AURA Robot Project - COMPLETE BEGINNER'S GUIDE v2.0

From Zero to 100 AI Robots - Every Single Step

🔥 **NOW WITH: Power Budget | Random Positions | 100 Parallel Robots | Smart Data Collection**

⏱️ **Total Time: 8-12 hours (spread over 2-3 days)** 💻 **Requirements: Windows/Mac computer with 8GB+ RAM (16GB+ recommended for 100 robots)** 📚 **Experience Needed: NONE - We'll teach you everything!**

🆕 **What's New in v2.0:**

- ✅ Power budget system (robots have limited energy per episode!)
- ✅ Random start/end positions (AI learns to generalize!)
- ✅ 100 parallel training environments (100x faster training!)
- ✅ Automatic best-performer tracking (only collects data from top robot!)

TABLE OF CONTENTS

DAY 1: SOFTWARE INSTALLATION

Step 1.1: Install Unity Hub (20 minutes)

What is Unity? Unity is a game development platform. We're using it to simulate our robot in 3D.

Windows:

1. Open your web browser

2. Go to: `https://unity.com/download`
3. Click **"Download Unity Hub"** (big blue button)
4. Once downloaded, find the file (usually in Downloads folder)
5. Double-click `UnityHubSetup.exe`
6. Click "Yes" if Windows asks for permission
7. Click **"I Agree"** on the license
8. Click **"Install"**
9. Wait for installation (2-5 minutes)
10. Click **"Finish"**
11. Unity Hub should open automatically

Mac:

1. Go to: `https://unity.com/download`
2. Click **"Download Unity Hub"**
3. Open the downloaded `.dmg` file
4. Drag Unity Hub to Applications folder
5. Open Unity Hub from Applications

✅ **Success Check:** Unity Hub window is open

Step 1.2: Create Unity Account (5 minutes)

1. In Unity Hub, click **"Sign in"** (top-right corner)
2. Click **"Create account"**
3. Fill in:
   - Email address
   - Password (write this down!)
   - Username
4. Check your email for verification
5. Click the verification link
6. Return to Unity Hub
7. Sign in with your new account

✅ **Success Check:** You're signed into Unity Hub

Step 1.3: Get Unity Personal License (2 minutes)

1. In Unity Hub, click your account icon (top-right)
2. Click **"Manage Licenses"**
3. Click **"Add"** or **"Get new license"**
4. Select **"Get a free personal license"**
5. Check "I don't use Unity in a professional capacity"
6. Click **"Done"**

✅ **Success Check:** License shows "Personal" in Unity Hub

Step 1.4: Install Unity Editor (30 minutes)

What's the Editor? This is the actual program where you'll build your robot.

1. In Unity Hub, click **"Installs"** (left sidebar)
2. Click **"Install Editor"** or **"Add"** (top-right)
3. Choose **"Unity 2022.3.x LTS"** (LTS = Long Term Support, most stable)
    - Look for the one with a little "LTS" badge
    - 2022.3 or 2021.3 both work perfectly!
4. Click **"Install"** or **"Next"**
5. On "Add modules" screen, CHECK these boxes:
    - ✅ **WebGL Build Support** (for web deployment)
    - ✅ **Visual Studio** or **Visual Studio Code** (for coding)
    - ✅ **Documentation** (helpful!)
6. Click **"Continue"** or **"Install"**
7. Wait 15-30 minutes (it's a big download!)
8. Get a snack ☕

✅ **Success Check:** Unity 2022.3.x or 2021.3.x appears in your "Installs" list with a green checkmark

---

Step 1.5: Install Python (15 minutes)

What's Python? A programming language we'll use to train the robot's AI brain.

⚠️ **CRITICAL: Must be Python 3.10 or older! NOT 3.11 or 3.12!**

Windows:

1. Go to: `https://www.python.org/downloads/`
2. Scroll down to find **"Python 3.10.11"** or similar 3.10.x version
   - Click "Download Python 3.10.11"
   - **DO NOT download 3.11 or 3.12!**
3. Run the downloaded installer
4. ⚠️ **CRITICAL:** Check the box **"Add Python to PATH"** at the bottom!
5. Click **"Install Now"**
6. Wait for installation
7. Click **"Close"**

Mac:

1. Go to: `https://www.python.org/downloads/`
2. Download Python 3.10.x (NOT 3.11+)
3. Open the downloaded `.pkg` file
4. Follow the installer
5. Click "Install"
6. Enter your Mac password if asked

Verify Python Installation:

1. **Windows:** Press `Win + R`, type `cmd`, press Enter
2. **Mac:** Press `Cmd + Space`, type `terminal`, press Enter
3. Type: `python --version` and press Enter
4. Should show: `Python 3.10.x`

If it shows 3.11 or 3.12:

- Uninstall Python
- Download 3.10.11 specifically
- Reinstall with "Add to PATH" checked

✅ **Success Check:** Python version shows 3.10.x

Step 1.6: Install Git (10 minutes)

What's Git? A tool that helps download code packages.

Windows:

1. Go to: `https://git-scm.com/download/win`
2. Download automatically starts (64-bit version)
3. Run the installer
4. Click **"Next"** on everything (default settings are fine)
5. Click **"Install"**
6. Click **"Finish"**

Mac:

Git is usually pre-installed. To check:

1. Open Terminal
2. Type: `git --version`
3. If not installed, it will prompt you to install Xcode Command Line Tools
4. Click "Install" and follow prompts

✅ **Success Check:** In command prompt/terminal, `git --version` shows a version number

---

🎉 DAY 1 CHECKPOINT: All Software Installed!

- Unity Hub ✅
- Unity Editor 2022.3 or 2021.3 LTS ✅
- Unity Personal License ✅
- Python 3.10.x ✅
- Git ✅

Take a break! Tomorrow we'll build the robot! 🤖

UNITY PROJECT SETUP

Step 2.1: Create Your Unity Project (5 minutes)

1. Open **Unity Hub**
2. Click **"Projects"** (left sidebar)
3. Click **"New Project"** (top-right, blue button)
4. In the templates, select **"3D Core"** or **"3D (Built-In Render Pipeline)"**
   - ✅ **3D Core** - CORRECT
   - ✅ **3D (Built-In)** - CORRECT
   - ❌ **NOT** "3D (URP)" or "3D (HDRP)"
   - ❌ **NOT** "3D with Extras"
5. At the bottom:
   - **Project Name:** Type `AURA-Robot-Sim`
   - **Location:** Choose where to save (Documents is fine)
   - **Unity Version:** Should show 2022.3.x or 2021.3.x
6. **Version Control:** Leave UNCHECKED (not needed)
7. Click **"Create Project"**
8. Wait 2-5 minutes while Unity sets up
9. Unity Editor will open!

✅ **Success Check:** Unity Editor is open with a blank 3D scene

---

Step 2.2: Unity Interface Tour (5 minutes)

Let me explain what you're seeing:

```
┌─────────────────────────────────────────────────┐
│ File  Edit  Assets  GameObject  Component  Window │ ← Menu Bar
├─────────┬─────────────────────┬─────────────────┤
│         │                     │                 │
│ Hierarchy  Scene View (3D)    │   Inspector     │
│         │                     │                 │
│ (List of│  [Camera view/      │  (Properties of │
│  objects│   Building area]    │  selected item) │
│  in     │                     │                 │
│  scene) │                     │                 │
│         │                     │                 │
├─────────┴─────────────────────┴─────────────────┤
│      Project (Your Files)      Console         │
│      [Assets, Scenes, etc.]    [Messages]      │
└─────────────────────────────────────────────────┘
```

Key Windows:

- **Hierarchy** (left): List of everything in your scene
- **Scene View** (center): Where you build your 3D world
- **Inspector** (right): Settings for selected objects
- **Project** (bottom): Your files and folders
- **Console** (bottom, tabs): Shows messages and errors

Pro Tip: If you can't find a window:

- Go to **Window** menu → Find the window name
- Click it to open

⚠️ **If you see warnings about "Assertion failed":**

- These are Unity bugs - IGNORE THEM
- They don't affect your project
- Just close the warning

Step 2.3: Install ML-Agents Package (10 minutes)

What's ML-Agents? Unity's machine learning toolkit - the AI brain system!

