ShapeInput ML - Unity Input Package

Finally a new type of input! Attack, cast spells, move, etc by drawing shapes!
Unity 2019.4.29f1 and later
Support: hybrid47+unity@gmail.com

Description

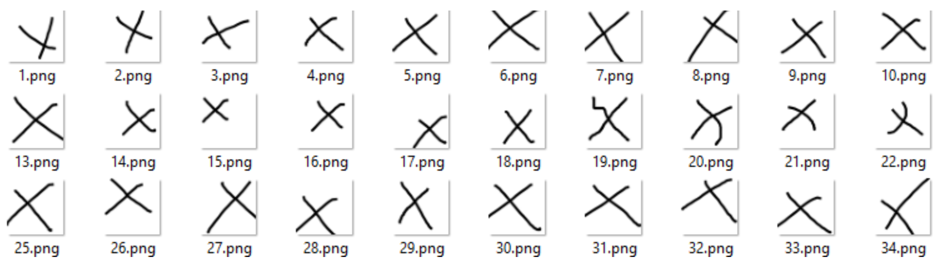
Exposes a new type of user input: *drawn shapes*. Use/edit the prefab Canvas+Input Panel as a visible or hidden drawing container, and read from `ShapeInput.GetShape()` as you would other inputs. Behind the scenes, the drawing is used as input on a super-fast pre-trained machine learning model.

When a predefined shape is drawn on the panel, it will process it through a pre-trained AI model and register as a new input in `ShapeInput` *until its value is read once*, clearing it. The confidence (float 0..1) is also exposed for possible use (eg. power modifier, accuracy).

The included model was trained on 6 different easily identifiable shapes, and a class for stray marks and nonsense drawings to avoid false classification of weak inputs (it was noticed that a stray mark was being classified as a defined shape *with* high confidence, preventing applying minimum confidence rules).

Instructions on how to build and integrate new models are included, mostly using **third party tools**. Generating new models is not considered part of the purchased package, and so the ***process is not supported***. A winforms .NET project to facilitate generating properly formatted training images is included (ShapeInputMLTrainingImageGenerator). This project is in .NET 4.8 (VS2019). You point it to an output folder, draw a shape, and press Enter to save, rinse and repeat. It is recommended to have 50-100+ samples of each shape, the more the better, drawn in all different locations, rotations, and sizes.

Some drawing samples for an X:



Unity Dependencies

Barracuda (com.unity.barracuda) 3.0+
https://docs.unity3d.com/Packages/com.unity.barracuda@3.0/manual/index.html

Package Installation Tip
If you cannot find the package in Package Manager UI, select the menu option to add package by name, and input: **`com.unity.barracuda`**

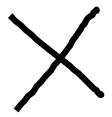
Usage

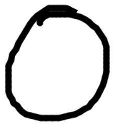
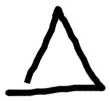
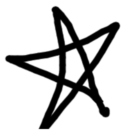
Open the demo scene and inspect the canvas prefab. Aside from the package dependency of com.unity.barracuda, there are no other modifications to the project from its defaults. While the demo uses the built-in renderer, it was tested and works with URP. Shapes traced over the input panel are interpreted by the machine learning model in real time and registered in the `ShapeInput` class to be read in the next `Update()`. The drawing board Panel can be hidden but maintain functionality by changing the alpha to 0. The input drawing Panel with the ShapeML scripts must maintain a 1:1 aspect ratio (ie. stay a square) so it scales proportionally to the expected ML model input size (224x224).

When using the ShapeInputCanvasScreenCamera prefab in your own project, make sure you also have an `EventSystem` gameobject in your scene.

When the project is run in Debug, it will output an `rt.png` to the root folder each time a user input is resolved, containing the raw image that the machine learning model will see (this is inside `MLShapeInput.cs`). If the image is blank or doesn't look like the training images, something may have been changed and **it will not work properly.**

Recognized Shapes

X **`Shapes.X`**	

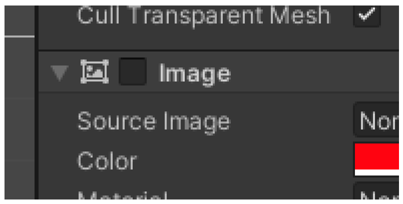
O `Shapes.O`	
Triangle `Shapes.Triangle`	
Star `Shapes.Star`	
Heart `Shapes.Heart`	[HEART]
Square `Shapes.Square`	[SQUARE]

An additional shape class, `Shapes.NONSENSE` was trained on stray marks and drawings that do not resemble any of these shapes. This would be considered either 'unrecognized' or an 'incorrect' input based on your usage, versus `Shapes.NONE` which is returned on frames with no queued input.

