Hide-and-Seek Agent Training with Unity ML-Agents

Step-by-step build, training, and export guide

Student: Zakria (Matricola: 512705)
Course: Applied Reinforcement Learning
Professor: Alberto Castellini
Date: January 16, 2026

1. Objective

The goal of this project is to train a reinforcement learning agent in a Hide-and-Seek environment using Unity ML-Agents. Training is performed locally on a Windows machine, producing an exported ONNX policy that can be executed inside Unity (inference mode) without Python.

2. Requirements and Versions

Component	Version / Notes
Unity Editor	2022.3.62f3 (LTS)
Unity ML-Agents (C#)	2.3.0-exp.2 (package detected at runtime)
Python (Conda env)	3.8.18
mlagents	0.30.0
mlagents_envs	0.30.0
PyTorch	1.7.1 + CUDA 11.0 build
protobuf	3.20.3 (required for ML-Agents 0.30.0)

Note: A dedicated Conda environment was used to isolate dependencies and avoid version conflicts.

3. Project and Scenes

The Hide-and-Seek environment is provided as a standalone Unity project (unity-ml-agents_hide-and-seek). Two scenes are important for this workflow:

- Training scene: Assets/Scenes/Training.unity
- Test (inference) scene: Assets/Scenes/Test.unity

4. Python Environment Setup

A Conda environment was created and activated before installing ML-Agents. The following steps were executed from the terminal.

```
conda create -n ml_agents python=3.8.18 pip -y
conda activate ml_agents
python --version
```

Package installation:

```
python -m pip install --upgrade pip setuptools wheel
pip install torch~=1.7.1 -f https://download.pytorch.org/whl/torch_stable.html
python -m pip install mlagents==0.30.0
```

4.1 Dependency Fixes

During setup, ML-Agents failed due to an incompatible protobuf version and a missing package required by TensorBoard logging. The following fixes were applied:

```
python -m pip uninstall -y protobuf
python -m pip install protobuf==3.20.3
python -m pip install six

mlagents-learn --help
```

5. Building the Training Executable (Windows)

Training was performed with a built Unity executable instead of training in the Editor. Executable training is faster, supports headless mode (--no-graphics), and avoids Editor overhead.

Unity build steps:

1. Open the Hide-and-Seek Unity project.
2. File -> Build Settings...
3. Add Assets/Scenes/Training.unity to "Scenes In Build" and ensure it is checked.
4. Select "PC, Mac & Linux Standalone" -> Target: Windows, Architecture: x86_64.
5. Click Build and select an output folder: Builds/Training/

Expected build output:

```
Builds\Training\unity-ml-agents_hide-and-seek.exe
Builds\Training\unity-ml-agents_hide-and-seek_Data\
```

5.1 Build Error and Resolution

The first build attempt failed due to a corrupted Unity package cache (Visual Scripting DLL missing in PackageCache). After rebuilding and allowing Unity to refresh packages, the build completed successfully and only warnings remained.

6. Discovering Behavior Names and Specs

The trainer configuration must use the correct behavior key(s). In this environment, behavior names were extracted directly from the built executable using the ML-Agents Python API.

```
python -c "from mlagents_envs.environment import UnityEnvironment;
env=UnityEnvironment(file_name=r'Builds\Training\unity-ml-agents_hide-and-seek.exe',
no_graphics=True); env.reset(); print('Behaviors:', list(env.behavior_specs.keys()));
[print(f'\n{name}\n  action_spec: {spec.action_spec}\n  obs_shapes: {[o.shape for o in
spec.observation_specs]}')  for name,spec in env.behavior_specs.items()]; env.close()"
```

Observed behaviors (two teams):

- HideAndSeekAgent?team=0
- HideAndSeekAgent?team=1

7. Creating the Trainer YAML (ml-agents_config.yaml)

The repository did not include a trainer YAML, so a new configuration file was created in the project root. A common mistake is to use team-suffixed behavior names as YAML keys. ML-Agents expects the base behavior name (HideAndSeekAgent).