Method 1: Git URL (Recommended)

1. In Unity, click **Window** → **Package Manager**
2. Wait for Package Manager window to open
3. Click the **"+"** button (top-left corner)
4. Select **"Add package from git URL"**
5. Type this EXACTLY:

```
com.unity.ml-agents@2.3.0-exp.3
```

6. Press **Enter**
7. Wait 1-2 minutes for installation
8. When done, it will say "ML Agents 2.3.0"

Method 2: If Git URL doesn't work

1. Go to: https://github.com/Unity-Technologies/ml-agents/releases/tag/release_20
2. Scroll down, click **"Assets"**
3. Download **"Source code (zip)"**
4. Extract the ZIP file
5. In Unity Package Manager:
    - Click **"+"** → **"Add package from disk"**
    - Navigate to extracted folder → `ml-agents-release_20` → `com.unity.ml-agents`
    - Select `package.json`
    - Click **"Open"**

✅ **Success Check:**

- Package Manager shows "ML Agents"
- Version: **2.3.0** or **2.3.0-exp.3**
- Status: "Up to date" or checkmark
- **NOT 2.0.2** (if you see 2.0.2, use Method 1 to upgrade)

Step 2.4: Save Your Scene (2 minutes)

IMPORTANT: Save often in Unity! It can crash!

1. Press **Ctrl + S** (Windows) or **Cmd + S** (Mac)
2. Name your scene: `RobotTraining`
3. Click **"Save"**

✅ **Success Check:**

- Bottom of scene view shows: "RobotTraining"
- In Project window → Assets → Scenes → RobotTraining.unity exists

---

🎉 **CHECKPOINT: Unity Project Created!**

- Project created ✅
- ML-Agents 2.3.0 installed ✅
- Scene saved ✅

Next: We build the robot! 🔧

BUILDING THE ROBOT SCENE

This is the fun part! We'll create ONE complete training area, then duplicate it 100 times later!

We're building:

- The floor for the robot to work on
- Start zone (green) and goal zone (red) - but they'll move randomly!
- A box for the robot to move
- The robot arm itself!

Step 3.1: Create Container for Training Area (2 minutes)

First, we'll organize everything into one container that we can duplicate!

1. In **Hierarchy** window (left side), right-click in empty space
2. Select **"Create Empty"**
3. An empty GameObject appears
4. Rename it: `TrainingArea_0`
5. In **Inspector** → **Transform**:

- Position: X = 0, Y = 0, Z = 0
- Make sure it's at the origin!

✅ **Success Check:** TrainingArea_0 exists at position (0, 0, 0)

Step 3.2: Create the Floor (5 minutes)

1. Right-click on **TrainingArea_0** (so it becomes the parent!)
2. Hover over **3D Object**
3. Click **Plane**
4. A flat plane appears as a child of TrainingArea_0!

Name it: 5. With the plane selected, look at **Inspector** (right side) 6. At the very top, change "Plane" to: `Floor` 7. Press Enter

Position it (should already be correct, but verify): 8. In Inspector → Transform:

- Position: X = 0, Y = 0, Z = 0
- Rotation: X = 0, Y = 0, Z = 0
- Scale: X = 3, Y = 1, Z = 3

Make it look nice: 9. Right-click in **Project** window (bottom) → **Create** → **Material** 10. Name it: `Floor_Mat` 11. Click on Floor_Mat in Project 12. In Inspector, find **Albedo** (color setting) 13. Click the white rectangle next to it 14. Choose a light gray color 15. In Hierarchy, drag **Floor_Mat** from Project onto **Floor**

✅ **Success Check:** You have a gray floor under TrainingArea_0!

Step 3.3: Create Start Zone (Green Zone) (5 minutes)

1. Right-click on **TrainingArea_0** → 3D Object → **Cube**
2. Rename it: `TargetZoneA`

Position: 3. In Inspector → Transform:

- Position X: `-4`
- Position Y: `0.05`
- Position Z: `0`

Scale (make it flat): 4. Transform → Scale:

- X: 1
- Y: 0.1 (very flat!)
- Z: 1

Make it green: 5. Project window → Right-click → Create → Material 6. Name: Target_Green 7. Click on Target_Green 8. Click the white box next to Albedo 9. Choose a bright green color 10. Drag Target_Green onto TargetZoneA in Hierarchy

✅ **Success Check:** Green flat square at (-4, 0.05, 0) under TrainingArea_0!

Step 3.4: Create Goal Zone (Red Zone) (3 minutes)

Easy way - Duplicate the green zone!

1. Click on **TargetZoneA** in Hierarchy
2. Press **Ctrl + D** (Windows) or **Cmd + D** (Mac)
3. A copy appears! Rename it: TargetZoneB
4. Make sure it's still under TrainingArea_0!

Move it to the right: 5. In Inspector → Transform → Position:

- X: 4 (positive 4, opposite side)
- Y: 0.05
- Z: 0

Make it red: 6. Create new material: Target_Red 7. Make it bright red 8. Drag Target_Red onto TargetZoneB

NOTE: These zones will move randomly during training! They're just visual indicators.

✅ **Success Check:** Red zone on right at (4, 0.05, 0), green zone on left!

Step 3.5: Create the Movable Box (5 minutes)

This is what the robot will pick up and move!

1. Right-click **TrainingArea_0** → 3D Object → **Cube**
2. Rename: MovableBox

Position (start on green zone): 3. Inspector → Transform → Position:

- X: `-4`
- Y: `0.75` (floating above green zone)
- Z: `0`

Scale (make it smaller): 4. Transform → Scale:

- X: `0.5`
- Y: `0.5`
- Z: `0.5`

Add Physics (IMPORTANT!): 5. With MovableBox selected, click **Add Component** (bottom of Inspector) 6. Type: `Rigidbody` 7. Click **Rigidbody** when it appears 8. In the Rigidbody component:

- Mass: `1`
- Drag: `0.5`
- Angular Drag: `0.5`
- ✅ Use Gravity: **CHECKED**

Make it blue: 9. Create material: `Box_Mat` 10. Make it blue 11. Apply to MovableBox

NOTE: Box position will randomize during training!

✅ **Success Check:** Small blue cube at (-4, 0.75, 0) with Rigidbody attached!

Step 3.6: Create the Robot Base (5 minutes)

Now we build the robot arm!

1. Right-click on **TrainingArea_0** → **Create Empty**
2. Rename: `RobotArm`
3. Inspector → Transform → Position: `0, 0, 0`

This will hold all robot parts!

Your Hierarchy should now look like:

```
TrainingArea_0
├── Floor
├── TargetZoneA
├── TargetZoneB
├── MovableBox
└── RobotArm (empty for now)
```

✅ **Success Check:** RobotArm is a child of TrainingArea_0 at (0,0,0)

Step 3.7: Create Base Rotation Platform (10 minutes)

This part rotates the whole arm 360°!

1. Right-click on **RobotArm** → 3D Object → **Cylinder**
2. Rename: `BaseRotation`

Position (relative to RobotArm): 3. Inspector → Transform:

- Position X: `0`
- Position Y: `0.25`
- Position Z: `0`

Scale: 4. Transform → Scale:

- X: `0.4`
- Y: `0.25`
- Z: `0.4`

Add the special physics (Articulation Body): 5. Click **Add Component** 6. Type: `Articulation Body` 7. Click it to add

Configure Articulation Body: 8. Find the **Articulation Body** component in Inspector 9. **Articulation Body Type:** Select **Fixed** (anchored to world)

Setup rotation controls: 10. Scroll down to find **X Drive** 11. Click the arrow next to **X Drive** to expand it 12. Set these values: - Lower Limit: `-180` - Upper Limit: `180` - Stiffness: `10000` - Damping: `1000` - Force Limit: `100`

Make it dark gray/black: 13. Create material: `Base_Mat` (dark gray or black) 14. Apply to BaseRotation

✅ **Success Check:** Dark cylinder at (0, 0.25, 0) under RobotArm with Articulation Body!

Step 3.8: Create Femur (Upper Arm) (5 minutes)

1. Right-click **BaseRotation** → 3D Object → **Cube**
2. Rename: `Femur`

Position (Local - notice the position is relative to parent!): 3. Inspector → Transform:

- Position X: `0`
- Position Y: `1.25`
- Position Z: `0`

Scale (tall and thin): 4. Transform → Scale:

- X: `0.25`
- Y: `2` (tall!)
- Z: `0.25`

Make it blue: 5. Create material: `Arm_Blue` (blue color) 6. Apply to Femur

✅ **Success Check:** Tall blue rectangle sticking up from base

---

Step 3.9: Create Femur Joint (Shoulder) (10 minutes)

This joint lets the upper arm rotate!

