UNISTROKE GESTURE RECOGNITION

Documentation

Contacts:
Telegram: @NKonovaloff
Email: nd.konovaloff@gmail.com

About Asset

This asset is inspired by the unusual game mechanics from Arx Fatalis and Black & White, which allow you to cast spells by drawing certain signs. This asset contains tools for working with gestures and their recognition to create unusual game mechanics in your projects.

The gesture recognition method is based on the $1 algorithm. The recognition algorithm does not require training and allows you to add new gestures even during operation. The recognition method is based on comparing the location of the path points of the recorded player gesture with pre-created patterns.

The recognition algorithm code is developed using the Burst Compiler and Unity Job System to ensure maximum performance on any platform.

The recognition system supports the following types of gestures:

- Rotation-sensitive/invariant;
- Direction-sensitive/invariant;
- Aspect ratio-sensitive/invariant.

v1.2 Changelog

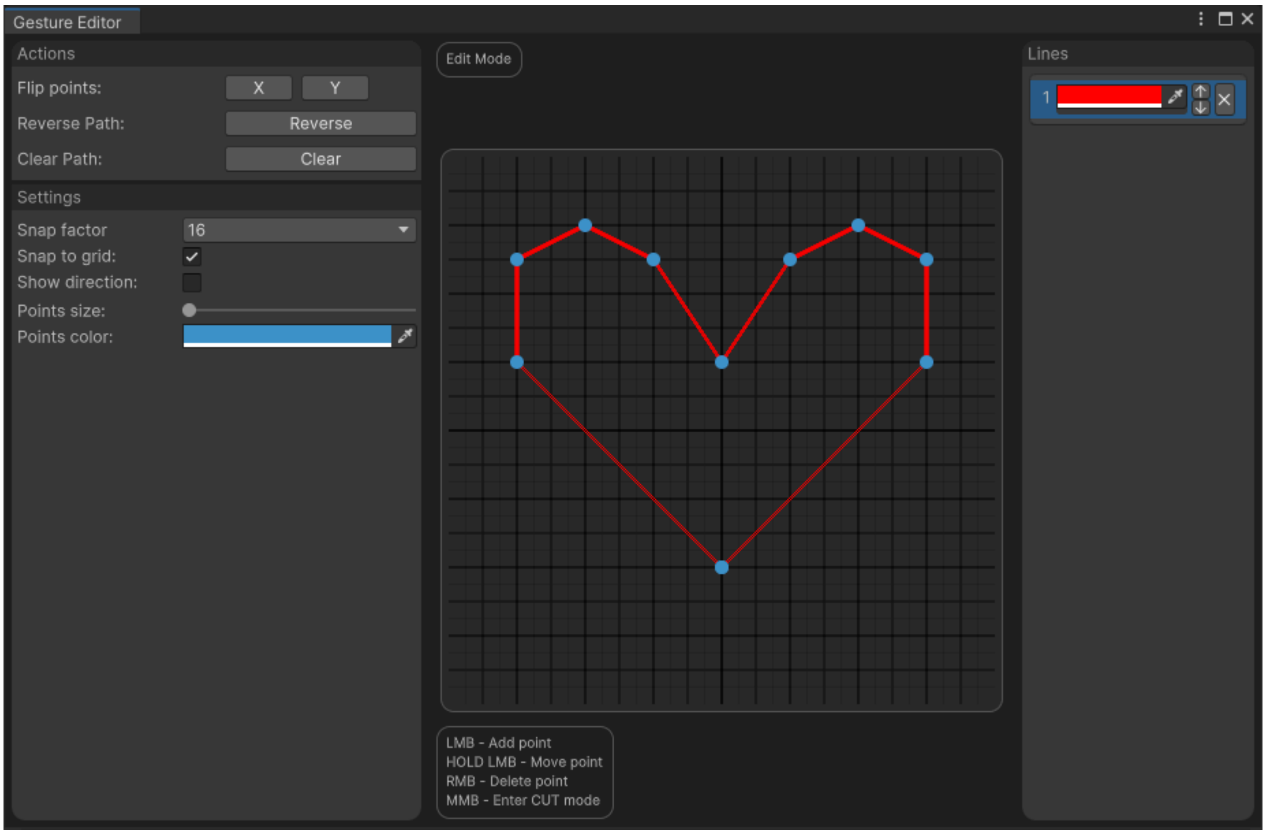
- Added support for direction-invariant gestures;
- Added support for rotation-invariant gestures;
- Updated the gesture pattern editor interface.

