How to Use Gestures or Create Your Own Gestures

There are two ways to add gesture detection to your Unity-project. The first one is easier and utilizes the KinectManager – a component of the KinectController-game object in the example scenes. It has a list-setting called "Player Common Gestures". Here you can put the gestures that will be detected for each user during the entire game, or the gestures that you need to test quickly.

The second way is to specify user or scene-specific gestures programmatically. You need to create a script that implements KinectGestures.GestureListenerInterface and use its methods for gesture initialization and gesture detection processing. As an example, look at the KinectScripts/Samples/SimpleGestureListener.cs-script. Here is a short description of the GestureListenerInterface's methods:

- UserDetected() – invoked when a new user is detected. It can be used to start the gesture detection.
- UserLost() – invoked when a user is lost. It can be used to clean up or free the allocated resources. You don't need to stop explicitly the detection of gestures added in UserDetected().
- GestureInProgress() - invoked when a gesture start is detected, but the gesture is not yet completed or cancelled. Can be used to report the gesture progress or to process continuous gestures (that never get completed), like for instance zooming, leaning, walking or running.
- GestureCompleted() - invoked when the gesture is completed. You can process the gesture detection here, and decide whether to reset the gesture (i.e. restart its detection) or not.
- GestureCancelled() - invoked, if the gesture gets cancelled. The gesture gets automatically cancelled, when it is not completed within the allowed time frame. You can decide, whether to reset the gesture (i.e. restart its detection) or not.

Currently Recognized Gestures

The following gestures are currently recognized:

- *RaiseRightHand / RaiseLeftHand* – left or right hand is raised above the shoulder and the user stays in this pose for at least 1.0 seconds (pose).
- *Psi* – both hands are raised above the shoulder and the user stays in this pose for 1.0 seconds (pose).
- *Tpose* – the hands are to the sides, perpendicular to the body (T-pose), for 1.0 seconds (pose).
- *Stop* – right hand is down and left hand is slightly to the side, but below the waist, or left hand is down and right hand is slightly to the side, but below the waist (pose).
- *Wave* – right hand is waved left and then back right, or left hand is waved right and then back left.
- *SwipeLeft* – right hand swipes left.
- *SwipeRight* – left hand swipes right.
- *SwipeUp / SwipeDown* – swipe up or down with the left or right hand
- *ZoomIn* - left and right hands are to the front and put together at the beginning, then the hands move apart in different directions (continuous gesture, reports zoom-factor).
- *ZoomOut* - left and right hands are to the front and at least 0.7 meter apart at the beginning, then the hands move closer to each other (continuous gesture, reports zoom-factor).

- *Wheel* - left and right hands are to the front and shoulder size apart at the beginning, then the hands keep the distance and move as if they turn an imaginary wheel counter clockwise, which returns positive angle, or clockwise - negative angle (continuous gesture, reports wheel angle).
- *Jump* – the hip center gets at least 10 cm above its last position within 1.5 seconds.
- *Squat* - the hip center gets at least 10 cm below its last position within 1.5 seconds
- *Push* – push forward with the left or right hand within 1.5 seconds.
- *Pull* - pull backward with the left or right hand within 1.5 seconds.
- *ShoulderLeftFront* – move the left shoulder to the front.
- *ShoulderRightFront* – move the right shoulder to the front.
- *LeanLeft* - lean left with the whole body (continuous gesture, reports lean angle).
- *LeanRight* – lean right with the whole body (continuous gesture, reports lean angle).
- *LeanForward* – lean forward with the whole body (continuous gesture, reports lean angle).
- *LeanBack* – lean back with the whole body (continuous gesture, reports lean angle).
- *KickLeft* – move the left foot to the front.
- *KickRight* – move the right foot to the front.
- *Run* – checks if the left knee is at least 10cm above the right knee, then – if the right knee is at least 10 cm above the left knee, then – back to the left knee check (continuous gesture, reports progress).
- *Raised right horizontal left hand* – the right hand is raised above the shoulder, while the left hand stays horizontal at shoulder level (courtesy of Andrzej W).
- *Raised left horizontal right hand* – the left hand is raised above the shoulder, while the right hand stays horizontal at shoulder level (courtesy of Andrzej W).
- *TouchRightElbow* – the fingers of the left hand touch the right elbow for at least 1.5 seconds.
- *TouchLeftElbow* – the fingers of the right hand touch the left elbow for at least 1.5 seconds.

How to Add Your Own Gestures

Here are some hints on how to add your own gestures to the Kinect gesture-detection procedure. You need some C# coding skills and a bit of basic understanding on how the sensor works. It reports the 3d-coordnates of the tracked body parts in the Kinect coordinate system, in meters.

To add detection of custom gesture, open Assets/KinectScripts/KinectGestures.cs. Then:

1. Find the Gestures-enum. First you need to add the name of your gesture at the end of this enum.
2. Find the CheckForGesture()-function. There is a long switch() there, and its cases process the detection of each gesture, defined in the Gestures-enum. You need to add a case for your gesture at the end of this switch(), near the end of the script. There you will implement the gesture detection.
3. Look at the processing of some simple gestures, like RaiseLeftHand, RaiseRightHand, SwipeLeft or SwipeRight.
4. As you see, each gesture has its own internal switch() to control the gesture's current state. Each gesture is like a state machine with numerical states (0, 1, 2, 3…). Its current state along with other data, is stored in an internal structure of type GestureData. This data-structure is created for each gesture that needs to be detected in the scene.

5. The initial state of each gesture is 0. At this state, the code needs to detect if the gesture is starting or not. To do this, it checks and stores the position of a joint, usually the left or right hand. If the joint position is suitable for a gesture start, it increments the state. At the next state, it checks if the joint has reached the needed position (or distance from the previous position), usually within a time interval, let's say within 1.0 - 1.5 seconds.
6. If the joint has reached its target position (or distance) within the time interval, the gesture is considered completed. Otherwise - it is considered cancelled. Then, the gesture state may be reset back to 0 and the gesture-detection procedure will start again.

To add detection of your own gestures, first try to understand how relatively simple gestures, like RaiseHand or Swipes, work. Then find a gesture similar to the one you need. Copy and modify its code, as to your needs.

For more information look at this tip: https://rfilkov.com/2015/01/25/kinect-v2-tips-tricks-examples/#t44

Note: The KinectGestures-class is Unity component now. This means you must add it to the scenes, where you need gesture recognition. You may also extend the class, when you want to add recognition of your own gestures. This will prevent overwriting your code, when a new version of the K2-asset is released.

More Information, Support and Feedback

Online Documentation: https://ratemt.com/k2docs/
Tip and Tricks: http://rfilkov.com/2015/01/25/kinect-v2-tips-tricks-examples/
Web: http://rfilkov.com/2014/08/01/kinect-v2-with-ms-sdk/

Contact: http://rfilkov.com/about/#contact (please mention your invoice number from Asset store)
Twitter: https://twitter.com/roumenf