**How to use ML Framework**

**(v2.9)**

**What’s new in v2.9:**

* **Non-constant biases**
* **Precise layers format**
* **Activation functions change (output inclusive)**
* **Weights initialization change**
* **Minor graph changes**
* Download the .zip file from GitHub. Select the folder with the latest stable version.
* In the selected folder, you have access to three C# scripts. Upload all in your Scripts folder inside your Unity project.
* Create the following GameObjects :
* Your own AI agent prefab and add *Agent.cs* as component.
* One empty GameObject called Trainer and *Trainer.cs* add as component.
* [Optional] One empty GameObject called Environment. (parent it to all moving environmental objects in the scene, the purpose of it is to reset the position of the environment at the beginning of each Episode, you can to this manually too by overriding Trainer.cs)
* [Optional] One TMP text (used for realtime statistics)
* [Optional] One RectTransform (used for evolution performance graph)
* Override *Agent.cs* Script:
* [Optional but highly recommended] Override Manual() in order to test your AI behavior by keyboard (Manual() is a method called in Update(), so use Time.deltaTime if needed). When testing, set Behavior to *Manual*.

Tip: Always keep agent Behaviour to Static. When training, the behavior is auto set to Learning.

* Decide your AI’s observations number. Override CollectObservations() by fullfiling SensorBuffer argument with specific data (data must be float type, this method is called in Update(), so you can use realtime data)
* Decide your AI’s actions. Override OnActionReceived() by assigning actions depeding

on values received from ActionBuffer. (outputs are in [-1f,1f] range, same as above, this method is called in Update(), use Time.deltaTime if needed)

* Create a Rewarding System. Use AddReward() or SetReward() to deprive or grant your agents performance. Use EndAction() to stop your agents from doing action. (use these methods in OnCollisionEnter/OnTriggerEnter when your AI touches a goal or a trap)
* Create a Header(“=== Stats ===”) and use variables like speed, rb, etc.

Note: Update is used in *AgentBase*. Update is virtual, if you need to useit, override it and call base.Update()

* Create a brain model:
* Go back in UnityEditor and **Play**. Select your agent and take a look on *Agent (Script)* Component. Set *SpaceSize* by agent’s observations number and *ActionSize* by agent’s action number (as decided in the previous step). Modify *HiddenLayers* depending on your preference regarding on the NeuralNetwork structure. (biases are not included\*)
* Press *SaveBrain* checkbox once (don’t worry if it doesn’t modify to check sign) and press

**Stop**.

* Look in Assets\StreamingAssets\Neural\_Networks folder. There was created a .txt file with a brain model assigned with randomized weights. Right click on the file and select CopyPath.
* Set your Trainer:
* Drag and drop your prefab/agentInHierarchy in *AI Model*.
* Paste Path copy in *BrainModelPath*.
* [Optional] Drag and drop Environment GameObject in *Environment*. The environment object is used to reset the scene objects’s position/rotation/scale after each Episode.
* [Optional] Drag and drop TMP GameObject from Canvas in *Labels*. (don’t forget to adjust the text area – best on left half of the screen)
* [Optional] Drag and drop RectTransform [from Canvas] in *Graph*. (don’t forget to adjust the rectangle area and turn on Gizmos in Scene editor to watch the graph)
* Set *=== Training Settings ===* at your preference. In the beginning let the rest Training Settings at their default states.
* [Optional] Override Trainer.cs Script:
* Add Environment movement by overriding EnvironmentAction()
* Override OnEpisodeBegin() if needed.
* Override OnEpisodeEnd() if needed. (this method is called for each individual AI)
* Awake(), Start() and SetupTeam() are virtual**, *call base*** if you need to use them.
* Run the simulation:
* Since is a mono-environmental training, create a new layer that doesn’t intersect with itself and assign it to your AI (Edit -> Project Settings -> Physics[2D] -> Layer Collision Matrix) The AI’s will start training from their model starting position.
* Turn ON Trainer GameObject and press **Play**.
* Check console to see the results of each generation.
* Enter in SceneEditor with Gizmos ON to watch the graph.
* Best AI’s brain is always overwritten over the .txt file placed in Trainer as path.
* Use the NeuralNetwork post-training:
* In Agent.cs, delete any AddReward(), SetReward() or EndAction() calls.
* Copy the path of the brain model and paste inside *Path* (from *=== Network Properties ===*).
* Set Behaviour to *Self*.
* Your AI’s is now ready.
* TIPS SECTION
* Use a small number of hidden layers (even 0), the training process will be faster.
* **Environment:** In *Trainer* component add an Environment that contains objects that are liable to move/rotate/scale during the training session in order to reset them in each Episode.
* **Save Brains:** When you see an AI that has good behavior and doesn’t manage to get too much fitness, *Save Brains* from Trainer or *Save Brain* from Agent, stop the training and use his brain for another training session.
* **TeamSize:** Use as much AI’s as possible while keeping the framerate stable (around 60)
* **Episodes per Evolution:** Let more episodes to train for one generation. Rewarding is cumulative. The next generation occur every ***x***episodes.
* **Maximum Episode Time:** Give a limited time to your AI’s per Episode. \**Because some AI’s might never end their action the episode will run forever.*
* **Training Strategy Switch:** At the beginning of training, start with *Strategy 1*. When you see an AI that is quite good since his behavior is close to what you expect and he managed a good fitness, switch to *Strategy 2* (this way the best brain will be inserted in 1/3 of the AI’s and mutated every Episode). If your AI is ready you can go for a training with Strategy 3, where only the best brain is reproduced, this might be good to find a better AI with the same behavior.
* **Mutation Strategy Switch:** Use Classic *MutationStrategy* mostly. You might switch to Light/Strong Percentage in combination with *Strategy 3* to fine-tune your agent abilities.
* **Activation Type Switch:** You can play with any of these functions. *\*Output activation function will affect your output values range. Tanh returns values in range (-1,1), BinaryStep returns binary values, Sigmoid returns values in range (0,1), ReLU returns values In range [0, +infinity) and SoftPlus returns same values as ReLU but I a smooth manner.*
* **Initialization Type Switch:** The switch will not make a significant change.