Breaking Changes:

- The "SpecificGestureRecognizer" class has been removed.
- Recognizers for different gesture types have been merged into a single recognizer for convenience.

Pattern Editor

The pattern editor is designed for creating and editing gesture pattern paths. The editor window can be opened via the Unity menu under "Tools > Gesture Editor" or by clicking the "Open Editor" button in the prefab's inspector. The open editor displays the currently selected pattern or will remain inactive if no pattern is selected.



Snap factor: 16
Snap to grid: ✓
Show direction: ☐

LMB - Add point
HOLD LMB - Move point
RMB - Delete point
MMB - Enter CUT mode

The open gesture pattern editor window

The editor window is divided into three sections:

- Editor Actions and Settings
- Pattern Path Editor
- Gesture Lines

The Path Editing Area is the core section, enabling the creation of a pattern path by defining points. The editor features three modes, each responsible for specific functions:

- Edit Mode: Add or remove points.
- Drag Mode: Move points.
- Cut Mode: Add points to an existing line.

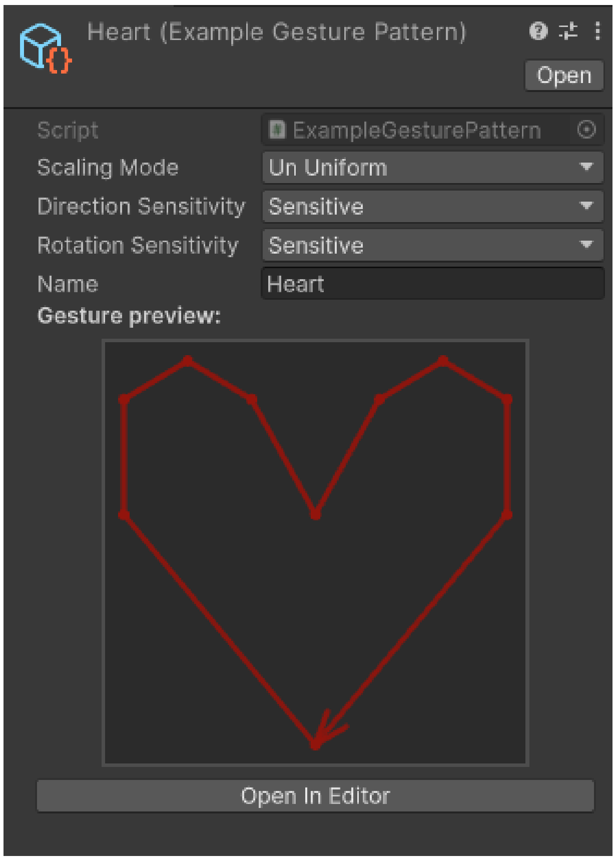
The current mode and input hints are displayed within the editor area.

Gesture Patterns

Gesture patterns are Scriptable Object instances derived from the IGesturePattern interface. You can create custom gesture types by inheriting from GesturePatternBase, adding the necessary data and functions. Your new type will be fully supported by both the editor and the inspector.

```
// You can add "CreateAssetMenu" attribute here
public class ExampleGesturePattern : GesturePatternBase {
    // Derivative classes will work with the editor and inspector
    // Add your data here
    [SerializeField]
    private string _name;
    public string Name => _name;
}
```

The pattern inspector contains gesture settings, any additional data you've added, and displays the gesture path along with a button to open the editor.