1. Right-click **Femur** → 3D Object → **Sphere**
2. Rename: `FemurJoint`

Position: 3. Transform → Position:

- X: `0`
- Y: `1` (at top of Femur)
- Z: `0`

Scale: 4. Transform → Scale:

- X: `0.35`
- Y: `0.35`
- Z: `0.35`

Add Articulation Body: 5. Add Component → **Articulation Body**

Configure: 6. **Articulation Body Type:** Select **Revolute** (rotating joint!) 7. **Anchor Rotation:**

- X: `0`
- Y: `0`
- Z: `90` (this sets rotation axis)

8. **Parent Anchor Position:**
    - X: `0`
    - Y: `1.25`
    - Z: `0`
9. **X Drive** (expand it):
    - Lower Limit: `-90`
    - Upper Limit: `135`
    - Stiffness: `10000`
    - Damping: `1000`
    - Force Limit: `100`

Make it gray: 10. Create material: `Joint_Gray` (gray color) 11. Apply to FemurJoint

✅ **Success Check:** Gray sphere at top of blue arm with Articulation Body (Revolute)!

Step 3.10: Create Shin (Lower Arm) (5 minutes)

1. Right-click **FemurJoint** → 3D Object → **Cube**
2. Rename: `Shin`

Position: 3. Transform → Position:

- X: `0`
- Y: `1`
- Z: `0`

Scale: 4. Transform → Scale:

- X: `0.2`
- Y: `1.8`

- Z: 0.2

Make it blue: 5. Apply **Arm_Blue** material (same as Femur)

✅ **Success Check:** Another blue arm piece extending from gray sphere!

Step 3.11: Create Knee Joint (10 minutes)

1. Right-click **Shin** → 3D Object → **Sphere**
2. Rename: KneeJoint

Position: 3. Transform → Position:

- X: 0
- Y: 0.9
- Z: 0

Scale: 4. Transform → Scale:

- X: 0.3
- Y: 0.3
- Z: 0.3

Add Articulation Body: 5. Add Component → **Articulation Body**

Configure: 6. **Articulation Body Type: Revolute** 7. **Anchor Rotation:**

- X: 0
- Y: 0
- Z: 90

8. **Parent Anchor Position:**
    - X: 0
    - Y: 1
    - Z: 0
9. **X Drive:**
    - Lower Limit: -150
    - Upper Limit: 0
    - Stiffness: 10000

- Damping: `1000`
- Force Limit: `100`

Make it gray: 10. Apply **Joint_Gray** material

✅ **Success Check:** Gray sphere at end of second arm piece with Articulation Body!

Step 3.12: Create Foot Magnet (CRITICAL!) (10 minutes)

This is the magic part - the magnetic pickup!

1. Right-click **KneeJoint** → 3D Object → **Sphere**
2. Rename: `FootMagnet`

Position: 3. Transform → Position:

- X: `0`
- Y: `0.9`
- Z: `0`

Scale: 4. Transform → Scale:

- X: `0.3`
- Y: `0.3`
- Z: `0.3`

ADD THE TRIGGER COLLIDER (VERY IMPORTANT!): 5. Click **Add Component** 6. Type: `Sphere Collider` 7. Click **Sphere Collider** to add it 8. In the Sphere Collider component:

- ✅ **Is Trigger:** CHECK THIS BOX! (CRITICAL!)
- Radius: `0.5`

Make it gold/yellow (magnet color!): 9. Create material: `Magnet_Gold` (bright yellow or gold) 10. Apply to FootMagnet

✅ **Success Check:**

- Yellow/gold sphere at end of arm
- Has **Sphere Collider** component
- **"Is Trigger"** is **CHECKED** ← Super important!

Step 3.13: Fix the Camera View (5 minutes)

Let's position camera so we can see everything!

1. In Hierarchy, click **Main Camera**
2. Inspector → Transform:
    - Position X: `0`
    - Position Y: `10`
    - Position Z: `-10`
3. Transform → Rotation:
    - X: `45`
    - Y: `0`
    - Z: `0`

Test the view: 4. Click on **Scene** tab (top of Scene view) 5. You should see your whole setup from above at an angle!

✅ **Success Check:** Can see floor, zones, box, and robot from nice angle

---

Step 3.14: Verify Your Complete Hierarchy (2 minutes)

Your Hierarchy should look EXACTLY like this:

```
Scene
├── Main Camera
├── Directional Light
└── TrainingArea_0
    ├── Floor
    ├── TargetZoneA
    ├── TargetZoneB
    ├── MovableBox (with Rigidbody)
    └── RobotArm
        └── BaseRotation (Articulation Body - Fixed)
            └── Femur
                └── FemurJoint (Articulation Body - Revolute)
                    └── Shin
                        └── KneeJoint (Articulation Body - Revolute)
                            └── FootMagnet (Sphere Collider with Is Trigger ✓)
```

If it doesn't match:

- Drag and drop objects to rearrange
- Each indentation = child of parent above
- Make sure everything is under TrainingArea_0 except Camera and Light

✅ **Success Check:** Hierarchy matches the structure above perfectly!

Step 3.15: SAVE YOUR WORK! (1 minute)

Press Ctrl + S (or Cmd + S) RIGHT NOW!

Unity can crash! Save often!

🎉 HUGE CHECKPOINT: Single Training Area Complete!

Take a break! You just built a complete robotic training environment! 🤖 ✨

What you have:

- Floor ✅
- Green start zone ✅
- Red goal zone ✅
- Blue movable box ✅

- Complete 2-joint robot with magnetic foot ✅
- All organized under TrainingArea_0 ✅

Next: We'll add the AI code, then duplicate this 100 times! 💻

ADDING THE CODE

Now we program the robot's brain! I've written all the code for you. Just copy it!

Step 4.1: Create Folders for Scripts (3 minutes)

1. In **Project** window (bottom), click on **Assets** folder
2. Right-click in empty space → **Create** → **Folder**
3. Name it: `Scripts`
4. Press Enter
5. Double-click **Scripts** folder to open it
6. Create these subfolders inside Scripts:
    - Right-click → Create → Folder → Name: `Agent`
    - Right-click → Create → Folder → Name: `Data`
    - Right-click → Create → Folder → Name: `Environment`

Your structure:

```
Assets
└── Scripts
    ├── Agent
    ├── Data
    └── Environment
```

Also create: 7. In Assets, create folder: `Prefabs`

✅ **Success Check:** You have Assets/Scripts/Agent, Data, Environment and Assets/Prefabs folders

---

Step 4.2: Get All the Code Files

You should have these files from me:

1. `RobotAgent_NEW.cs` - Robot's AI brain

2. `DataCollector_NEW.cs` - Data collection system
3. `PerformanceTracker.cs` - Tracks best performing robot
4. `TrainingAreaSpawner.cs` - Creates 100 training areas

If you don't have them, I'll provide them below!

Step 4.3: Create RobotAgent Script (5 minutes)

1. Navigate to **Assets/Scripts/Agent**
2. Right-click → **Create** → **C# Script**
3. Name it: `RobotAgent` (EXACT name!)
4. Press Enter
5. Double-click to open it
6. **DELETE ALL** existing code (Ctrl+A, Delete)
7. **PASTE** the entire RobotAgent_NEW.cs code I provided
8. **Save** (Ctrl+S)
9. Close the code editor
10. Return to Unity
11. Wait for compilation (5-10 seconds)
12. Check **Console** (bottom) - should be NO red errors

✅ **Success Check:** RobotAgent.cs exists, no errors in Console

---

Step 4.4: Create DataCollector Script (5 minutes)

1. Navigate to **Assets/Scripts/Data**
2. Right-click → Create → C# Script
3. Name: `DataCollector`
4. Double-click to open
5. Delete all existing code
6. Paste the entire DataCollector_NEW.cs code
7. Save (Ctrl+S)
8. Close editor
9. Return to Unity

10. Wait for compilation

✅ **Success Check:** DataCollector.cs exists, no errors

Step 4.5: Create PerformanceTracker Script (5 minutes)

This tracks which robot performs best!

1. Navigate to **Assets/Scripts/Environment**
2. Right-click → Create → C# Script
3. Name: `PerformanceTracker`
4. Double-click to open
5. Delete all existing code
6. Paste the PerformanceTracker.cs code I provided
7. Save
8. Close editor
9. Return to Unity
10. Wait for compilation

✅ **Success Check:** PerformanceTracker.cs exists, no errors

Step 4.6: Create TrainingAreaSpawner Script (5 minutes)

This creates 100 copies of your training area!

1. Still in **Assets/Scripts/Environment**
2. Right-click → Create → C# Script
3. Name: `TrainingAreaSpawner`
4. Double-click to open
5. Delete all existing code
6. Paste the TrainingAreaSpawner.cs code
7. Save
8. Close editor
9. Return to Unity
10. Wait for compilation

✅ **Success Check:** TrainingAreaSpawner.cs exists, no errors, ALL 4 scripts compiled!

