.Range(3.8f, -3.8f), transform.localPosition.y, Random.Range(3.8f, -3.8f));
        targetTransform.localPosition = new Vector3(Random.Range(3.8f, -3.8f), targetTransform.localPosition.y, Random.Range(3.8f, -3.8f));
    }

    0 references
    public override void CollectObservations(VectorSensor sensor)
    {
        sensor.AddObservation(transform.localPosition);
        sensor.AddObservation(targetTransform.localPosition);
    }

    0 references
    public override void OnActionReceived(ActionBuffers actions)
    {
        float moveX = actions.ContinuousActions[0];
        float moveZ = actions.ContinuousActions[1];

        transform.localPosition += new Vector3(moveX, 0, moveZ) * Time.deltaTime * moveSpeed;
    }

    0 references
    public override void Heuristic(in ActionBuffers actionsOut)
    {
        ActionSegment<float> continuousActions = actionsOut.ContinuousActions;
        continuousActions[0] = Input.GetAxis("Horizontal");
        continuousActions[1] = Input.GetAxis("Vertical");
    }
    Unity Message | 0 references
    private void OnTriggerEnter(Collider other)
    {
        if (other.TryGetComponent<Goal>(out Goal goal))
        {
            SetReward(+1f);
            floorMeshRenderer.material = winMaterial;
            EndEpisode();
        }
        if (other.TryGetComponent<Wall>(out Wall wall))
        {
            SetReward(-1f);
            floorMeshRenderer.material = lossMaterial;
            EndEpisode();
        }
    }
}
```

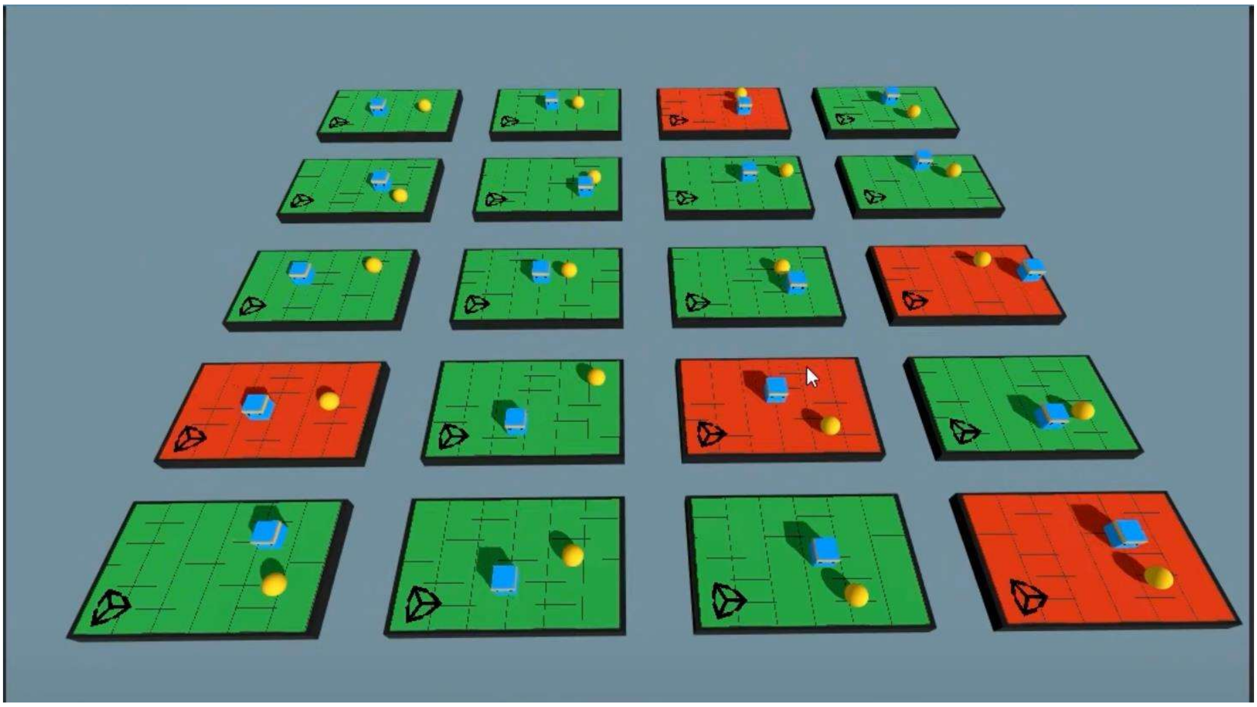
## How It Works in ML-Agents Training

1. **Training Phase**:
   a. The agent starts at a random position.
   b. It tries different movements using reinforcement learning.
   c. It gets **rewarded for moving towards the target**.
   d. It gets **penalized for moving away**.
   e. The model updates based on rewards, improving over time.