Open up `DemoGameScript.cs` and review the `Update()` function for a usage example.

Tracer `RectTransform`

An optional UI component that will follow the path of the player's drawing as visual feedback. This component could contain a TrailRenderer or Particle System, which requires that our Canvas be in "Screen Space - Camera". Alternatively, it can be a simple image such as a crosshair or magic wand. The tracers Image component is disabled by default, so it is not visible unless you enable it:



Training New Models (as of 6/28/2022)

The process of training new models is relatively straightforward, **when everything works**. As it relies heavily on third party programs, we cannot offer support for this process when related to those tools. Once you get the process down, repeating it with new training sets should be easy.

Training Set Images

The first step is generating a training set of example images for each shape you wish to use. Use the included .NET application to generate a training set of at least 50 image samples of each shape you want in your game. Open `ShapeInputMLTrainingImageGenerator.exe` and choose an output directory for the shape, it is suggested to create a folder for each shape. Start drawing on the canvas and press Enter to save or Escape to clear the canvas. Maintain the brush size at 10. Having a drawing tablet is recommended, as it will result in much more comparable samples to the end result finger based drawings that will be interpreted. After completing a shape press Enter and a 224x224 numbered png will be saved to the selected output folder. Continue until you have completed your training set of at least 50 images per shape. You may also want to create a folder with a few test images of each shape, so that you can test your model after training it.

Training

Models are trained using this free web-based tool:

```
Teachable Machine
https://teachablemachine.withgoogle.com/train/image
```

On the left hand side there are 2 classes to start (each shape is a "class"). Begin to rename the classes to match your shapes, click 'upload' within the class, and drag and drop all of the training images for that shape into that area. Repeat until all of your shapes are registered.

Next, click Advanced under the training box and change Epochs to 100 (or higher, but not too high to avoid overfitting). Click Train Model and wait for it to complete.

Once complete, choose 'Input file' in the testing area and upload some of the test images, and validate that it correctly identifies the shape. **If it does not,** you may need to change the number of epochs, batch size, add more training images, or remove shape(s) that are too similar to other shapes and re-train.

Once you are ready to export, click "`Export Model`", select the middle "`Tensorflow`" tab, select model conversion type '`savedmodel`', and click '`Download my model`'. Once it downloads, we move on to the next step.

Converting to ONNX

The Unity ML Interpreter (Barracuda) uses the open ONNX model format. Unfortunately, the Teachable Machine tool does not offer that as an export option, but it is trivial to convert the TF export to ONNX using Python.

Prerequisites

After installing python, the subsequent `pip` commands should be entered into a command prompt.

Install Python (if you do not have it)	https://www.python.org/downloads/
Install Tensorflow:	`pip install tensorflow`
Install ONNXRuntime:	`pip install onnxruntime`
Install tf2onnx: https://github.com/onnx/tensorflow-onnx	`pip install -U tf2onnx`

If all went well, we should be able to convert our downloaded model with one command. Open a command line window and navigate it to where you have unzipped your downloaded model. You should be in a folder with a `labels.txt` file and a `models.savedmodel` subfolder. Run the below command to convert your model to an onnx file:

```
python -m tf2onnx.convert --saved-model model.savedmodel --output shapemodel.onnx
```

Feel free to change the output filename, but you *must retain the .onnx extension*. If all goes well, you will now have a `shapemodel.onnx` file which can be imported into Unity.

Importing into Unity

Drag and drop the ONNX file into your project. Replace the included NNModel on the prefab in the inspector with your model. Go back to your downloaded model folder and open the `labels.txt` file. Back in Unity, open the `Shapes.cs` file from my package. **In order for your model to work properly, you need to update the Shapes enum to match your labels.txt file**. You can label the shape as you wish, but the **integers must match your labels.txt file**. Please note you must keep the `NONE = -1` enum value.

Example:

labels.txt

```
0 Circle
1 Square
2 Triangle
```

Shapes.cs

```
public enum Shapes{
  NONE = -1,//DO NOT MODIFY OR REMOVE
  CIRCLE = 0,
  SQUARE = 1,
  TRIANGLE = 2
}
```

If you've made it this far, your model is now ready for use.

Caveats

Without training an extra class on stray marks, random pixels, nonsense shapes, etc. it has been observed that a single dot or line segment registers as an X (on shapeml.onnx) with high confidence. Depending on usage, this can be undesirable.