



BLACKCHERRY GESTURE LIBRARY FOR UNITY 3D

OVERVIEW

The Gestures Library is a set of C# classes that can be used by a Unity program to be notified of different finger gestures in Unity-based iPhone, iPad, and Android applications. Most of the gestures can also be used for mouse-based gestures on Apple systems or Windows, and while developing in the Unity Game window.

A gesture class is put on a GameObject to provide the GameObject with the different gestures that are in the library.

The following summarizes the different gestures that are available.

Drag	Move the object relative to some number of fingers.
Line Swipe	Using one finger to swipe different shapes.
Long Press	Press and hold some number of fingers a set amount of time
Pinch	Scale the object bigger and smaller using two fingers.
Rotate	Rotate the object on any axis using one or two fingers.
Slice	Swipe one finger across the object starting and ending off the object.
Swipe	Swipe some number of fingers in a single direction.
Tap	Tap one finger a set number of times.
Touch	Basic finger down, move, and up.

Processing primarily consists of evaluating what the finger (or mouse) is doing and how it relates to the current target gesture. It is constantly evaluating the situation to reject it or see if it has finished. This document will discuss gesture functionality with this concept.

USING THE TOOL

Each gesture type is a class that can be added to a game object. The class will send messages to the game object when gesture related activities occur, and in some cases interact with the



object changing its position, rotation and scaling.

Each gesture has a set of input properties that control its behaviour. Common controlled behaviours include the fingers being on or off the object and which type of gesture to recognize. When a gesture happens, the class will send related messages to the object that always include the gesture class itself. On the returned gesture class there are return properties that your program can use.

The Gesture Tool comes with a sample scene for each type of gesture. The sample allows you to set the input properties and then perform the gesture to learn how the properties work. When a gesture happens the sample will show some of the relevant return properties.

The typical development scenario would be to first identify the GameObjects that need gestures and which individual gestures are required. The object must have a collider if the gesture needs to interact with the object, for instance determining if the gesture is over or not over the object.

Drop the related gesture classes on to the GameObjects. Gesture class properties can be updated using the Unity inspector or from your code.

Create a class to receive gesture messages and drop it on the GameObject with the gesture. The target object of each gesture defaults to the gesture class' home GameObject but it can be changed to another arbitrary GameObject. All gestures also allow specification of a second non-related GameObject that messages will also be sent to.

If you wanted to update input properties from your code, you need to get the gesture class. You can do this with code like this:

```
TouchGesture myTouchGesture = this.gameObject.GetComponentInChildren<TouchGesture>();
```

This example finds the TouchGesture class on the same GameObject.

Your code needs to define message receptor methods for the gestures you are using. Each gesture has a set of messages sent when a gesture happens. In case of the Touch Gesture, the messages are GestureStartTouch, GestureMoveTouch, and GestureEndTouch. In your class they would be defined as follows.

```
void GestureStartTouch(TouchGesture gesture) {  
  
Debug.Log("GestureStartTouch " + touchGesture.finger.startPosition);  
  
}  
  
void GestureMoveTouch(TouchGesture gesture) {
```



```
Debug.Log("GestureMoveTouch " + touchGesture.finger.position);  
  
}  
  
void GestureEndTouch(TouchGesture gesture) {  
  
Debug.Log("GestureEndTouch " + touchGesture.finger.endPosition);  
  
}
```

The gesture class sending the message includes itself as the message parameter. The class can be used to see different attributes relating to that gesture such as finger position, swipe direction or transformation amount.

You only need to define the messages for the ones you want to use. For example with touch, you could just add GestureStartTouch to know whenever a finger touch happens. Some samples use the touch class to find out when a gesture may be happening to adjust the interface.

You can also add a Gesture class to code. An example adding a Touch Gesture.

```
void Start() {  
  
this.gameObject.AddComponent("TouchGesture");  
  
}
```

When you run the samples watching the inspector, you may notice a TouchGesture appearing during some samples. It is being added from code for the samples that need to know when a finger down happens.

In the BC Gesture Library / _help folder there are two files called GestureEvents – one of C# and one for JavaScript. They contain skeleton code you can copy so you can receive gesture events in your code.

Using the Gesture Library is that simple.

Note: The BC Gesture Library folder should be placed in the Plugins folder to ensure its classes are there for the compiler, mandatory for JavaScript.

THE GESTURES

This section documents some concepts and the parameters for each gesture class.

GESTURE CONCEPTS

The Gesture classes use some common enumerated type for defining properties.

FingerLocation

FingerLocation is used to indicate the required location of the finger relative to the object. Possible values:

Always	The finger is anywhere.
Over	Finger is over the object. There must be a collider to perform this.
Not Over	Finger is not over the object. There must be a collider to perform this.
Atleastoneover	At least one finger is over the object. This is not used on single finger gestures (Line, Slice, Tap, and Touch) where it will performed as Over.

