**How to use ML Framework**

**(v5.38)**

**What’s new in v5.38:**

* **Minor bugs solved.**
* **Improved documentations.**
* **Several testing.**
* **SHORT TUTORIAL**
* Create the **AI** and add ***Agent.cs*** component. **Override** the Script. Create a new **AI Layer** that doesn’t interact with itself and a specific **Tag** ,apply both to the agent model. **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 behavior to **Self**.
* **TUTORIAL**
* Installation:

Download the .zip file from GitHub.

1. 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, or

1. From UnityEditor, select **Assets** -> **Import Package -> Custom Package** then choose the package from the zip file.
2. Create the following GameObjects :

* Your own AI agent [prefab] (+specific layer and tag) 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. [When dealing with more environments, insert in each environment a copy of the agent (consider the agent has a specific tag)**,** this way the trainer will now know how to reset agents positions for each different environment]
* [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 real time statistics)
* One RectTransform (used for evolution performance graph and neural net visualization)
* Override *Agent.cs* Script:
* Override **Heuristic()** and **OnActionReceived()**,and set behavior 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 Behavior 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 observations. \*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 (for Reinforcement training):
* 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 Training Strategies on 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.
* Depending of what kind of training you want to simulate, turn on or off the Trainer. *See below how to process* ***Heuristic Training****.*
* 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 Behavior to *Self*.
* Your AI’s is now ready.
* **AGENT IN DEPTH**
* **Behavior:** there are 4 types of behavior.
  + **Static:** the AI remains stationary, with no control by neural network or by keyboard.
  + **Manual:** the AI is controller by user. See MANUAL CONTROL.
  + **Self:** the AI is controlled by it’s own neural network.
  + **Heuristic:** the AI is trained heuristically. See HEURISTIC IN DEPTH.
* **Path:** represents the path to the Neural Network. Right click on the neural network file and press Copy Path, then paste it here.
* **Save Brain:** it [creates and] saves a neural network in StreamingAssets/Neural\_Networks.
* **Sensor Size:** the number of inputs/observations.
* **Action Size:** the number of outputs /actions.
* **Hidden Layers:** the number of hidden layers. Each element is a layer. Each element value represents the number of neurons on that hidden layer.
* **[Output] Activation Type:** Function used for neuron activation. *[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 SiLU returns same values as ReLU but in a smooth manner.
* **Initialization Type:** The switch will not make a significant change, is more like a flavor for weights and biases initialization.
* **Heuristic Properties:** check HEURISTIC IN DEPTH.
* **Methods to override:**
  + **CollectObservations(SensorBuffer) ->** AddObservation(observation)
  + **OnActionReceived(ActionBuffer) ->** GetAction(index)
  + **Heuristic(ActionBuffer) ->** SetAction(index, actionValue)
  + **HeuristicOnSceneReset() ->** called after EndAction() for Manual/Heuristic behavior.
* **Methods to use:**
  + **AddReward(reward, [bool value]) ->** if the boolean value is true, the reward is added even if the agent’s action has ended. (usually used in OnEpisodeEnd())
  + **SetReward(reward)**
  + **EndAction() ->** sets agent’s behavior to static until reseting the episode.
* **TRAINER IN DEPTH**
* **AI Model:** drag and drop the AI gameobject.
* **Brain Model Path**: Copy the brain file path.
* **Interaction Type:** Choose the training environment option (see INTERACTION TYPES)
* **Reset Brain Model Fitness:** When a new training session begins, the fitness from the brain file is not taken in consideration, and the NN starts with 0 fitness (the training does not continue from the fitness found in the file). In this situation, the neural network will be affected from the first 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.
* **Camera Follows Best AI:** implicitly finds the first object with Camera component. It follows he AI with the best real-time fitness.
* **Labels and Graph:** Drag and drop a TextMeshPro canvas object and a normal Canvas empty object to view the real-time statistics.
* **TeamSize:** Use as much AI’s as possible while keeping the frame rate stable (around 60)
* **Episodes per Evolution:** Let more episodes to train for one generation. Rewarding is cumulative. The next generation occurs every ***x***episodes.
* **MaxEpisodes:** represents the length of the current training session.
* **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:** 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.
* **Methods to override:**
  + **SetupTeam() -> access each AI from *team* struct array.**
  + **EnvironmentAction() -> called each frame**
  + **OnEpisodeBegin() -> called immediately after each episode reset**