2. **Inference Phase** (Using Trained Model):
   a. After training, the agent applies its learned model to navigate toward the goal optimally.
   b. No further learning occurs; it simply executes decisions based on past training.

## Summary

Your **custom agent script** defines **what the agent sees, how it acts, how it learns, and how it resets**. This is **essential for training AI-controlled characters** in Unity ML-Agents.

**After 5 Steps We have to setup to train AI Based Model || While Training It looks like this:**



## 13. Understanding Training Phases

Each **step** of ML-Agent training goes through:

1. **Observation**: Agent receives inputs from environment.
2. **Decision Making**: ML model predicts best action.

3. **Action Execution**: Agent performs the action.
4. **Reward Assignment**: Agent gets feedback on performance.
5. **Learning Update**: Model is updated to improve future decisions.

## 14. Reinforcement Learning (RL) vs Imitation Learning

- **Reinforcement Learning (RL)**: The agent learns from trial and error based on rewards.
- **Imitation Learning**: The agent learns by mimicking human-controlled actions.

### Which One to Use?

- Use **RL** if the agent needs to discover solutions on its own.
- Use **Imitation Learning** if you want to guide the agent's behavior based on expert demonstrations.

## 15. Deploying a Trained Model

1. Train the model.
2. Locate the trained `.onnx` file in `results/MoveToGoal/`.
3. In Unity, assign the `.onnx` file to the **Behavior Parameters** of your agent.