File created:

```
unity-ml-agents_hide-and-seek\ml-agents_config.yaml
```

Final YAML content used:

```
behaviors:
  HideAndSeekAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 3.0e-4
      beta: 1.0e-3
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 256
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 2000000
    time_horizon: 256
    summary_freq: 20000
    checkpoint_interval: 100000
```

8. Training Command

Training was launched from the Conda environment using mlagents-learn. Headless mode was enabled for speed (--no-graphics) and the Unity time scale was increased.

```
mlagents-learn "ml-agents_config.yaml" --env "Builds\Training\unity-ml-agents_hide-and-seek.exe" --run-id hns_run1 --force --no-graphics --time-scale 20
```

Successful startup indicators:

- Connected new brain: HideAndSeekAgent?team=0
- Connected new brain: HideAndSeekAgent?team=1
- HideAndSeekAgent. Step: ... Training.

8.1 Notes on Training Warnings

During training, ML-Agents reported that multiple teams were present and suggested enabling self-play for adversarial games. This run (hns_run1) was treated as a baseline training run.

9. Exported Models and Results Directory

ML-Agents periodically exports ONNX policies during training at the checkpoint interval. Export messages appeared in the console (e.g., Exported ...HideAndSeekAgent-999900.onnx).

Results directory:

```
results\hns_run1\HideAndSeekAgent
```

Example exported files:

- HideAndSeekAgent-99900.onnx
- HideAndSeekAgent-199900.onnx
- HideAndSeekAgent-999900.onnx
- HideAndSeekAgent-1599900.onnx

Note: The latest trained policy is the ONNX file with the highest step number in its filename.

10. Running the Trained Model in Unity (Inference)

After training, the exported ONNX model can be assigned to the agents in the Test scene to run without Python.

- Stop training from the terminal with Ctrl + C.
- Open Assets/Scenes/Test.unity.
- Select each agent GameObject and open Behavior Parameters.
- Set Behavior Type to Inference Only.
- Assign the chosen .onnx model file to the Model field.
- Press Play to observe behavior in Unity.

11. Reproducibility Checklist

A run is reproducible if the following items are present:

- Build executable: Builds/Training/unity-ml-agents_hide-and-seek.exe
- Trainer config: ml-agents_config.yaml
- Training command (mlagents-learn) used for hns_run1
- Exported ONNX policy file(s) under results/hns_run1/HideAndSeekAgent/

Appendix: Useful Commands

Verify GPU availability (optional):

```
python -c "import torch; print(torch.cuda.is_available());
print(torch.cuda.get_device_name(0) if torch.cuda.is_available() else 'N/A')"
```

Force training on GPU (optional):

```
mlagents-learn "ml-agents_config.yaml" --env "Builds\Training\unity-ml-agents_hide-and-seek.exe" --run-id hns_gpu1 --force --no-graphics --time-scale 20 --torch-device cuda
```

Appendix A – TensorBoard Training Logs (Screenshots)

This appendix contains selected screenshots from the TensorBoard dashboard and command-line output to document the training process and key metrics observed during the Hide-and-Seek experiment.