Step 4.7: Attach RobotAgent to RobotArm (15 minutes)

Now we connect the brain to the body!

1. In Hierarchy, expand **TrainingArea_0**
2. Click on **RobotArm** (inside TrainingArea_0)
3. Look at Inspector (right side)
4. At bottom of Inspector, click **Add Component**
5. Type: `RobotAgent`
6. Click on **RobotAgent (Script)**

You'll see LOTS of empty fields! Fill them ALL:

Robot Joint Components:

7. **Base Rotation** field:
   - In Hierarchy, expand RobotArm → Drag **BaseRotation** to this field
8. **Femur Joint** field:
   - Navigate to BaseRotation → Femur → Drag **FemurJoint** here
9. **Knee Joint** field:
   - Navigate to FemurJoint → Shin → Drag **KneeJoint** here
10. **Foot Magnet** field:
    - Navigate to KneeJoint → Drag **FootMagnet** here

Environment Objects:

11. **Movable Box**: Drag **MovableBox** (from TrainingArea_0)
12. **Target Zone A**: Drag **TargetZoneA**
13. **Target Zone B**: Drag **TargetZoneB**
14. **Floor**: Drag **Floor**

Magnet Settings:

15. **Magnetic Range:** `0.5`
16. **Magnetic Strength:** `100`

17. **Visualize Magnet Range:** ✅ CHECK

Training Parameters:

18. **Max Motor Force:** `100`
19. **Movement Speed:** `50`
20. **Reward Multiplier:** `1`

Physics Data Collection:

21. **Collect Detailed Physics:** ✅ CHECK

🆕 **Power Budget System:**

22. **Use Power Budget:** ✅ CHECK
23. **Max Power Budget:** `100`

🆕 **Random Position System:**

24. **Use Random Positions:** ✅ CHECK
25. **Min Distance:** `2`
26. **Max Reach:** `4`
27. **Workspace Radius:** `3.5`

✅ **Success Check:** ALL fields filled, NO "None" anywhere!

Step 4.8: Attach DataCollector (10 minutes)

1. With **RobotArm** still selected
2. Add Component → `DataCollector`

Configure: 3. **Collect Data:** ✅ CHECK 4. **Data Directory:** `TrainingData` 5. **Max Episodes To Record:** `1000` 6. **Auto Export On Interval:** ✅ CHECK 7. **Export Interval:** `50` 8. **Export Detailed Physics:** ❌ **UNCHECK** (will be enabled dynamically by PerformanceTracker!) 9. **Export Inverse Kinematics:** ✅ CHECK 10. **Export Statistical Summary:** ✅ CHECK 11. **File Prefix:** `aura_robot`

✅ **Success Check:** DataCollector configured (detailed physics UNCHECKED!)

Step 4.9: Add Behavior Parameters (10 minutes)

This tells Unity this is an AI agent!

1. With **RobotArm** selected
2. Add Component → Behavior Parameters

Configure (EXACT values!): 3. **Behavior Name:** RobotArm (EXACT! Capital R, capital A!) 4. **Vector Observation → Space Size:** 30 5. **Actions → Continuous Actions:** 3 6. **Discrete Branches:** 0 7. **Model:** Leave empty 8. **Behavior Type:** Default

✅ **Success Check:** Behavior Name = "RobotArm", 30 observations, 3 actions

---

Step 4.10: Add Decision Requester (3 minutes)

1. With **RobotArm** selected
2. Add Component → Decision Requester

Configure: 3. **Decision Period:** 5 4. **Take Actions Between Decisions:** ✅ CHECK

✅ **Success Check:** Decision Requester added with period 5

Step 4.11: Final Component Verification (5 minutes)

RobotArm should have these 4 components:

1. ✅ Transform (always there)
2. ✅ **RobotAgent (Script)** - All fields filled
3. ✅ **DataCollector (Script)** - Configured (detailed physics OFF)
4. ✅ **Behavior Parameters** - "RobotArm", 30 obs, 3 actions
5. ✅ **Decision Requester** - Period 5

If anything missing, add it now!

Step 4.12: Test ONE Robot (Important!) (5 minutes)

Before making 100 copies, test that one works!

1. Press **Play** button (▶️ at top)
2. You should see:

- Robot moves (might be random at first)
- Green/Red zones might move (random positions!)
- Box might move
- Top-left shows stats
- NO errors in console

3. Watch for "Box attached!" message when foot touches box
4. Press **Stop** (⏹️) after 10-20 seconds

If you see errors:

- Check all script references are set
- Make sure FootMagnet has "Is Trigger" checked
- Verify Behavior Name = "RobotArm" exactly

✅ **Success Check:** Robot moves, no errors, zones randomize positions!

Step 4.13: SAVE EVERYTHING! (1 minute)

Ctrl + S or **Cmd + S**

🎉 CHECKPOINT: Single Robot Working!

Your robot has:

- AI brain (RobotAgent) ✅
- Power budget system ✅
- Random position generation ✅
- Data collector ✅
- All components configured ✅

Next: Create 100 copies for parallel training! 🚀

CREATE 100 TRAINING ENVIRONMENTS

Step 5.1: Create Prefab from TrainingArea (5 minutes)

What's a Prefab? A reusable template we can duplicate!

1. In **Project** window, navigate to **Assets/Prefabs**
2. In **Hierarchy**, select **TrainingArea_0** (the parent of everything)
3. **Drag TrainingArea_0** from Hierarchy into the Prefabs folder
4. You now have a blue prefab icon!
5. The original in Hierarchy turns blue (it's now connected to prefab)

✅ **Success Check:** TrainingArea_0 prefab exists in Assets/Prefabs folder

---

Step 5.2: Delete Original Training Area (2 minutes)

We'll let the spawner create them all!

1. In Hierarchy, select **TrainingArea_0**
2. Press **Delete** key
3. Confirm deletion
4. Your scene should now only have Camera and Light!

This is OK! The spawner will recreate everything!

✅ **Success Check:** Hierarchy only has Main Camera and Directional Light

---

Step 5.3: Create Training Manager (10 minutes)

1. In Hierarchy, right-click → **Create Empty**
2. Name: `TrainingManager`
3. Position: (0, 0, 0)

Add Spawner Script: 4. Select TrainingManager 5. Add Component → `TrainingAreaSpawner`

Configure Spawner: 6. **Training Area Prefab:**

- Click the circle next to this field
- Select **TrainingArea_0** from the list
- OR drag TrainingArea_0 from Prefabs folder

7. **Number Of Areas:** `100` (or start with 25 if computer is slow)
8. **Spacing:** `15`

9. **Areas Per Row:** `10` (makes a 10x10 grid)

✅ **Success Check:** TrainingManager has spawner with TrainingArea_0 prefab assigned

---

Step 5.4: Test the Spawner! (5 minutes)

Let's see if it works!

1. Press **Play** (▶️)
2. Wait a few seconds
3. Check Console - should see: "Creating 100 training areas..."
4. Then: "Successfully spawned 100 areas!"
5. In Scene view, zoom out (scroll mouse wheel)
6. You should see a GRID of 100 robot arms!

Navigate the view:

- Hold **Right Mouse Button** and move mouse to look around
- Use **WASD** keys to fly around
- Scroll wheel to zoom

7. Press **Stop** (⏹️) when done looking

If you only see a few robots:

- Camera might be too close, zoom out more!

If you see errors:

- Check TrainingArea_0 prefab is assigned in spawner
- Verify all scripts compiled without errors

✅ **Success Check:** 100 training areas spawned in a 10x10 grid!

Step 5.5: Adjust Camera for 100 Robots (5 minutes)

The camera needs to see more!

1. Select **Main Camera**

2. Inspector → Transform:
    - Position X: `75` (center of 10x10 grid)
    - Position Y: `100` (high up!)
    - Position Z: `75`
    - Rotation X: `90` (looking straight down)
    - Rotation Y: `0`
    - Rotation Z: `0`
3. Camera component → **Field of View:** `90` (wider angle)
4. Press Play to test - you should see most/all robots from above!

✅ **Success Check:** Can see the full 10x10 grid of robots from above

---

Step 5.6: SAVE! (1 minute)

Ctrl + S

🎉 CHECKPOINT: 100 Parallel Training Environments Ready!