FingerCountRestriction

FingerCountRestriction is used to restrict the number of fingers the gesture will use.

Any	Any number of fingers is acceptable.
One	Only one finger can be used.
Two	Only two fingers can be used.
Three	Only three fingers can be used.
Four	Only four fingers can be used.
Five	Only five fingers can be used.
OneOrTwo	Only one or two fingers can be used.

OneOrTwoOrThree	Only one or two or three fingers can be used.
TwoOrThree	Only two or three fingers can be used.

Note: On Apple touch devices four and five finger gestures are taken by the operating system for its own functionality. The swipe gestures cannot return four and five finger gestures on Apple touch devices.





Swipes and SwipeDirection





The Gestures Library uses swipes consistently across all gestures. A swipe has a direction, a start, an end, and a distance. The distance is only used in Line Gestures.

Performing a swipe must be done in a continuous motion. Any major variation off the swipe line will cancel the swipe.

Swipes can be done in 8 different directions. This reduces the swipes and shapes that can be defined, but greatly increases the likelihood of the gestures being identified successfully.




The 8 directions are defined in a `SwipeDirection` enumeration as follows. They are described using a clock face analogy using the hour hand. These types will be returned by swipes and used as swipe segments in line shapes. The naming is relative to the up direction being zero degrees.




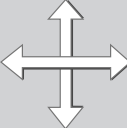
Up		Vertically up, 12 o'clock.
Plus45		45 degrees right of up, between up and right, between 1 and 2 o'clock.
Right		Horizontally right, 3 o'clock
Plus135		135 degrees right of up, between right and down, between 4 and 5 o'clock.

Down		Vertically down, 6 o'clock.
Minus135		135 degrees left of up, between down and left, between 7 and 8 o'clock.
Left		Horizontally left, 9 o'clock.
Minus45		45 degrees left of up, between left and up, between 10 and 11 o'clock.

Every direction has friendly directions which are the directions on either side of a direction. For example. Plus135 and Minus135 are the friendly directions to Down.

The SwipeDirection enumerated type also contains a set of values for specifying what swipes will be accepted in addition to the specific swipe directions. These values allow multiple swipe directions to be accepted for a swipe.

Any		Any of the 8 swipe directions.
LeftDiagonal		Minus45 or Plus135
RightDiagonal		Plus45 or Minus 135

Vertical		Up or Down
Horizontal		Left or Right
AnyCross		X, LeftDiagonal or RightDiagonal
AnyPlus		+, Vertical or Horizontal
None		Matches nothing. Can be used in code to delineate a null value inside a direction.

Finger and finger used

The Finger class has a set of properties that describe a finger (or mouse). Some Gestures will return a list of Finger classes. Every gesture has a finger property returned that is a Finger. Finger is the finger that caused the message to be sent, in the case of multiple fingers this can be unpredictable. The Finger class has the following properties.

isDown	Boolean indicating if the finger is down
hasMoved	Boolean indicating if the finger has moved in the gesture
position	The current position of the finger, screen Vector2.
startPosition	The start position of the finger, screen Vector2.
endPosition	The end position of the finger, screen Vector2.
Index()	The zero based finger number. On a device, this is the finger number within the fingers. On a mouse based computer, this is the button number pressed.

startTime	The time the first finger down occurred. Float with Time.time.
downAndMovingStartTime	The time the first move occurred. Float with Time.time.

Class has additional properties that are used by the gesture system.

Swipe Segment

SwipeSegment is a class used by swipes to represent a straight section of a swipe. In the case of SwipeGesture and SliceGesture, they are a swipe with one segment. In LineGesture the swipe can have multiple turns to define a shape consisting of a SwipeSegment list. A SwipeSegment list is created by chaining them together using next and previous references. More about this in the LineGesture section.

A SwipeSegment has the following properties.

direction	The swipe segment direction. See SwipeDirection.
distance	The segment length in pixels. Float, positive rational number.
velocity	The finger velocity creating the segment. Float, positive rational number in pixels per second.
fingers	A list of the fingers used for the segment. List <Finger>, see Finger.
startPosition	The screen position of the segment start. Vector2.
endPosition	The screen position of the segment end. Vector2.
startTime	The time the segment was started. Float with Time.time.
endTime	The time the segment was ended. Float with Time.time.
previous	The previous SwipeSegment in the segment list. Null at end.
next	The next SwipeSegment in the segment list. Null at end.

Drag Gesture

The Drag Gesture can be used to move an object relative to some number of fingers. Using a single finger, the drag position is the finger. With multiple fingers, the drag position will be calculated as the middle of the down fingers at any given time. The drag can happen maintaining the relative position to the object or centering the object on drag position.