Heart (Example Gesture Pattern)

Script: ExampleGesturePattern
Scaling Mode: Un Uniform
Direction Sensitivity: Sensitive
Rotation Sensitivity: Sensitive
Name: Heart
Gesture preview:

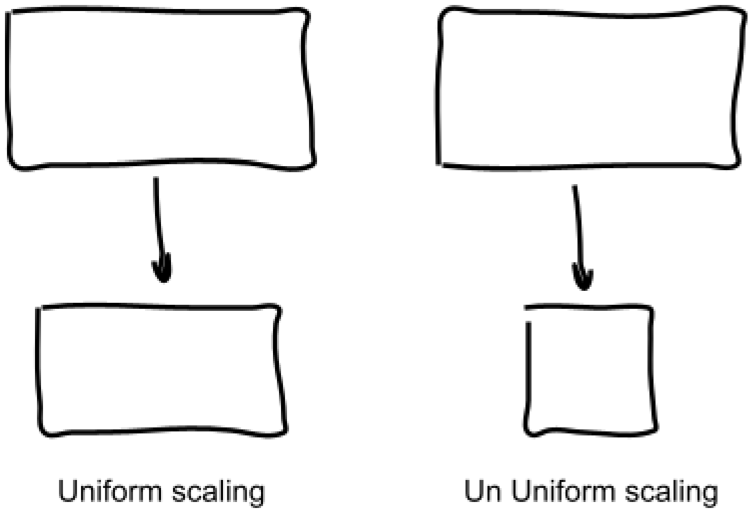
The gesture pattern prefab inspector

By default, gestures have three parameters that determine how they will be compared to the player's gesture input, as well as the gesture path.

The Path of the pattern prefab contains the gesture path represented as a sequence of points. All points in the path are at coordinates from (0;0) to (1;1), and in such a sequence that the first point in the array corresponds to the first point in the gesture, the second to the second, and so on.

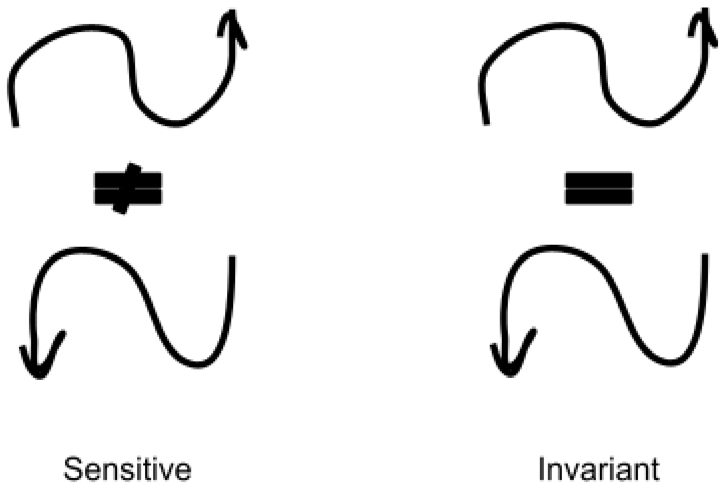
The Scaling Mode determines the method of scaling the gesture when working with the path. It has two modes:

- Uniform: Makes the gesture sensitive to aspect ratio — the gesture will maintain its aspect ratio when resized (e.g., if the gesture is drawn as a square, it won't match a rectangular gesture pattern).
- UnUniform: Makes the gesture invariant to aspect ratio — the gesture will scale to a 1:1 aspect ratio (a square gesture will match a rectangular gesture pattern).