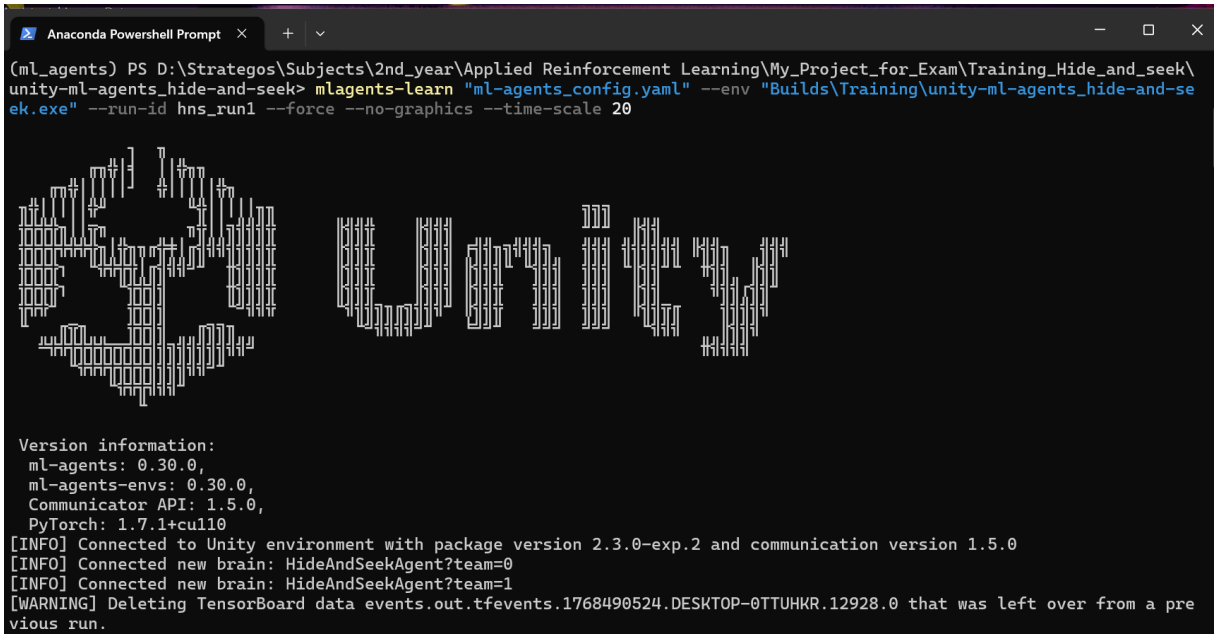

Figure A1. Command used to start the PPO training run (mlagents-learn) and environment connection logs.

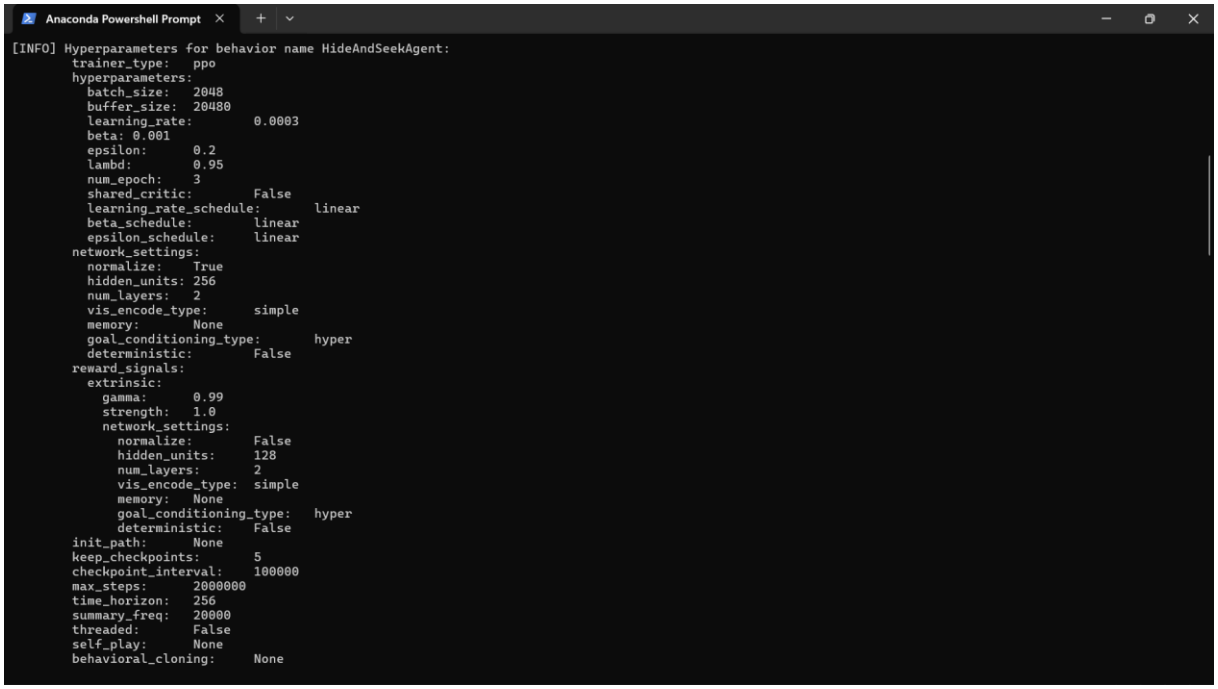

Figure A2. Printed hyperparameters and network settings used for the HideAndSeekAgent behavior.

