## What is Gesture Recognizer?

Gesture Recognizer is a very simple tool for handling and recognizing user input. Users can draw on the screen and Gesture Recognizer can recognize the sketch.

## How do I use Gesture Recognizer?

Gesture Recognizer is now even easier to use and there are two methods.

## First method (event-based new way)

This is the new method added in this release and it is based on events. Also, it includes a drawing method on its own.

1- Drag & drop the **GestureRecognizer** prefab onto the scene from the **Prefabs** folder. Modify it as you like (color, width etc.). Don't forget to fill **Library To Load** attribute on this prefab. This is the name of the XML file (without the ".xml" extension) that includes your gestures.

2- Create a C# script and include the `GestureRecognizer` namespace at the beginning of your script:

```
using GestureRecognizer;
```

3- Add your own method and name it as you desire. This is the method to do whatever you want to do (shoot fireballs, cast magic spells etc.). For the sake of clarity, I named it `OnGestureRecognition` here:

```
void OnGestureRecognition(Result r) {
    Debug.Log("Gesture is " + r.Name + " and scored: " + r.Score);
}
```

4- Subscribe this method to `OnRecognition` method of `GestureBehaviour` in `OnEnable` and unsubscribe it in `OnDisable` and `OnDestroy`:

```
void OnEnable() {
    GestureBehaviour.OnRecognition += OnGestureRecognition;
}

void OnDisable() {
    GestureBehaviour.OnRecognition -= OnGestureRecognition;
}

void OnDestroy() {
    GestureBehaviour.OnRecognition -= OnGestureRecognition;
}
```

## Second Method (the old way)

This method captures the gesture in `Update` method (sometimes `Update` provides more control than events), however, it expects you to write your own code to draw the gesture on the screen.

1- Put `gestures.xml` in **Resources** folder in **Assets** folder. The name of this XML file ("gestures") will be your library name.

2- Include the `GestureRecognizer` namespace at the beginning of your script:

```
using GestureRecognizer;
```

3- Define a new `GestureLibrary` variable:

```
private GestureLibrary gl;
```

4- Construct it in `Start()` method with your library name (as stated above, it is "gestures"):

```
void Start() {
    gl = new GestureLibrary(libraryToLoad);
}
```

5- Capture mouse (or touch) position in `Update()` method:

```
List<Vector2> points = new List<Vector2>();

if (Input.GetMouseButtonDown(0)) {
    points.Clear();
}

if (Input.GetMouseButton(0)) {
    points.Add(
        new Vector2(Input.mousePosition.x, -Input.mousePosition.y)
    );
}
```

6- Create a Gesture from captured points and recognize it:

```
if (Input.GetMouseButtonUp(0)) {
    Gesture g = new Gesture(points);
    Result result = g.Recognize(gl);
}
```

7- At this point, the result variable would hold the name of the recognized gesture and its score:

```
Debug.Log("Recognized gesture: " + result.Name + "; score: " + res
ult.Score);
```

## What is MultiStroke Recognizer?

Gesture Recognizer is a very good tool, but it is missing something very important: the ability to recognize multi stroke gestures. You can recognize single stroke shapes, such as rectangles, circles, numbers or letters with Gesture Recognizer. However, in order to recognize gestures that have multiple strokes such as Japanese alphabet, Elvish letters or Dovahzul (Dragon Language), you need to use MultiStroke Recognizer. However, please keep in mind that MultiStroke Recognizer is a little bit more complex than Gesture Recognizer.

## How do I use MultiStroke Recognizer?

MultiStroke Recognizer works very similar to Gesture Recognizer and there are two different methods just like Gesture Recognizer.

## First method (event-based)

This is the same method that you can find in Gesture Recognizer and it is based on events. Also, it includes a drawing method on its own.

1- Drag & drop the **MultiStrokeRecognizer** prefab onto the scene from the **Prefabs** folder. Modify it as you like (color, width etc.). Don’t forget to fill **Library To Load** attribute on this prefab. This is the name of the XML file (without the “.xml” extension) that includes your multi-strokes. Use a multi-stroke XML file, using a standard single stroke XML files will give errors.