  + **OnEpisodeEnd(AI) -> called immediately after reseting the episode**
* **AI structure has the following fields:**
  + - agent: access agent GameObject
    - script: access Agent script
    - fitness: access agent’s current fitness (read-only, modifications do not affect)
* **INTERACTION TYPES**
* Use *Not Specified* if not needed.
* Add tag **Environment** for all your environments you want to use (at least 1).
* Add tag **AI** (or named as you wish\*) for your agent (if not added it may cause bugs).
* Add a copy of your agent in each environment. Adjust their positions and rotations (even of it’s children) at your preference. You can rename the copies as you wish to avoid ambiguities, but is not necessary.
* **More Agents Per Environment**: In this situation, the agents are trained together in the same environment by overlapping each other. When there is more than 1 environment, is necessary for each of them to have a model of the agent inside (representing the starting position), otherwise the main model is used as a start.
* **One Agent Per Environment**: This is mainly used when you need to let just 1 agent to interact with the environment. To properly use this, take your normal environment and clone it for several times. When training, one agent is placed inside each of them. (team size is adjusted automatically to the number of environment clones).
* **\*Whenever you want to not use an environment, remove it’s Tag.**
* **MANUAL CONTROL**
* ***It REQUIRES a neural network to be instantiated before use.***
* **Heuristic()** and **OnActionReceived()** methods must be overridden in order to control the AI.
* Call **EndAction()** to reset the AI’s position.
* **SELF CONTROL**
* ***It REQUIRES a neural network to be instantiated before use.***
* **CollectObservations()** and **OnActionReceived()** methods must be overridden in order for the AI to self control.
* **EndAction()** calls from the Agent script should be removed.
* **HEURISTIC TRAINING**
* Load a path to a brain. Turn OFF the *Trainer*. Set behavior to Heuristic. Set module to **Append/Write** (it will write the training data in a file). Start your simulation and control your agent consistently. **Do not** **stop** the simulation until the data was stored. After this, go to the newly created Directory *Heuristic\_Samples* and copy the path of the file to **Samples Path**. Set the Module to **Learn**. Run the simulation again, this time deep learning is processed using the data inside the file. Do not stop until is finished.
* **Samples Path** is the path of the file where training data is stored or used. It is auto created if doesn’t exist when appending/writing.
* **Kill Static Action** button is used to omit training samples where the user didn’t introduced inputs *(all elements in the action vector are equal to 0).* This is used for a more optimized training, but only for special cases (usually there are samples with no inputs, and this kind of samples are slowing down the process). If the AI needs to “wait”(it means no actions) at some point, this feature must be turned off.
* **Module** represents the interaction mode. If is set to **Append** or **Write**, the user gives training data through the keyboard or mouse, data is stored in a txt file in Heuristic\_Samples. If is set to **Learn**, the machine reads the data from the *Samples\_Path* and trains. \*Note: do not append more than 1 million samples in just 1 file, the machine will work extremely slow when training.
* **Epochs** are the number of times the data is reprocessed. A high number of Epochs may cause overfitting, a low number may cause underfitting. \*\*\****Reducing the number of epochs while training is the only way to stop safely the simulation.***
* **Session Length** represents the time you spent on giving data to the training file. Longer the session, more divert the data will be.
* **LearnRate** can be modified depending on the learn impact for your agent. 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 performance. For example: if the learn rate is 0.01, for 1 epoch the impact is 1%. A good formula to follow:

Learn Rate = impact you want% / number of epochs.

DevNote: the learnRate is divided by the batch\_size in HeuristicPreparation().

* **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 as much as possible. When the *Current Error* is approaching to 0, you can end the Heuristic Training and pursue with the Trainer.

* **ErrorGraph** shows the progression of the error. To get the best results for your agent, watch the graph carefully. When the error reaches a minimum value and from there is starts rising slowly, finnish the training by changing the epoch from inspector to 0.
* **Environment** works in the same way, but don’t needs a specific tag, just drag and drop your Environment gameobject here to reset the environment every time the agent action ends. Supports only *monoenvironment*.
* Override **HeuristicOnSceneReset()** to modify the environment or agent position differently for each Scene.
* This type of training is good especially before the Trainer, to yield a good start for the agents. UnityEditor FPS will decrease dramatically ( to ~1 fps), that’s because all the resources are used only for the training.