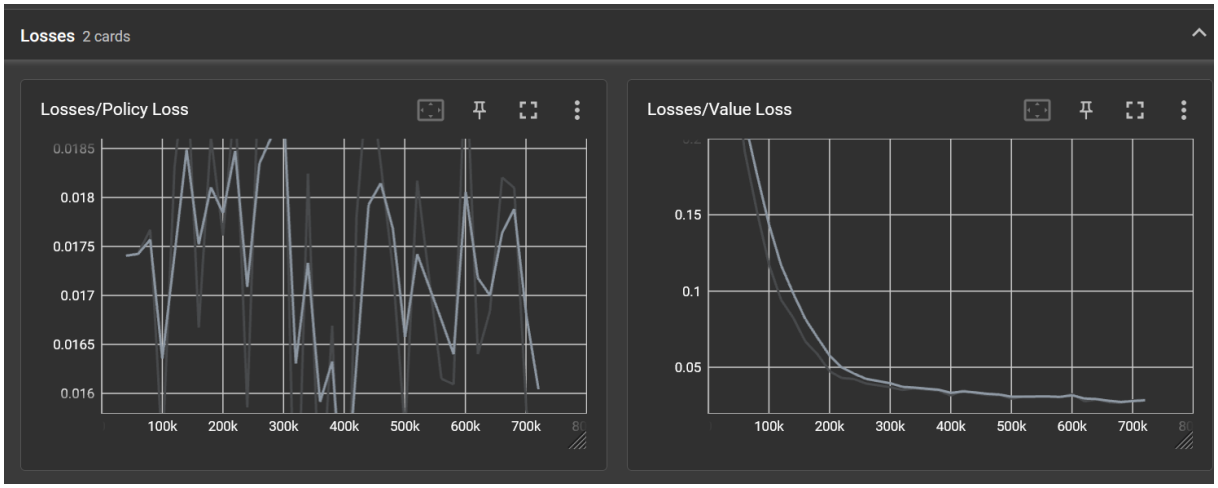

Figure A3. TensorBoard: Policy loss and Value loss during training.

