**How to use ML Framework**

**(v4.0)**

**What’s new in v4.0:**

* **General interface changes and script changes**
* **Agent Manual control must now be implemented by overriding Heuristic() method**
* **Heuristic training leaks, settings available but is not fully implemented and tested**
* **Trainer EnvironmentType *NotSpecified* added.**
* **Reset Brain Model Fitness button added.**
* **Perspective Camera follow added.**
* **More Framework testing (also on 3D).**
* **SHORT TUTORIAL**
* Create the **AI** and add ***Agent.cs*** component. **Override** the Script. Create a new **AI Layer** that doesn’t interact with itself. **Run** the simulation, **set a brain** and press **SaveBrain** (check the brain in /Neural\_Networks/), then **stop**.
* Create a **Trainer** GameObject and add ***Trainer.cs*** component. [Override the Script]. **Add the AI** Model. **Add the path** of the brain. [Choose the **training environment Type** and add the ***Environment*** **tag** to it.] [Add TMP\_text and Rect for statistics.] **Setup the training** settings. **Run the training**, the brain model is overwritten.
* Place the trained brain path to the AI and set behaviour to **Self**.
* **IN-DEPTH TUTORIAL**
* 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 but recommended for training performance] One or more empty GameObjects. Add for each one ***Environment*** tag. [For more environments, insert in each environment a copy of the agent and call it **Start,** this way the trainer will now know how to reset agents positions for each different 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 the according Trainer.cs methods )
* [Optional] A canvas with RenderMode on Screen Space – Camera (and drag your main camera in Render Camera), followed by the following objects:
* One TMP text (used for realtime statistics)
* One RectTransform (used for evolution performance graph and neural net visualization)
* Override *Agent.cs* Script:
* [Optional but necessary for manual test and heuristic training] Override **Heuristic()** and **OnActionReceived()**,and set behaviour to *Manual* in order to test your AI behavior by keyboard. Override **CollectObservations()** and set Behavior to Heuristic to train your AI by learning from your actions.

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

* Decide your AI’s observations number. Override **CollectObservations()** by fullfiling *SensorBuffer* argument with specific data. Use *AddObservation(*) method to add different kind of observation. \*Note: every observation might have different size depending on how many float values are inside them*. Example: You decided to have 14 input values, you can add a Transform observation (where observation size is 10, 3 for position, 3 for localScale, 4 for rotation) ,a Vector3 observation (where observation size is 3, 1 for x, 1 for y and 1 for z) ,and an int value observation (where observation size is 1) . 10 + 3 + 1 = 14 input values. Do not let any gaps or inputs empty.*
* Decide your AI’s actions. Override **OnActionReceived()** by assigning actions depeding

on values received from *ActionBuffer*.You can access each action individualy by using *GetAction()* method and specify the index of the action. (outputs are in a range depending on the output activation function, usually (-1,1) if you use tanh, 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.\*TIP: divide the reward by *episodesPerEvolution* to normalize the fitness (in case you want to train your agents on different environments/on more episodes, the average fitness will be counted this way). Use *EndAction()* to stop your agents from doing action. (use these methods in OnCollisionEnter/OnTriggerEnter when your AI touches a goal or a trap)
* Use or create variables like speed, rb, etc. under ***===AI Properties===*** header.

Note: Update and Awake are used in *AgentBase*. Update and Awake are virtual, if you need to use them, override them and call their bases.

* 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] Specify an Environment Type. The type is used to reset the scene objects’s and agent’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. The Neural Network represents the brain of the best AI in the scene, with the neurons colored in its color, biases in green, positive weights in blue and negative ones in red) \*adjust Canvas plane Distance accordingly when in a 3D project
* 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.
* **ENVIRONMENT TYPES**
* Use *NotSpecified* if not needed.
* Add tag **Environment** for all your environments you want to use (at least 1).
* Add a copy of your agent in each environment and change it’s name to **Start**. Adjust it’s the position and rotation (even of it’s children).
* **MultipleLayers MonoEnvironment:** This is used if you want to train your agents together in a single environment. Place your AIModel [or a copy of it] in a starting position if you do not already have a *Start* (it represents the starting position for all of your agents in each episode depending on the environment).
* **MultipleLayers MultipleEnvironment:** This is used if you want to train your agents together but in different environments. If you don’t want some of the environments to be engaged in the training, remove their *Environment* tag. *EpisodesPerEvolution* will be changed automatically to the number of environments engaged.
* **SingleLayer MonoEnvironment**: This is used if you want to train your agents separately but in a similar environment. Take your one Environment (only with the *Environment* tag) and duplicate it as much as you need and place each environment in different places, such way they do not intersect. Team size will be automatically adjusted to the number of environments. (dev note: ai’s env might swap because the team is sorted, but it doesn’t matter)
* **STRATEGIES**
* Use a small number of hidden layers (even 0), the training process will be faster.
* **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.
* **HEURISTIC TRAINING**
* **NOT RELEASED YET.**
* **LearnRate** can be modified depeding on the batch size. Keeping it too low may not cause major impact on the network itself, keeping it to high doesn’t show any beneficial results, and can even shatter the old parameters peformance.
* **CostType** is the function used to calculate the total error of the outputs comparing to the user controls (desired outputs). Best keep on Quadratic.
* **CurrentError** shows the error between the network outputs and user outputs.

The heuristic training process must decrease this value.