2- Create a C# script and include the `GestureRecognizer` namespace at the beginning of your script (if you are confused with the names, please refer to Troubleshooting):

```
using GestureRecognizer;
```

3- Add your own method and name it as you desire. This is the method to do whatever you want to do (shoot fireballs, cast magic spells etc.). For the sake of clarity, I named it `OnMultiStrokeRecognition` here:

```
void OnMultiStrokeRecognition(Result r) {
    Debug.Log("Multi stroke is " + r.Name + " and scored: 
" + r.Score);
}
```

4- Subscribe this method to `OnRecognition` method of `MultiStrokeBehaviour` in `OnEnable` and unsubscribe it in `OnDisable` and `OnDestroy`:

```
void OnEnable() {
    MultiStrokeBehaviour.OnRecognition += OnMultiStrokeRecognition
;
}

void OnDisable() {
    MultiStrokeBehaviour.OnRecognition -
= OnMultiStrokeRecognition;
}

void OnDestroy() {
    MultiStrokeBehaviour.OnRecognition -
= OnMultiStrokeRecognition;
}
```

## SECOND METHOD (CAPTURE AND RECOGNIZE IN UPDATE)

This method captures the multi stroke in `Update` method (sometimes `Update` provides more control than events), however, it expects you to write your own code to draw the multi stroke on the screen. Be aware that this method is harder than its counterpart in single stroke recognition.

8- Put `multistrokes.xml` in **Resources** folder in **Assets** folder. The name of this XML file ("multistrokes") will be your library name.

9- Include the `GestureRecognizer` namespace at the beginning of your script:

```
using GestureRecognizer;
```

10- Define a new `MultiStrokeLibrary` variable. Since there are multiple strokes, you need to be able to record the stroke IDs along with the points. So, you are going to need a list of points with IDs. I would suggest using MultiStrokePoint class included in the package, which offers this functionality by default. You also need to keep an int variable to hold the stroke IDs.

```
private MultiStrokeLibrary ml;
private List<MultiStrokePoint> multiStrokePoints;
private int lastStrokeID = -1;
```

11- Construct it in `Start()` method with your library name (as stated above, it is "multistrokes"):

```
void Start() {
    ml = new MultiStrokeLibrary(libraryToLoad);
    multiStrokePoints = new List<MultiStrokePoint>();
}
```

12- Capture mouse (or touch) position in `Update()` method. You will have to create a new stroke with a unique ID on each mouse down. You can implement a method on your own, or you can use something similar to `AddStroke()` method that can be found in `MultiStrokeCapturePoints.cs`:

```
List<Vector2> points = new List<Vector2>();
```

```
if (Input.GetMouseButtonDown(0)) {
    point = Vector2.zero;
    lastPoint = Vector2.zero;
    AddStroke();
    lastStrokeID++;
}

if (Input.GetMouseButton(0)) {
    points.Add(
        new MultiStrokePoint(Input.mousePosition.x, -
Input.mousePosition.y, lastStrokeID)
    );
}
```

13- Create a MultiStroke from captured points and recognize it:

```
if (Input.GetMouseButtonDown(1)) {
    MultiStroke m = new MultiStroke(points.ToArray());
    Result result = m.Recognize(ml);
}
```

14- At this point, the result variable would hold the name of the recognized multi stroke and its score:

```
Debug.Log("Recognized multi
stroke: " + result.Name + "; score: " + result.Score);
```

## TROUBLESHOOTING

Problem: It does not recognize accurately! I drew a triangle but it says "it is recognized as rectangle". What gives?
Solution: Open up the demo scene, draw whatever you want to draw, if it doesn't recognize accurately, just enter the correct name in "Add as" field and click "Add". The more you teach, the better GestureRecognizer recognizes. I suggest **adding at least three samples** for a single gesture. But beware, GestureRecognizer uses strings as keys for gestures. That means, you need to write the correct name while adding a new gesture. If you draw a rectangle and name it as a triangle you might get confusing results.

Problem: I just added a new gesture with wrong name!
Solution: Open up `gestures.xml` in your favorite text editor (mine is Notepad++), and delete the last gesture you added. It should be something like this:

```xml
<gesture name="wrong-name" usergenerated="true">
    <point x="307" y="216" />
    <point x="333" y="186" />
    <point x="356" y="215" />
    <point x="375" y="186" />
    <point x="399" y="216" />
    <point x="418" y="186" />
</gesture>
```

Problem: I have added new gestures, but it doesn't seem to be included in my build!
Solution: There is no problem in Unity Editor, since Gesture Recognizer uses the XML file in the Resources folder in Editor mode. However in builds (.exe, .apk, etc.), GestureRecognizer copies the XML files from Resources folder to persistent data path if it is not already there (because on mobile systems we can only save in persistent data path). This ensures that you can save your new gestures to the XML file on all platforms (except web player). If you want to include your latest XML file, if you are using,

a- the old way and on PC or Mac:
Go to the persistent data path, you will find your XML file there. Just delete it and the new one will be included in the build.

b- the old way and on mobile:
Delete the game and install the new build.

c- the new way:
Enable force copy in Gesture Recognizer prefab and build your game again. This will force it to use the new one. Don't forget to turn it off if you are saving new gestures in your games, since this will overwrite the XML file everytime the player starts a new game.

d- any way and for any build:
Changing your version number in publish settings would be enough, because applications will use a new persistent data path for each version number.

Problem: I can't save new gestures!
Solution: Is `gestures.xml` in **Resources** folder, directly under **Assets**?