4. Set **Behavior Type** to **Inference Only**.

## 16. Optimizing Training Performance

✅ Increase training time to allow the agent to fully learn. ✅ Improve the reward function to encourage perfect behaviors. ✅ Reduce exploration as training progresses to refine decisions. ✅ Use curriculum learning to gradually increase difficulty.

## 17. Useful Commands

| Command | Description |
|---|---|

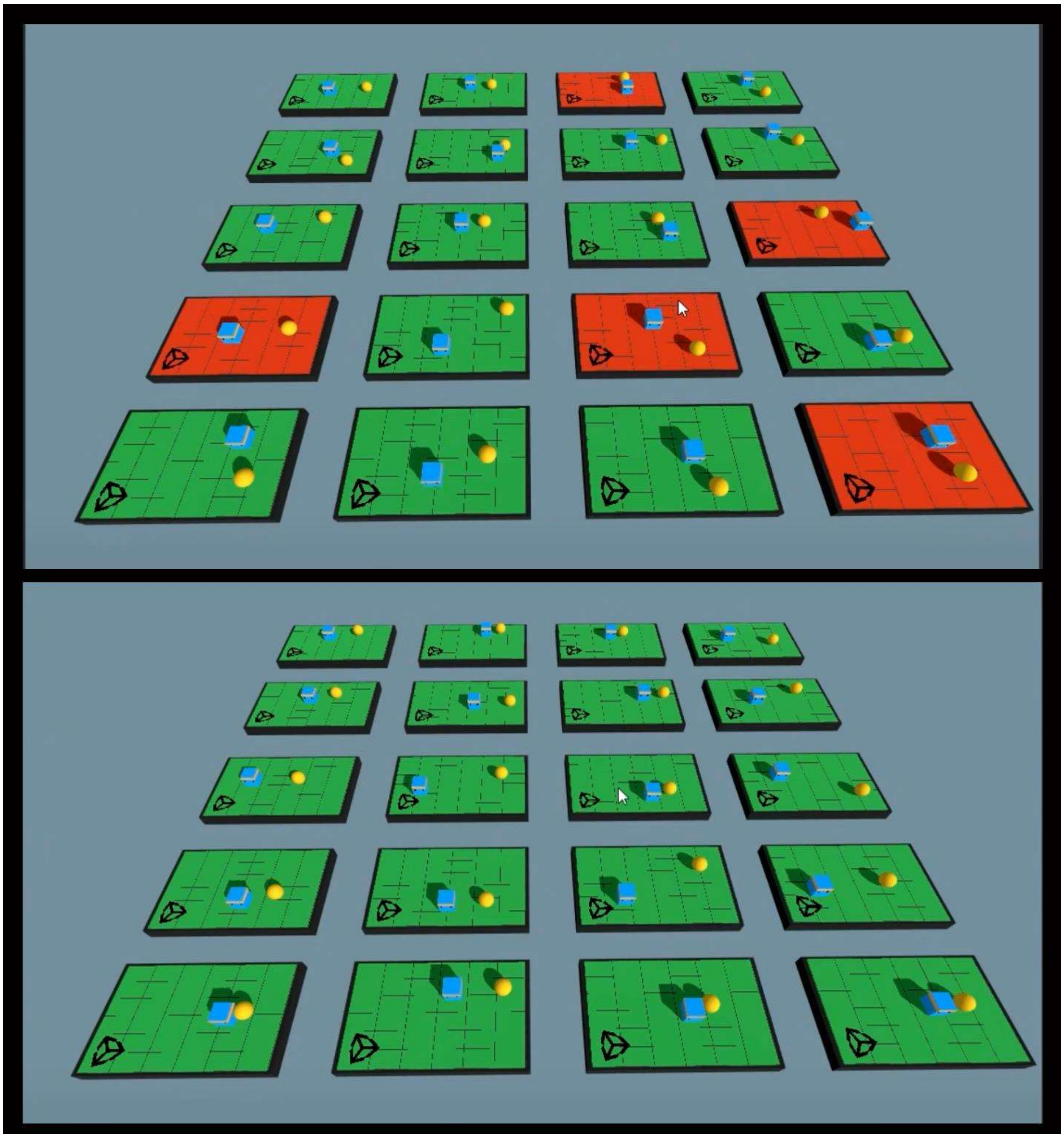
| | |
|---|---|
| `mlagents-learn config.yaml --run-id=MoveToGoal --no-graphics` | Start training |
| `mlagents-learn config.yaml --run-id=MoveToGoal --resume --no-graphics` | Resume training |
| `mlagents-learn config.yaml --run-id=MoveToGoal --force --no-graphics` | Override old training data |
| `mlagents-learn config.yaml --run-id=MoveToGoal --initialize-from=MoveToGoal2` | Initialize training from a previous model |

### Conclusion

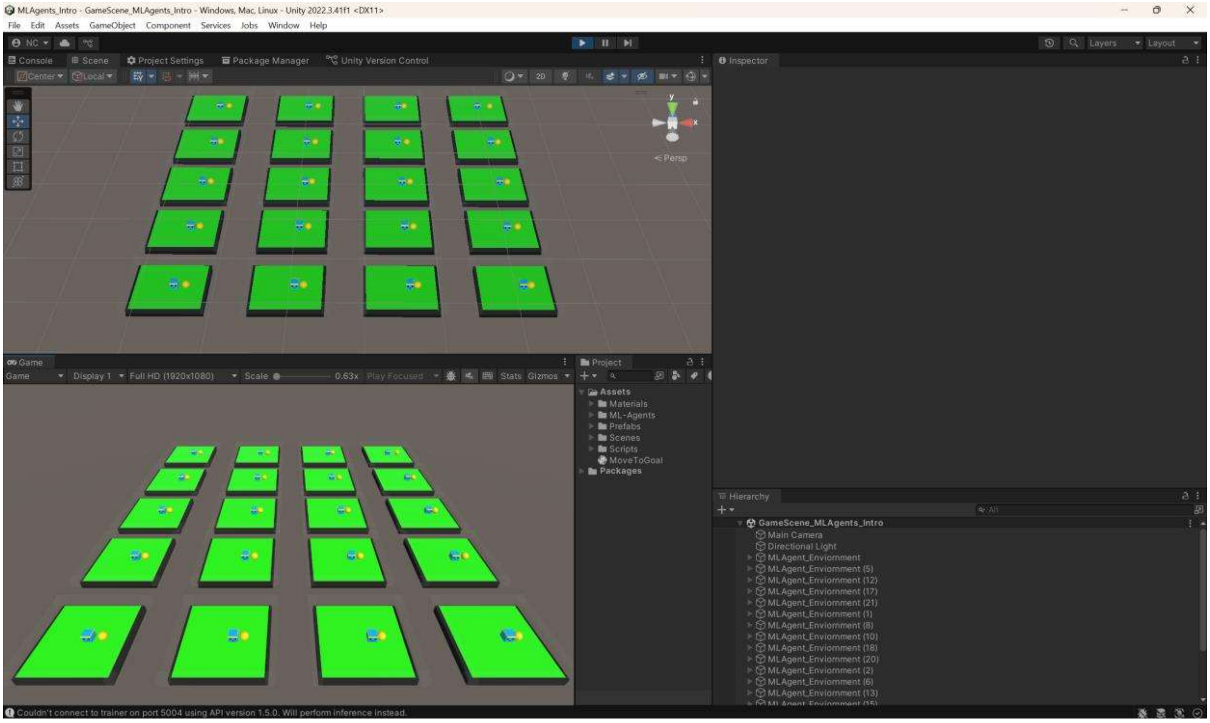
By understanding and tweaking these settings, you can train ML-Agents efficiently and optimize their performance for different use cases in Unity.

## 17. Results:

1.



2. **In My System Implementation of AI Based Trained Model Setup and Ready to Make Decisions on its Own:**




## 18. Link of Project If Wanted a Just Basic Setup for Work with MLAgents Instant In Unity:

1. **GitHub Link:** MLAgentBasicStartup_Demo

2. **MLAgents Package Requirements GoogelDrive Link:** MLAgents_Packages_Requirements

# Thank You

For attending the Orientation Session

Your participation is what make us Successful