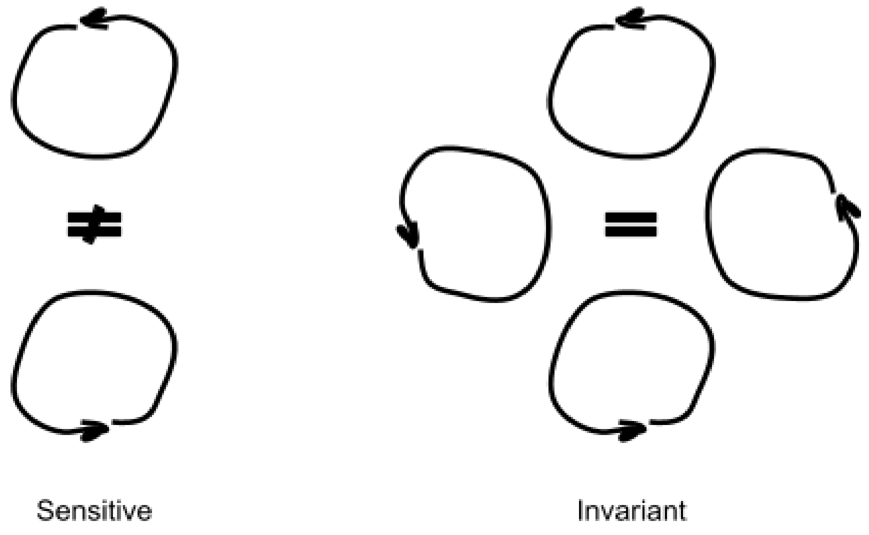
Directional Sensitivity determines the impact of the input direction on gesture recognition. It has two modes:

- Sensitive: The gesture can only be input from one direction.
- Invariant: The gesture can be input starting from either the beginning or the end.



Rotation Sensitivity determines whether the gesture can be rotated. It has two modes:

- Sensitive: The input gesture will match the pattern only if it is drawn without rotation.
- Invariant: The input gesture will match the pattern even if it is rotated.



Record Gesture Path

To record the gesture path, it is necessary to implement the logic for reading the position of the next point on the gesture path being drawn by the player. The asset provides an example of the simplest implementation for recording the path from the screen, inputted using the mouse.

To record the player's gesture, use an object of the GestureRecorder class.

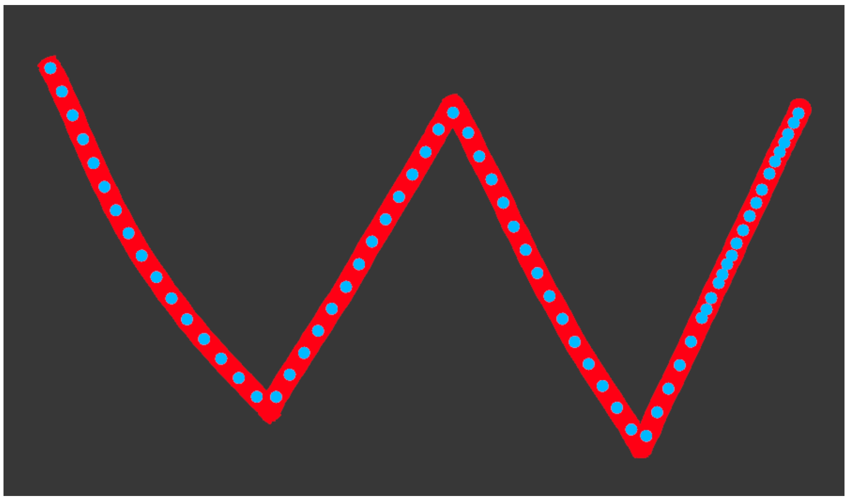
```
var gestureRecorder = new GestureRecorder(254, 0.1f);
```

The constructor of the GestureRecorder class takes two parameters:

- pathMaxLength: int — the maximum size of the array for recording the path.
- newPointMinDistance: float — the required distance between the last point and the new point to record (if the distance is below this threshold, the point will not be recorded).