- 100 robot arms ✅
- Each with own floor, zones, box ✅
- All configured identically ✅
- PerformanceTracker will monitor them all ✅
- Only best performer will collect detailed data ✅

Next: Python setup and TRAINING! 🐍

PYTHON SETUP

Step 6.1: Open Command Prompt/Terminal (2 minutes)

Windows:

1. Press **Windows key + R**
2. Type: `cmd`
3. Press Enter

Mac:

1. Press **Cmd + Space**
2. Type: `terminal`
3. Press Enter

✅ **Success Check:** Command prompt/terminal is open

Step 6.2: Navigate to Your Project (5 minutes)

Find your project folder:

1. Open File Explorer (Windows) or Finder (Mac)
2. Navigate to where you created the project
3. Usually: `Documents/AURA-Robot-Sim`
4. Copy the full path

In terminal, navigate there:

Windows:

```bash
cd C:\Users\YourName\Documents\AURA-Robot-Sim
```

Mac:

```bash
cd /Users/YourName/Documents/AURA-Robot-Sim
```

Easier method:

- Type `cd ` (with space)
- Drag the AURA-Robot-Sim folder into terminal
- Path pastes automatically!
- Press Enter

Verify you're there:

```bash
```

```
dir     # Windows
ls      # Mac
```

Should see: Assets, Library, ProjectSettings folders

✅ **Success Check:** You're in the AURA-Robot-Sim folder

Step 6.3: Create Virtual Environment (5 minutes)

```bash
python -m venv mlagents-env
```

Press Enter, wait 30-60 seconds.

✅ **Success Check:** Folder `mlagents-env` appears in your project

Step 6.4: Activate Virtual Environment (2 minutes)

Windows:

```bash
mlagents-env\Scripts\activate
```

Mac/Linux:

```bash
source mlagents-env/bin/activate
```

You'll know it worked when:

- `(mlagents-env)` appears before your path

✅ **Success Check:** `(mlagents-env)` is visible

Step 6.5: Install PyTorch (5 minutes)

```bash
```

```
pip install torch torchvision torchaudio
```

Wait 2-5 minutes. Lots of text will scroll.

✅ **Success Check:** "Successfully installed torch..." at the end

---

Step 6.6: Install ML-Agents (5 minutes)

```bash
pip install mlagents==0.30.0
```

Wait 1-2 minutes.

✅ **Success Check:** "Successfully installed mlagents-0.30.0"

Step 6.7: Verify Installation (2 minutes)

```bash
mlagents-learn --help
```

Should show ML-Agents help text!

✅ **Success Check:** Help text appears, no errors

Step 6.8: Create Config Folder & File (5 minutes)

1. In your project folder (AURA-Robot-Sim), create folder: `config`
2. Inside config, create file: `RobotArm_config.yaml`
3. Open it in a text editor
4. Paste this configuration:

```yaml
```

```yaml
behaviors:
  RobotArm:
    trainer_type: ppo

    hyperparameters:
      learning_rate: 0.0003
      learning_rate_schedule: linear
      batch_size: 2048
      buffer_size: 20480
      num_epoch: 3
      gamma: 0.99
      lambd: 0.95
      beta: 0.005
      epsilon: 0.2

    network_settings:
      hidden_units: 256
      num_layers: 3
      normalize: true
      vis_encode_type: simple

    reward_signals:
      extrinsic:
        strength: 1.0
        gamma: 0.99
      curiosity:
        strength: 0.01
        gamma: 0.99
        encoding_size: 256

    max_steps: 3000000
    time_horizon: 64
    summary_freq: 10000
    keep_checkpoints: 5
    checkpoint_interval: 50000

engine_settings:
  width: 84
  height: 84
  quality_level: 1
  time_scale: 20
```

```
target_frame_rate: -1
capture_frame_rate: 60
```

5. Save the file

Your structure:

```
AURA-Robot-Sim/
├── Assets/
├── config/
│   └── RobotArm_config.yaml
├── mlagents-env/
└── ... other folders
```

✅ **Success Check:** RobotArm_config.yaml exists in config folder

---

🎉 **CHECKPOINT: Python Ready!**

- Virtual environment ✅
- PyTorch installed ✅
- ML-Agents installed ✅
- Config file ready ✅

Next: TRAIN 100 ROBOTS! 🚀

TRAINING THE ROBOT

This is it! 100 robots learning simultaneously!

Step 7.1: Final Unity Setup (5 minutes)

1. Return to Unity
2. Make sure scene is saved
3. **Don't press Play yet!**

Optional: Reduce visual load 4. Select Main Camera 5. Camera component → **Target Display:** Display 1 6. Can also reduce Quality: Edit → Project Settings → Quality → Set to "Low"

✅ **Success Check:** Unity ready, not in Play mode

Step 7.2: Start Training Command (5 minutes)

In your terminal (make sure mlagents-env is activated!):

For 100 environments:

```bash
mlagents-learn config/RobotArm_config.yaml --run-id=RobotArm_100x_v1 --num-envs=100
```

If your computer is slower, use fewer:

```bash
mlagents-learn config/RobotArm_config.yaml --run-id=RobotArm_25x_v1 --num-envs=25
```

Press Enter.

What you'll see:

1. Text appears
2. Some warnings (yellow - that's OK!)
3. Configuration summary
4. Eventually: **"Start training by pressing the Play button in the Unity Editor"**

✅ **Success Check:** Message says to press Play button

Step 7.3: Press Play in Unity! (1 minute)

1. Switch to Unity
2. Click **Play** button (▶️) at top
3. Scene changes to Game view
4. ALL 100 robots start moving!
5. Terminal shows: "Connected to Unity environment"
6. Steps start counting: "Step: 100", "Step: 200", etc.

What you'll see:

- 100 robots moving (looks chaotic at first!)

- Boxes moving
- Zones randomly positioned
- Stats in top corners
- Terminal showing progress

✅ **Success Check:**

- All robots moving
- Terminal shows increasing step count
- No errors
- PerformanceTracker shows "Best Performer" in Unity

Step 7.4: Understanding Training with 100 Robots (10 minutes)

Your terminal will show:

```
Step: 1000. Time Elapsed: 12.3 s. Mean Reward: -0.234
Step: 2000. Time Elapsed: 24.6 s. Mean Reward: -0.156
...
```

With 100 robots:

- Step 1000 = 10 steps per robot on average
- They're all learning from different random positions!
- Some robots might succeed while others fail
- This is GOOD - more diverse learning!

In Unity (top-right), you'll see:

```
=== PERFORMANCE TRACKER ===
Best Performer: TrainingArea_42/RobotArm
Success Rate: 12.5%
Avg Time: 18.45s
Score: 23.67
```

This means:

- Robot #42 is currently performing best
- Only that robot collects detailed physics data

- Switches as robots improve!

Training Stages (Much Faster Now!):

Steps	Time	What's Happening
0-10k	2 min	Random flailing, exploring
10k-50k	10 min	Learning to move joints
50k-150k	30 min	Learning magnetic pickup
150k-500k	1.5 hr	Improving success rate
500k-1M	3 hr	Optimizing efficiency
1M+	5+ hr	Peak performance

With 100 robots, you can reach 1M steps in ~3 hours instead of ~30 hours!

✅ **Success Check:** Training progressing, step count increasing

---

Step 7.5: Monitor with TensorBoard (Optional) (10 minutes)

Want graphs?

1. Open **NEW** terminal/command prompt
2. Navigate to project folder
3. **Don't activate virtual environment**
4. Type:

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

5. Open browser → `http://localhost:6006`

You'll see graphs:

- Environment/Cumulative Reward (should increase!)
- Environment/Episode Length

- Losses/Policy Loss
- And more!

✅ **Success Check:** Graphs visible in browser

Step 7.6: How Long to Train? (Information)

With 100 robots:

Target	Steps	Time	Success Rate
Minimum	500k	1.5 hr	~40-60%
Good	1M	3 hr	~60-80%
Excellent	2M	6 hr	~80-90%
Peak	3M	9 hr	~90-95%

My recommendation:

- Let it run to **1M steps** (about 3 hours)
- Check success rate in PerformanceTracker
- If >70%, you can stop
- If <70%, let it continue

You can leave it running overnight!

Step 7.7: Watching Training (Tips)

In Unity:

- Can zoom in on one robot to watch closely
- Best Performer has green glowing foot!
- Watch for "Box attached!" messages
- Success rate updates every episode

Console messages:

```
[RobotAgent] Box attached! Episode 234
[PerformanceTracker] NEW BEST PERFORMER: TrainingArea_67 (Score: 45.23)
[DataCollector] TrainingArea_67/RobotArm - NOW COLLECTING detailed physics
```

✅ **Success Check:** Can see robots learning, improving over time

---

Step 7.8: Stopping Training (2 minutes)

When you want to stop:

1. Go to terminal with ML-Agents
2. Press **Ctrl + C**
3. Wait for it to save (10-20 seconds)
4. You'll see: "Exported .onnx file"
5. In Unity, click **Stop** button (⏹️)

Your trained model saved to:

```
AURA-Robot-Sim/results/RobotArm_100x_v1/RobotArm.onnx
```

✅ **Success Check:** Terminal shows export complete, .onnx file exists

---

Step 7.9: Resuming Training (If Needed)

If you stopped and want to continue:

```bash
mlagents-learn config/RobotArm_config.yaml --run-id=RobotArm_100x_v1 --num-envs=100 --resume
```

Note the `--resume` flag!