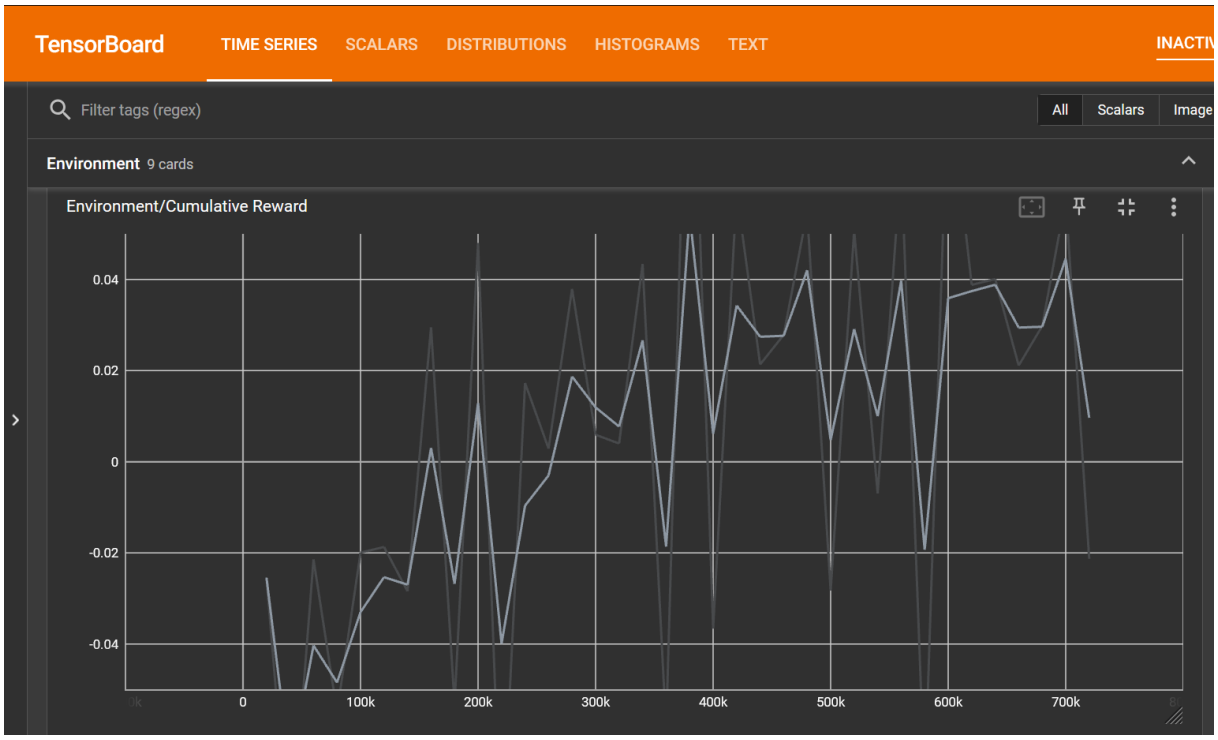

Figure A4. TensorBoard: Cumulative reward trend over training steps.

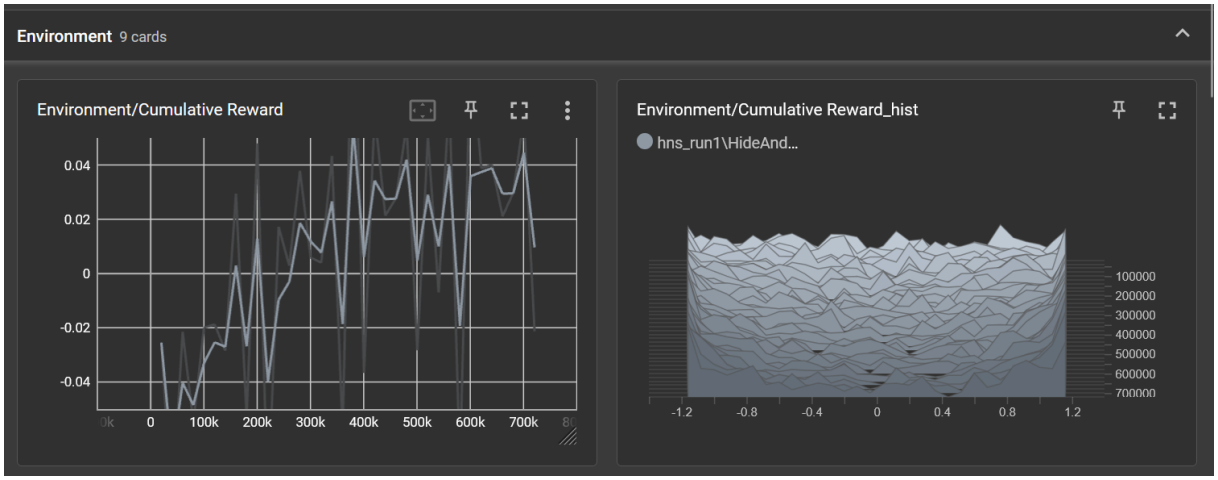

Figure A5. TensorBoard: Cumulative reward and its histogram (distribution across updates).