When recording a gesture, GestureRecorder optimizes the recorded path so that it always fits within the array of limited size. This helps prevent memory overflow caused by allocating large arrays, at the cost of a small loss in the accuracy of the recorded path. Resampling of the recorded path occurs when the allocated array is filled. The higher the pathMaxLength parameter, the less frequently resampling will occur, resulting in higher accuracy for the recorded gesture.

The image below shows the distribution of points when recording a gesture path. You can test the algorithm using the example included in the asset.


The cluster of points on the right side of the gesture represents the newly recorded points that have not yet undergone resampling

Gesture Recognition

To recognize the recorded path, use an object of the GestureRecognizer class.

```
var recognizer = new GestureRecognizer<GesturePattern>(patterns, 128);
```

The constructor of the GestureRecognizer class takes two parameters:

- patterns: IEnumerable<G> — a collection of patterns for recognition.
- resamplePointsNumber: int — the number of points in the patterns and the recorded path after they are prepared for recognition.

When an instance of the GestureRecognizer object is created, the recognizer is set up. The provided patterns will be prepared for recognition: the number of points in the path will be resampled to the value specified in the constructor, and the path will be normalized according to the mode set in the pattern properties. The higher the resamplePointsNumber, the greater the recognition accuracy, but the computational cost of comparisons also increases. The relationship between recognition time and the number of points in the gesture path can be found in the Performance section.

If you need to add a new gesture pattern at runtime, the only way to do so is by recreating the recognizer object.

To recognize the recorded path, use the Recognize or ScheduleRecognition methods:

- Recognize blocks the main thread until recognition is complete and immediately returns the recognition result.
- ScheduleRecognition schedules the recognition using the Unity Job System and returns a JobHandle. Save this handle during the task's execution to track its status

```
private void RecognizeRecordedGesture() {
            // Schedule the recorded path recognition and
            // get a recognition results in the late update

            _recognizeJob =
                _recognizer.ScheduleRecognition(_gestureRecorder.Path);

            // If you want to get the result instantly,
            // use _recognizer.Recognize(_gestureRecorder.Path) method instead
        }
```

Scheduled tasks are executed in the background during the current frame's processing, without blocking the main thread. You can retrieve the result during LateUpdate, when the recognition task is most likely completed.

```
private void LateUpdate() {
      if (!_recognizeJob.HasValue)
          return;

      // Make sure to complete the recognition task
      _recognizeJob.Value.Complete();

      // Get recognition result
      RecognizeResult<ExampleGesturePattern> result = _recognizer.Result;

      // You can set the required precision value for the results.
      // A score is a value in the range of 0 to 1.
      // Where 1 means that the recorded path is
     // exactly the same as the pattern path.
      // 0.8 is a good choice, but you can choose another value.
      if (result.Score >= .8f) {
          // Get the recognized pattern from the result
          ExampleGesturePattern recognizedPattern = result.Pattern;
          _nameController.Set(recognizedPattern.Name);

          Debug.Log($"{recognizedPattern.Name}: {result.Score}");
      }

      _recognizeJob = null;
  }
```

The recognition result can be accessed through the Result property of the recognizer.

Caution: Accessing this property before the scheduled recognition task is completed will lead to unpredictable behavior.

The recognition result is represented by the RecognizeResult structure, which contains two values:

- Score: The similarity level between the input path and the pattern path with the highest match. The value ranges from 0 to 1, where 1 indicates a perfect match.
- Pattern: A reference to the pattern with the highest similarity to the input path.

Due to the nature of the recognition algorithm, recognition always produces a result, even if the input path differs significantly from all registered patterns. To evaluate whether the input path truly matches the recognized pattern, check the similarity level (Score).

```
        // You can set the required precision value for the results.
        // A score is a value in the range of 0 to 1.
        // Where 1 means that the recorded path is
        // exactly the same as the pattern path.
        // 0.8 is a good choice, but you can choose another value.
        if (result.Score >= .8f) {
            // Get the recognized pattern from the result
            ExampleGesturePattern recognizedPattern = result.Pattern;
         }
```

Set a threshold for the similarity Score to filter out results that are weakly similar to the recognized pattern. Scores above 0.8 are generally sufficiently similar to the identified pattern. However, depending on your gesture set and system specifics, you may choose a different value.