Problem: I don't want your gestures, I want to define a whole new set!
Solution: Open up `gestures.xml` in your favorite text editor (remember Notepad++?), remove everything between `<gestures>` and `</gestures>`. File should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<gestures>
</gestures>
```

Start the demo scene and start adding your new gestures. I know, this is a lame way to do, but I will implement a gesture editor in an upcoming release. Same is also applied to MultiStroke Recognizer. An empty multi-stroke XML file should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<multistrokes>
</multistrokes>
```

Problem: I have cleared `gestures.xml`, now it gives me this error:

```
XmlException: Text node cannot appear in this state.  Line 1, position 1.
```

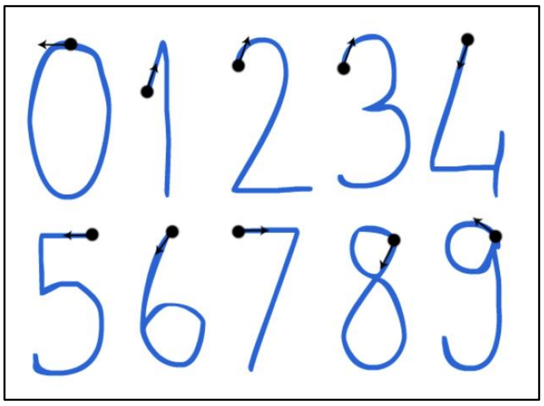
Solution: Be sure `gestures.xml` file starts with

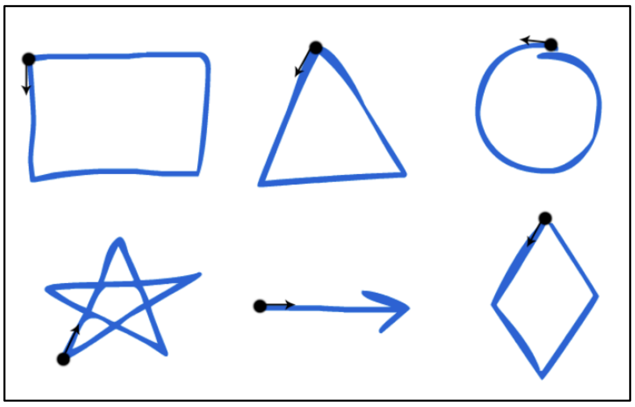
```
<?xml version="1.0" encoding="utf-8"?>
```

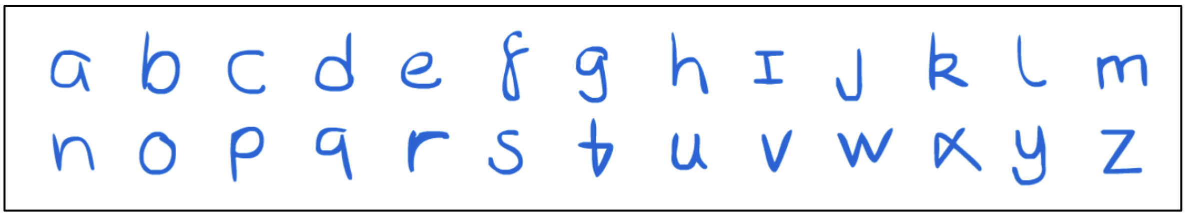
and the encoding is set as UTF-8 without BOM.