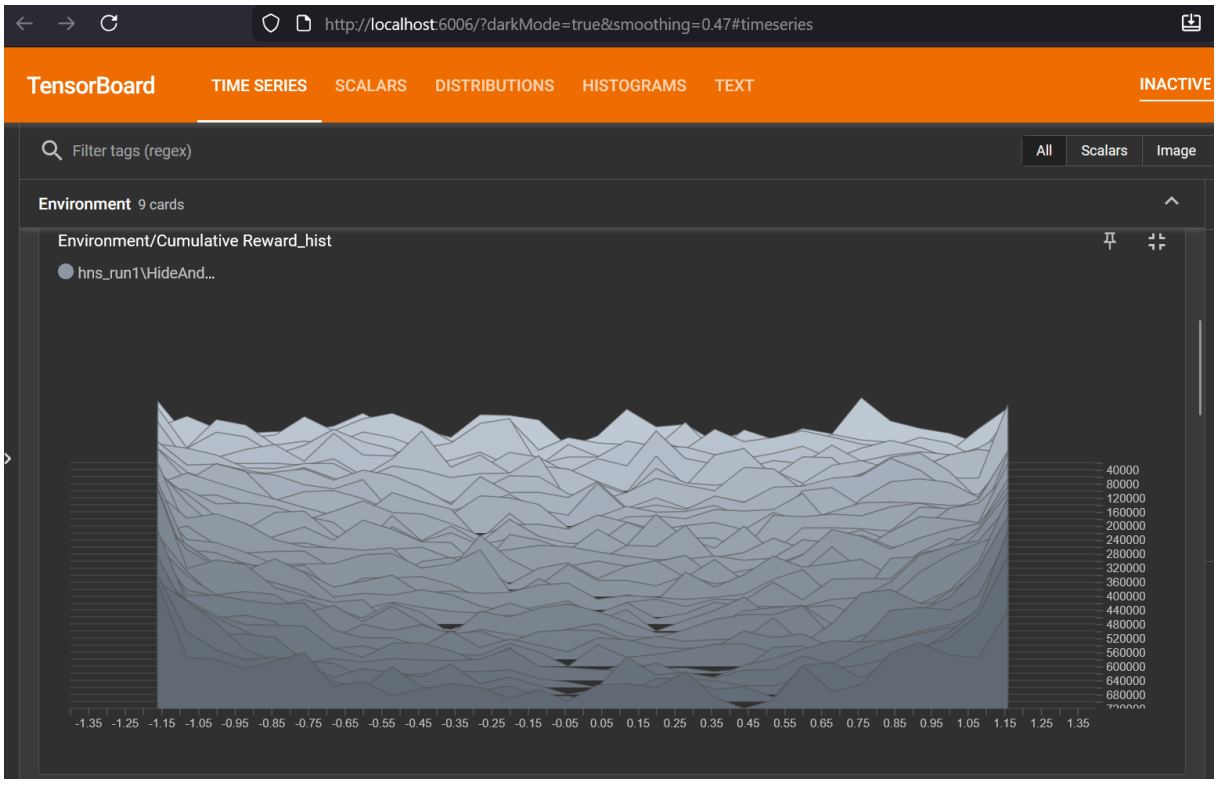

Figure A6. TensorBoard: Cumulative reward histogram view (distribution visualization).

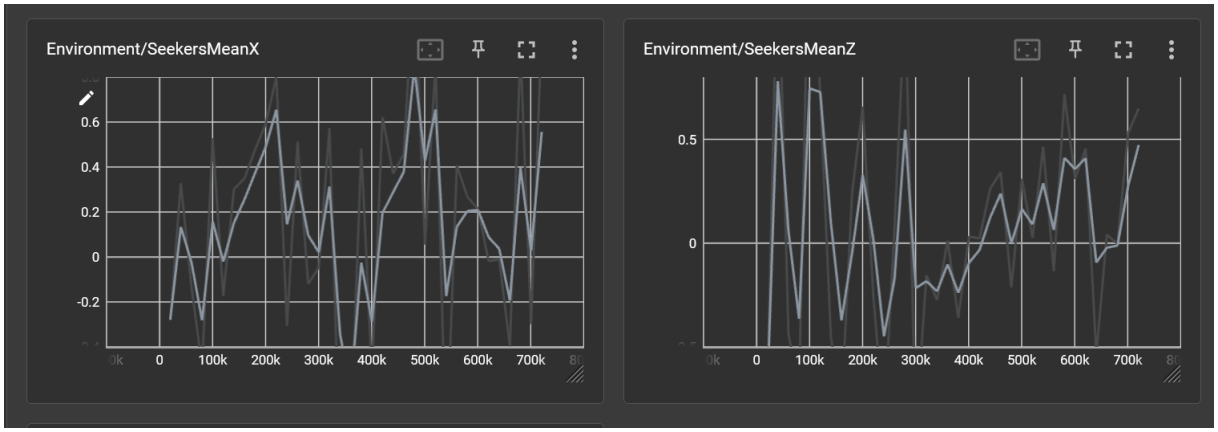

Figure A7. TensorBoard: Seeker agents mean position metrics (MeanX and MeanZ).