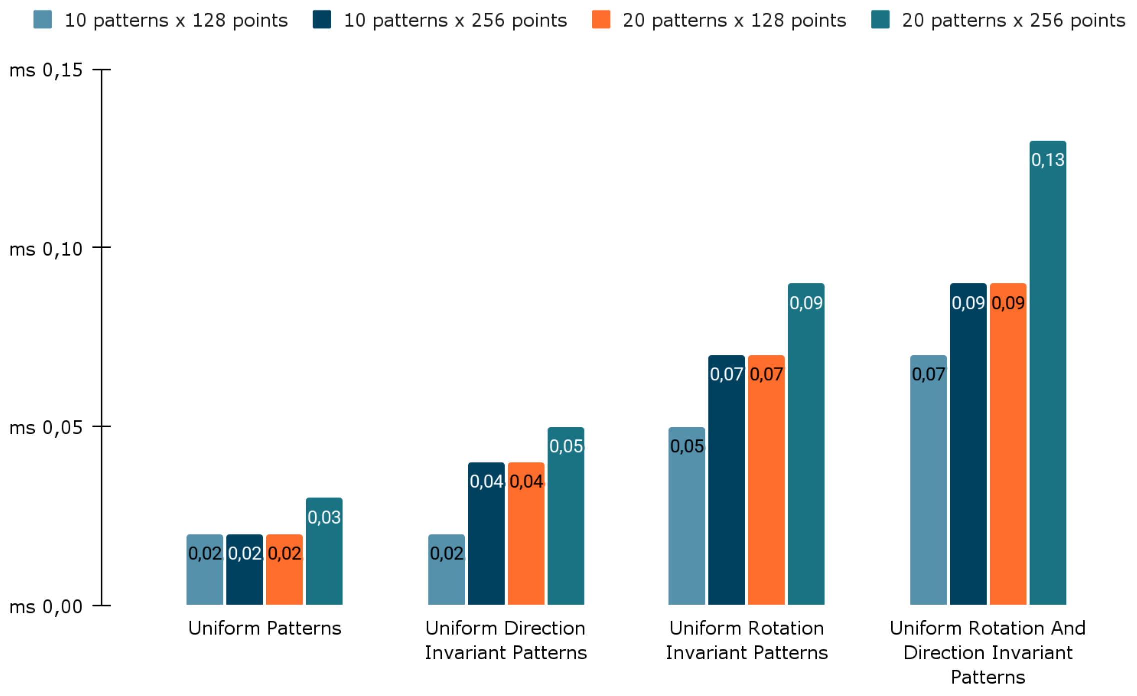
Performance

The performance of the recognition method can vary depending on the parameters of the registered patterns. This is because, based on the pattern settings, the pattern path and the recorded path will be compared using different algorithms.

To evaluate performance, a recognition algorithm benchmark was developed. You can find the test code at the following link: [Tests Repository]. Simply download the source code and add it to your project with the imported asset.

The benchmark is designed to measure the execution time of the gesture path recognition method with gesture patterns featuring different parameters, varying numbers of patterns, and varying numbers of points in the path.

Test system:
CPU: Intel® Core™ i5-10500H @ 2.50GHz
RAM: SODIMM Crucial Ballistix 8GB * 2 DDR4
OS: Windows 11


10 patterns x 128 points
10 patterns x 256 points
20 patterns x 128 points
20 patterns x 256 points
ms 0,15
ms 0,10
ms 0,05
ms 0,00
Uniform Patterns
Uniform Direction Invariant Patterns
Uniform Rotation Invariant Patterns
Uniform Rotation And Direction Invariant Patterns