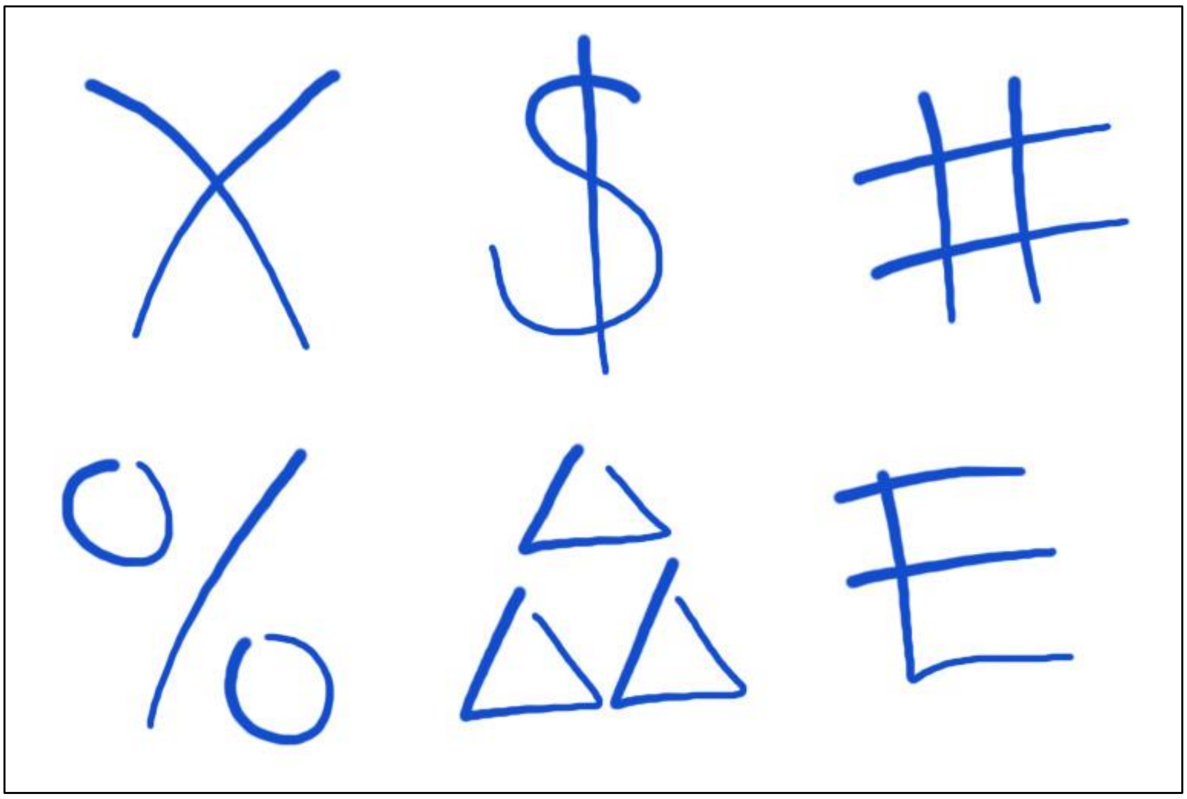
Problem: You talk about `gestures.xml` but there isn't a `gestures.xml` file?
Solution: In the previous version I have included 4 gesture libraries and they are name appropriately to reflect their contents: `alphabet.xml, shapes.xml, numbers.xml, multistroke_shapes.xml`

Problem: But what gestures are in them?
Solution:









Please keep in mind that you cannot use a single stroke XML file in multi-stroke recognition and you cannot use a multi-stroke XML file in gesture recognition.

Problem: I followed the document exactly but I cannot see anything on the screen?
Solution: If you used the second method you need to add your own code to see the gesture on the screen. You can use the code in `CapturePoints.cs` as a base for your own script. Hell, you can even use it as it is.

Problem: There seems to be two different methods for recognizing, right?
Solution: Yes, there are. The default method is $1 method, which is a good recognizing method and the scoring system it uses is very simple (return a float between 0 and 1, 1 being exactly the same). The second method (Protractor )is much faster than the default $1 method, however its scoring system is a little bit vague. You can use this second method by adding `true` as a second argument in `Recognize()` method just like this:

```
    Result result = g.Recognize(gl, true);
```

Or if you are using the new event-based way, you can just enable **Use Protractor** in **Gesture Recognizer** prefab.

Problem: These names really confuse me.
Solution: Well, actually it is very simple. Gesture Recognizer is:

- The name of the Asset Store Package.
- The name of the namespace which includes every single class in this package.
- The name of the system that recognizes single stroke gestures.

On the other hand, MultiStroke Recognizer is:

- The name of the system that recognizes multi stroke gestures.

You see, simple right?

Problem: Why are there two different recognizers? How do I decide which one to use?
Solution: Gesture Recognizer recognizes only single stroke gestures while MultiStroke Recognizer can recognize both single and multi-stroke gestures. If you are going to use single and multi-stroke gestures, you have to use MultiStroke Recognizer. If you are going to use only single stroke gestures and need something simple I would recommend using Gesture Recognizer, but you can still use MultiStroke Recognizer if you would like to.

Problem: They are not actually that similar, are they?
Solution: Well, you are right. While the start point and the direction of the gesture are important in Gesture Recognizer, MultiStroke Recognizer does not take into account these variables while recognizing a gesture. So, when you draw a rectangle in Gesture Recognizer, starting from upper left point and going down, it might give unexpected results if you compare it with a rectangle drawn in a different way (say, starting from lower left point and going up). However, in MultiStroke Recognizer, you can draw them anyway you want, and it will still recognize it correctly.

Problem: How does the scoring work?
Solution: MultiStroke Recognizer uses a different scoring system in which the lower the score, the similar the multi-strokes. Think about it as distance between two multi-strokes. The lower the distance, the similar they are.

Problem: Who are you again?
Solution: My name is Oguz Konya. I am a software developer located in Istanbul, TURKEY. I mainly develop games and simulations for customers only.

Problem: How can I contact you?
Solution: Just email me at oguz@oguzkonya.com or oguzkonya34@gmail.com. Please use a clear subject related to product such as “GestureRecognizer can’t seem to overthrow the world”.