🎉 CHECKPOINT: Training Complete!

What you accomplished:

- Trained 100 robots simultaneously ✅
- AI learning from diverse scenarios ✅

- Power budget forcing efficiency ✅
- Random positions ensuring generalization ✅
- Best performer auto-tracked ✅
- Detailed data collected from optimal behavior ✅

Next: Test your trained robots! 🧪

TESTING AND DATA ANALYSIS

Step 8.1: Delete Training Environments (2 minutes)

We only want ONE robot for testing!

1. In Unity (not in Play mode!)
2. In Hierarchy, select **TrainingManager**
3. Press **Delete**
4. All 100 training areas disappear!

Step 8.2: Create Single Test Robot (5 minutes)

1. In Project → Prefabs, drag **TrainingArea_0** into Hierarchy
2. Position it at (0, 0, 0)
3. Rename to `TestArea`
4. Select Main Camera
5. Reset camera position:
    - Position: (0, 10, -10)
    - Rotation: (45, 0, 0)

✅ **Success Check:** One robot visible in nice camera angle

Step 8.3: Load Your Trained Model (5 minutes)

1. Select **TestArea** → **RobotArm** in Hierarchy
2. In Inspector, find **Behavior Parameters** component
3. **Model** field (currently empty):

- Click the circle next to it
- Navigate to: `results → RobotArm_100x_v1`
- Select **RobotArm.onnx**
- Click **Select**

4. **Behavior Type:** Change to **Inference Only**

✅ **Success Check:** Model shows "RobotArm", type is "Inference Only"

Step 8.4: Enable Data Collection for Testing (3 minutes)

1. Still on TestArea/RobotArm
2. DataCollector component:
   - ✅ **Export Detailed Physics:** CHECK (for testing)
   - ✅ **Collect Data:** CHECK

✅ **Success Check:** Data collection enabled

Step 8.5: Test Your Trained Robot! (10 minutes)

The moment of truth!

1. Press **Play** (▶️)
2. Watch your trained robot!

What you should see:

- Robot moves purposefully (not random!)
- Rotates base toward random box position
- Extends arm smoothly
- When foot approaches: **"Box attached!"**
- Lifts box
- Rotates toward random target
- Places box near target
- **SUCCESS!**

Try multiple episodes: 3. Let it run for 5-10 episodes 4. Watch success rate in top-left 5. Notice how it handles different random positions!

Check the stats:

```
=== Robot Status ===
Episode: 10
Success Rate: 80.00% ← Hopefully high!
Box Attached: YES
Distance to Target: 0.23m
Energy Used: 45.67
Power Remaining: 54.3% ← Using power budget efficiently!
```

✅ **Success Check:** Robot successfully moves box >70% of the time from random positions!

Step 8.6: Manual Control Test (Optional) (5 minutes)

Want to drive it yourself?

1. Stop Play mode
2. RobotArm → Behavior Parameters → Behavior Type: **Heuristic Only**
3. Press Play

Controls:

- **Q/E**: Rotate base left/right
- **W/S**: Femur up/down
- **A/D**: Knee extend/contract

Try to:

- Move foot to box (wherever it randomly spawned!)
- Watch for magnetic pickup
- Move box to target

✅ **Success Check:** You can control robot with keyboard

Step 8.7: Check Your Data Files! (10 minutes)

1. In File Explorer/Finder, navigate to your project

2. Open folder: **TrainingData**

You should see 5 files:

1. aura_robot_basic_[timestamp].csv

- Episode-by-episode data from ALL 100 robots
- Thousands of rows!
- Open in Excel/Google Sheets

Columns:

- Episode, Time_Taken, Accuracy, Energy_Consumed, Success, Timestamp

2. aura_robot_physics_detailed_[timestamp].csv

- Frame-by-frame data from BEST PERFORMER ONLY
- 23 columns of physics data
- Optimal behavior trajectories!

Columns:

- Joint angles, velocities, controls
- Foot position (X,Y,Z)
- Box position and velocity
- Energy per step
- Box_Attached flag

3. aura_robot_kinematics_[timestamp].csv

- Inverse kinematics data
- End effector positions
- Joint angle configurations
- Reachability data

4. aura_robot_distributions_[timestamp].csv

- Frequency bins for histograms
- Time, Energy, Accuracy distributions
- Relative and cumulative frequencies

5. aura_robot_statistics_[timestamp].txt

- Complete statistical summary
- Open in text editor

Check what's inside:

```
=== AURA Robot Training - Statistical Summary ===
Session ID: 20260203_143022
Generated: 2026-02-03 14:45:12

--- Overall Performance ---
Total Episodes: 4523
Successful: 3654
Failed: 869
Success Rate: 0.8079

--- Central Tendency Measures ---
Mean Time: 18.45 seconds
Median Time: 17.23 seconds
Std Dev: 5.67

--- Power Budget Analysis ---
Mean Power Remaining: 42.3%
(Shows robots learned to be efficient!)

... and much more!
```

✅ **Success Check:** All 5 files exist and contain data!

Step 8.8: Quick Data Analysis (10 minutes)

Open basic CSV in Excel/Sheets:

Calculate overall success rate:

```
=AVERAGE(Success_Column) * 100
```

Average time for successful episodes:

```
=AVERAGEIF(Success_Column, 1, Time_Column)
```

Average power remaining (efficiency!):

```
=AVERAGE(Power_Remaining_Column)
```

Create a learning curve chart:

1. Select Episode and Success columns
2. Create rolling average (50-episode window)
3. Insert line chart
4. See improvement over time!

Analyze power budget:

- Compare energy used vs. time taken
- Are faster episodes more efficient?
- Correlation analysis!

✅ **Success Check:** You can open and analyze the CSV files!

Step 8.9: Understanding Your Data (Information)

What makes this data special:

From 100 robots:

- 🎯 Massive sample size (4000+ episodes!)
- 📊 Statistical significance
- 🎲 Diverse scenarios (random positions!)
- 📈 Clear learning trends

From best performer only:

- ⭐ Optimal behavior patterns
- 🔬 High-quality physics data
- 💡 Efficient strategies
- 🎯 Best-case analysis

Power budget data:

- ⚡ Energy efficiency metrics

- 📉 Work vs. performance tradeoff
- 🎓 Perfect for mechanics analysis

Random positions:

- 🎲 Probability distributions
- 📐 Inverse kinematics variety
- 🗺️ Workspace coverage
- 🔍 Generalization proof

🎉 FINAL CHECKPOINT: PROJECT COMPLETE!

What You've Accomplished:

Unity Skills:

- ✅ Built complete 3D training environment
- ✅ Created 2-joint articulated robot
- ✅ Implemented physics simulation
- ✅ Spawned 100 parallel environments

AI/Machine Learning:

- ✅ Trained neural network from scratch
- ✅ Used reinforcement learning (PPO algorithm)
- ✅ Implemented reward shaping
- ✅ Achieved 70-90% success rate

Advanced Features:

- ✅ Power budget system (limited energy)
- ✅ Random position generation
- ✅ 100x parallel training
- ✅ Best-performer tracking
- ✅ Adaptive data collection

Data Science:

- ✅ Collected 5 comprehensive datasets

- ✅ Episode-level statistics (4000+ samples)
- ✅ Frame-level physics data
- ✅ Inverse kinematics logs
- ✅ Probability distributions
- ✅ Statistical summaries

Academic Analysis Ready:

- ✅ Probability & Statistics (distributions, correlations, hypothesis tests)
- ✅ Mechanics (forces, work, energy, trajectories, IK)
- ✅ Performance optimization (efficiency analysis)
- ✅ Machine learning (convergence, generalization)

TROUBLESHOOTING GUIDE

Problem: Unity won't install

Solution:

- Check disk space (need 5GB+)
- Try different Unity version (2021.3 or 2022.3 LTS)
- Run installer as Administrator (Windows)
- Disable antivirus temporarily

Problem: ML-Agents package won't install

Solution 1 - Git URL method:

```
com.unity.ml-agents@2.3.0-exp.3
```

Solution 2 - Manual download:

- https://github.com/Unity-Technologies/ml-agents/releases/tag/release_20
- Extract and use "Add package from disk"