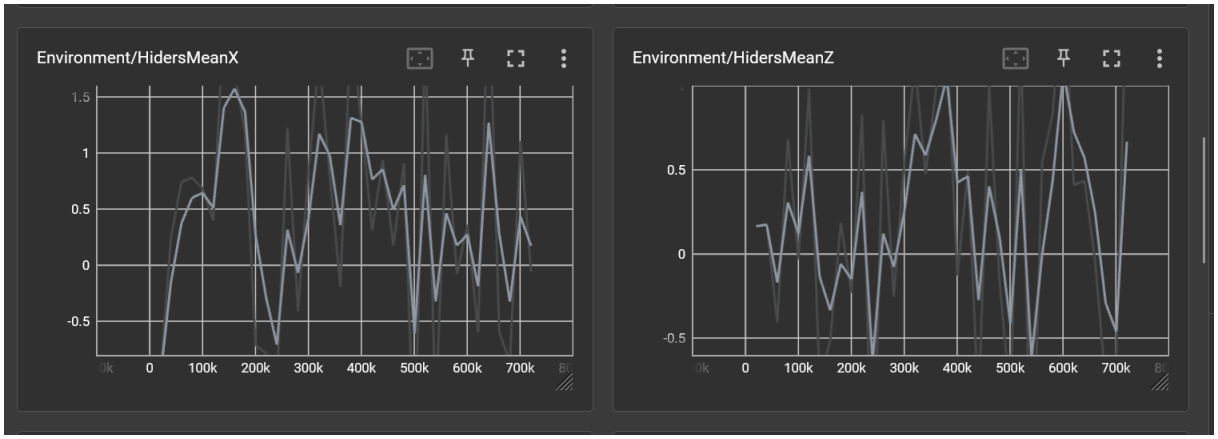

Figure A8. TensorBoard: Hider agents mean position metrics (MeanX and MeanZ).

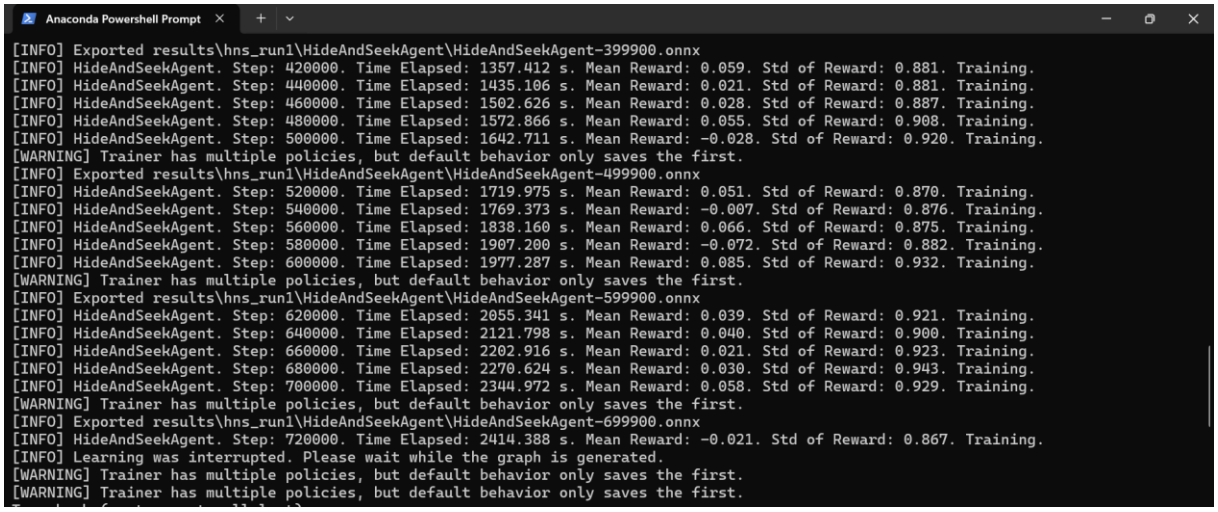

Figure A9. Training interruption example and periodic ONNX export messages produced by ML-Agents.