If you see version 2.0.2:

- That's old! Use Git URL method to get 2.3.0

Problem: Python version wrong

Check version:

```bash
python --version
```

If 3.11 or 3.12:

1. Uninstall Python completely
2. Download Python 3.10.11 specifically
3. **Check "Add to PATH"** during install
4. Restart computer
5. Verify: `python --version`

Problem: Can't activate virtual environment

Windows specific:

```bash
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Then try activate again.

Alternative - use full path:

```bash
mlagents-env\Scripts\python -m pip install torch
mlagents-env\Scripts\python -m pip install mlagents
```

Problem: "mlagents-learn not found"

Solution:

1. Make sure virtual environment is activated (see `(mlagents-env)`)
2. Try: `python -m mlagents.trainers.learn --help`
3. If that works, use full command:

```bash
python -m mlagents.trainers.learn config/RobotArm_config.yaml --run-id=test
```

Problem: Robot falls apart / explodes

Solution:

1. Select each Articulation Body (BaseRotation, FemurJoint, KneeJoint)
2. Increase **Stiffness** to `50000`
3. Increase **Damping** to `5000`
4. Reduce **Force Limit** to `50`

Still exploding?

- In config YAML, reduce `time_scale` from 20 to 10
- Check all joint limits are correct

Problem: Magnet not picking up box

Critical checks:

1. FootMagnet → Sphere Collider → **Is Trigger:** MUST BE CHECKED ✓
2. MovableBox has Rigidbody
3. Magnetic Range = 0.5 (try 0.7 if still not working)
4. Check console for "Box attached!" messages

If never attaching:

- Verify FootMagnet has MagnetTrigger script (added automatically)
- Check FootMagnet is child of KneeJoint
- Make sure box has Rigidbody (not just collider)

Problem: No learning progress (success rate stays 0%)

Check these in order:

1. Behavior Name:

- RobotArm → Behavior Parameters
- Name MUST be exactly: `RobotArm` (capital R, capital A)

2. Observations:

- Vector Observation Space Size = **30** (not 20!)

3. All references set:

- Click RobotArm → RobotAgent component
- NO field should say "None"
- All joints, zones, box, floor must be assigned

4. Config file location:

- Must be in: `AURA-Robot-Sim/config/RobotArm_config.yaml`
- Behavior name in YAML matches Unity: "RobotArm"

5. Increase learning rate:

- Edit YAML: `learning_rate: 0.001` (increase from 0.0003)

Problem: Training very slow / computer freezing

Solutions:

1. Reduce number of environments:

```bash
--num-envs=25  # Instead of 100
```

2. Use --no-graphics flag:

```bash
```

```
mlagents-learn config/RobotArm_config.yaml --run-id=test --num-envs=100 --no-graphics
```

(Can't watch training, but MUCH faster!)

3. Reduce time_scale in YAML:

```yaml
time_scale: 10  # Instead of 20
```

4. Lower quality:

- Unity: Edit → Project Settings → Quality → "Low"

5. Close other programs:

- Web browsers
- Video players
- Other Unity instances

Problem: Data files empty / not created

Solutions:

1. Check DataCollector:

- Select RobotArm
- DataCollector → **Collect Data:** MUST BE CHECKED ✓

2. Check TrainingData folder exists:

- Should be in: `AURA-Robot-Sim/TrainingData/`
- Create it manually if missing

3. Manual export:

- Right-click DataCollector component
- Select "Export Data Now"
- Check if files appear

4. Check console for errors:

- Any red messages about file writing?
- Might be permissions issue

Problem: Only getting basic CSV, not detailed physics

This is CORRECT if using PerformanceTracker!

Why:

- Only the BEST performer collects detailed physics
- Reduces file size
- Better quality data

To verify it's working:

- Check Unity GUI (top-right) for "Best Performer: TrainingArea_X"
- That robot should have detailed data
- Console should show: "NOW COLLECTING detailed physics"

To collect from all robots (not recommended):

- On each RobotArm → DataCollector
- Check "Export Detailed Physics"
- Warning: Creates HUGE files!

Problem: Random positions not working

Check:

1. RobotAgent → **Use Random Positions:** CHECKED ✓
2. Press Play - watch zones move to different positions each episode
3. Console should show: "New positions - Start: (x,y,z), End: (x,y,z)"

If zones don't move:

- Check "Use Random Positions" is checked
- Verify Min Distance and Workspace Radius values are set
- Look for errors in console

Problem: Power budget not working

Check:

1. RobotAgent → **Use Power Budget:** CHECKED ✓
2. **Max Power Budget:** Should be number like 100
3. During play, watch GUI: Should show "Power Remaining: X%"

If not seeing power:

- Check "Use Power Budget" is checked
- Verify currentPower variable exists in script
- Console might show "Out of power" messages when depleted

Problem: Performance Tracker not showing

Solutions:

1. Check it exists:

- Play mode → Look for "PERFORMANCE TRACKER" in top-right
- If missing, PerformanceTracker script not running

2. Spawner should create it:

- TrainingAreaSpawner script auto-creates PerformanceTracker
- Check console: "[PerformanceTracker] Registered X robots"

3. Manual creation:

- Create Empty GameObject: "PerformanceTracker"
- Add Component → PerformanceTracker script

Problem: "Assertion failed" errors

These are Unity bugs - IGNORE THEM!

What they are:

- Internal Unity editor warnings
- Don't affect your project
- Don't break anything
- Common in Unity 2022.3

What to do:

- Nothing! Just ignore them
- Can clear console with "Clear" button
- They don't mean your code is wrong

Problem: Training interrupted / computer crashed

Resume training:

```bash
mlagents-learn config/RobotArm_config.yaml --run-id=RobotArm_100x_v1 --num-envs=100 --resume
```

If that doesn't work:

- Start new run with different run-id
- Previous checkpoints still saved in results folder

Problem: WebGL build fails

Solutions:

1. Switch scripting backend:

- File → Build Settings → Player Settings
- Other Settings → Scripting Backend → IL2CPP

2. Reduce quality:

- Edit → Project Settings → Quality → "Low"

3. Smaller builds:

- Build Settings → Compression: Gzip

4. Try simpler scene:

- Test with just 1 robot, not 100

Problem: Can't find trained model

Location:

```
AURA-Robot-Sim/results/[your-run-id]/RobotArm.onnx
```

If not there:

- Training might not have saved properly
- Check if results folder exists
- Look in all run-id folders
- Try training again for at least 50k steps

NEXT STEPS - BEYOND THE TUTORIAL

🎓 For Your Academic Project

Probability & Statistics Analysis:

1. **Descriptive Statistics:**
    - Calculate mean, median, mode of completion times
    - Standard deviation, variance
    - Create box plots
    - Identify outliers
2. **Probability Distributions:**
    - Fit normal distribution to time data
    - Test goodness-of-fit (Chi-square test)
    - Create histograms from frequency data
    - Calculate probability of success given distance
3. **Hypothesis Testing:**

- T-test: Compare successful vs. failed episodes
- ANOVA: Compare performance across different distance ranges
- Correlation: Time vs. Energy, Energy vs. Accuracy

4. **Confidence Intervals:**
    - 95% CI for mean completion time
    - 95% CI for success rate
    - Bootstrap analysis

Mechanics Analysis:

1. **Kinematics:**
    - Forward kinematics verification
    - Inverse kinematics calculations
    - Workspace envelope mapping
    - Velocity and acceleration profiles
2. **Dynamics:**
    - Calculate torques from control inputs
    - Energy analysis (kinetic + potential)
    - Work done: W = F × d
    - Power consumption over time
3. **Forces:**
    - Magnetic force estimation
    - Joint loads during movement
    - Box acceleration analysis
    - Impact forces
4. **Optimization:**
    - Find minimum-energy paths
    - Analyze trajectory efficiency
    - Compare optimal vs. actual paths

🐍 Python Analysis Script

Create: `analyze_aura_data.py`

python

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import seaborn as sns

# Load data
df_basic = pd.read_csv('TrainingData/aura_robot_basic_YYYYMMDD_HHMMSS.csv')
df_physics = pd.read_csv('TrainingData/aura_robot_physics_detailed_YYYYMMDD_HHMMSS.csv')
df_kinematics = pd.read_csv('TrainingData/aura_robot_kinematics_YYYYMMDD_HHMMSS.csv')

# === STATISTICS ===
print("=== Overall Statistics ===")
print(f"Total Episodes: {len(df_basic)}")
print(f"Success Rate: {df_basic['Success'].mean() * 100:.2f}%")
print(f"Mean Time: {df_basic['Time_Taken'].mean():.2f}s")
print(f"Std Dev Time: {df_basic['Time_Taken'].std():.2f}s")
print(f"Mean Energy: {df_basic['Energy_Consumed'].mean():.2f}")

# === LEARNING CURVE ===
plt.figure(figsize=(12, 6))
rolling_success = df_basic['Success'].rolling(window=50).mean() * 100
plt.plot(rolling_success)
plt.title('Learning Curve - Success Rate Over Episodes (50-ep Moving Avg)')
plt.xlabel('Episode')
plt.ylabel('Success Rate (%)')
plt.grid(True)
plt.savefig('learning_curve.png')
plt.show()

# === TIME DISTRIBUTION ===
plt.figure(figsize=(10, 6))
plt.hist(df_basic['Time_Taken'], bins=30, edgecolor='black', alpha=0.7)
plt.title('Distribution of Completion Times')
plt.xlabel('Time (seconds)')
plt.ylabel('Frequency')
plt.axvline(df_basic['Time_Taken'].mean(), color='red', linestyle='--', label='Mean')
plt.axvline(df_basic['Time_Taken'].median(), color='green', linestyle='--', label='Median')
plt.legend()
plt.savefig('time_distribution.png')
plt.show()

# === CORRELATION ANALYSIS ===
```

```python
successful = df_basic[df_basic['Success'] == 1]
corr_time_energy = successful['Time_Taken'].corr(successful['Energy_Consumed'])
print(f"\nCorrelation (Time vs Energy): {corr_time_energy:.3f}")

plt.figure(figsize=(10, 6))
plt.scatter(successful['Time_Taken'], successful['Energy_Consumed'], alpha=0.5)
plt.title('Time vs Energy Consumption (Successful Episodes)')
plt.xlabel('Time (seconds)')
plt.ylabel('Energy Consumed')
plt.savefig('time_vs_energy.png')
plt.show()

# === INVERSE KINEMATICS ANALYSIS ===
plt.figure(figsize=(10, 10))
plt.scatter(df_kinematics['End_Effector_X'], df_kinematics['End_Effector_Z'],
           c=df_kinematics['Reach_Distance'], cmap='viridis', alpha=0.3)
plt.colorbar(label='Reach Distance')
plt.title('Robot Workspace - End Effector Positions')
plt.xlabel('X Position (m)')
plt.ylabel('Z Position (m)')
plt.axis('equal')
plt.grid(True)
plt.savefig('workspace_map.png')
plt.show()

# === HYPOTHESIS TEST ===
# H0: Mean time for successful vs failed is the same
failed = df_basic[df_basic['Success'] == 0]
t_stat, p_value = stats.ttest_ind(successful['Time_Taken'], failed['Time_Taken'])
print(f"\nT-Test (Successful vs Failed Times):")
print(f"  t-statistic: {t_stat:.3f}")
print(f"  p-value: {p_value:.4f}")
if p_value < 0.05:
    print("  Result: Significant difference!")
else:
    print("  Result: No significant difference")

# === POWER BUDGET ANALYSIS ===
if 'Power_Remaining' in df_basic.columns:
    plt.figure(figsize=(10, 6))
    plt.hist(df_basic['Power_Remaining'], bins=30, edgecolor='black', alpha=0.7)
    plt.title('Power Efficiency - Remaining Power Distribution')
    plt.xlabel('Power Remaining (%)')
    plt.ylabel('Frequency')
```

```
    plt.savefig('power_efficiency.png')
    plt.show()

print("\n=== Analysis complete! Check generated PNG files! ===")
```

Run with:

```bash
pip install pandas matplotlib numpy scipy seaborn
python analyze_aura_data.py
```

🚀 Improvements & Extensions

1. Add More Joints:

- 3-joint arm (elbow + wrist)
- Gripper instead of magnet
- More realistic robot

2. More Complex Tasks:

- Stack boxes
- Sort by color
- Place in specific orientations

3. Obstacles:

- Add walls
- Forbidden zones
- Multiple boxes

4. Vision-Based:

- Add camera observation
- Detect box with computer vision
- Learn from pixels

5. Multi-Agent:

- 2 robots cooperating
- Pass box between them
- Collaborative tasks

📊 Create Academic Report

Sections to include:

1. **Introduction**
   - Project goals
   - Why reinforcement learning
   - Why 100 parallel environments
2. **Methodology**
   - Robot design (2-joint + rotation)
   - Power budget system
   - Random position generation
   - PPO algorithm
   - Training setup
3. **Results**
   - Learning curves
   - Success rate analysis
   - Time/energy statistics
   - Workspace analysis
4. **Statistical Analysis**
   - Descriptive statistics
   - Distributions
   - Hypothesis tests
   - Correlations
5. **Mechanics Analysis**
   - Kinematics calculations
   - Inverse kinematics verification
   - Energy and work analysis

- Trajectory optimization

6. **Discussion**
    - What worked well
    - Challenges faced
    - Power budget impact
    - Random positions impact
7. **Conclusion**
    - Key findings
    - AI learned efficient strategies
    - Statistical significance
    - Future work

🎯 Demonstration Ideas

For Presentation:

1. **Live Demo:**
    - Show trained robot
    - Random positions each time
    - Highlight success rate
2. **Video Recording:**
    - Record training progress
    - Time-lapse of learning
    - Before/after comparison
3. **Visualizations:**
    - Learning curves
    - Heatmap of workspace
    - Energy efficiency plots
    - Trajectory animations
4. **Interactive:**
    - Let audience try manual control
    - Compare human vs. AI
    - Show different random scenarios

🏆 What You've Mastered

Technical Skills:

- ✅ Unity 3D development
- ✅ Physics simulation
- ✅ Reinforcement learning (PPO)
- ✅ Python data analysis
- ✅ Statistical methods
- ✅ Inverse kinematics
- ✅ System optimization

Concepts:

- ✅ Machine learning
- ✅ Neural networks
- ✅ Reward shaping
- ✅ Parallel training
- ✅ Data collection strategies
- ✅ Performance tracking
- ✅ Probability distributions
- ✅ Mechanical analysis

Project Management:

- ✅ Breaking complex problems down
- ✅ Iterative development
- ✅ Testing and debugging
- ✅ Data-driven decision making

CONGRATULATIONS! 🎉 🎊 🏆

You Did It!

You've completed an advanced AI/robotics project that includes:

- **100 parallel robots** learning simultaneously
- **Power budget** constraints (realistic energy limits)
- **Random positions** (generalization, not memorization)
- **Smart data collection** (best performer tracking)
- **5 comprehensive datasets** ready for analysis

This is **graduate-level** work! 🎓

Your Achievement:

Built:

- ✅ Full 3D simulation environment
- ✅ 2-joint robotic arm with physics
- ✅ Magnetic pickup system
- ✅ 100 parallel training areas
- ✅ Performance tracking system

Trained:

- ✅ AI from complete scratch
- ✅ Reinforcement learning (PPO)
- ✅ 70-95% success rate
- ✅ Generalized behavior (any position!)
- ✅ Energy-efficient strategies

Collected:

- ✅ 4000+ episode samples
- ✅ Detailed physics data
- ✅ Inverse kinematics logs
- ✅ Statistical distributions
- ✅ Power usage metrics

This Demonstrates:

To Professors:

- Real AI/ML implementation

- Understanding of reinforcement learning
- Statistical analysis capability
- Mechanical engineering application
- System design skills

To Employers:

- Unity development
- Machine learning
- Python programming
- Data analysis
- Problem-solving
- Project completion

Share Your Work:

1. **GitHub:** Upload your project
2. **Portfolio:** Add screenshots and results
3. **LinkedIn:** Post about your achievement
4. **Resume:** "Developed 100-agent parallel RL system"

Final Tips:

1. **Save your trained models** - They took hours to make!
2. **Keep your data** - Unique research dataset!
3. **Document your findings** - Write that report!
4. **Take screenshots** - For presentations!
5. **Backup everything** - Don't lose your work!

Where to Go From Here:

More ML-Agents:

- Unity ML-Agents examples
- Different algorithms (SAC, MA-POCA)

- Multi-agent scenarios

More Robotics:

- ROS (Robot Operating System)
- Real robot simulation
- Computer vision integration

More Data Science:

- Advanced statistics
- Machine learning theory
- Deep learning courses

You're now officially an AI developer! 🤖 ✨

Questions? Review the relevant section!

Want to go further? The sky's the limit!

Most importantly: Be proud of what you've accomplished!

This was a challenging, real-world project and you COMPLETED IT! 🌟

Thank you for following this guide! Now go show the world what you built! 🚀