Class: DragGesture

Input Properties

doDrag	Perform the drag actions on the object, Boolean. If false the gesture will only send drag messages and not move the object.
dragPosition	How the object will drag relative to the fingers. One of: Relative – drag maintaining related position of down Centred – drag under the drag position NoDrag – do not perform drag but still send the messages
fingerLocation	Required location of the finger relative to the object. See <i>FingerLocation</i>.
restrictFingerCount	Restrict the number of fingers a gesture can use. See <i>FingerCountRestriction</i>.
targetCollider	The collider that the gesture will interact with. If not set the collider on the game object where the gesture resides will be used as a default.
targetMessageObjects	An array of gameobjects where gesture messages will be sent. If not set then messages will be sent to the game object where the gesture resides as a default.
alternateCamera	All gesture default to using the Camera marked Camera.main. If there is no Camera.main then a Camera will be used. alternateCamera overrides this selection and sets the camera to be used. It is important the right camera to be used for any gestures that interact with game object colliders.
topColliderOnly	A boolean controlling if FingerLocation collider checks will only look for the first collider from the camera or look through all colliders to find the gesture's collider.
restrictDirection	Optionally restrict the direction of the drag to only the X or Y directions. Uses XYRestriction type. XYRestriction - one of AllDirections, XDirecton or YDirection.

restrictScreenMin	If less than or equal to one, this specifies minimum screen percent when restrictDirection is XDirection or YDirection. If greater than one it specifies a minimum screen position. The object will keep to the right or below this value. A float.
restrictScreenMax	If less than or equal to one, this specifies maximum screen percent when restrictDirection is XDirection or YDirection. If greater than one it specifies a maximum screen position. The object will keep to the left or above this value. A float.

Messages

All messages return a DragGesture object.

GestureStartDrag	Sent when a drag starts.
GestureMoveDrag	Sent for each move during a drag.
GestureEndDrag	Sent when the drag ends.

Return Properties

dragFingerCount	The number of fingers used in the gesture, Integer greater than zero.
startPoint	World Vector3 coordinate object started at.
endPoint	World Vector3 coordinate object ended at.

Line Gesture

The Line Gesture is used to swipe shape-based Gestures such as numbers, letters and geometrical shapes using one finger. The LineGesture is initialized to have no gestures. You must add specific line gestures to the search list and the LineGesture will look for those shapes in the order specified.



Each shape is a gesture broken into a set of components. The gesture is a finger down, with straight movements to corners that are 90 or 45 degrees that lead to other straight sections and ends with a finger up. A component is a straight section within the gesture and has a swipe direction, positions, and distance.

For example, the letter N can be created with three segments of swipe directions Up, Plus135 and Up each having about the same length.

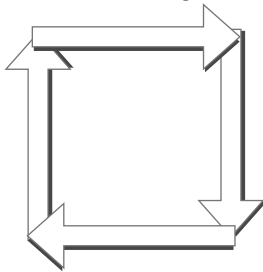


The gesture can be drawn in both directions with backwards using the segment's direction reversed going from end to start. The N backwards would be Down, Minus45, Down.

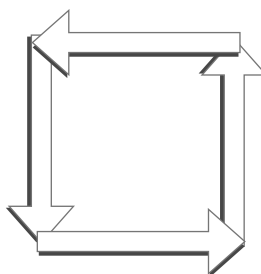
Each component will be a single direction or have an optional direction. When matching, either of the directions will be used. A nice example for this is the letter M. The outside segments could be vertical or slanted.

A Line Gesture can be a closed shape such as a square or a triangle. In this case, the gesture has components that allow the gesture path to finish at the start and the gesture is marked closed. This forces the start and end finger positions to be very near each other. Closed line gestures can be done forward or reverse. Additionally, a closed gesture can be started on any vertice or in the middle of any side.

In this example the box is defined by Left, Down, Right, Up and marked closed.

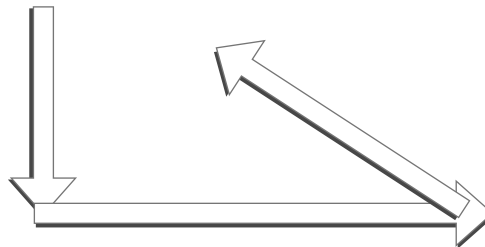


**Start and stop on any corner
going either direction**



**Start in the middle of a side
completing the box**

The relative length of segments are defined by ratios. Each segment is assigned an integer number that defines its size as a ratio relative to the other segments. When the gesture engine calculates, it allows the sizes to adjust to the correct size within a range. Specify zero to indicate a segment can be any relative size, hence only direction matters. So, a square has segment ratios 1, 1, 1, 1 which means that all sides are the same size. A rectangle could be defined with segment ratios of 1, 0, 1, 0 which is to say two opposite sides are the same and the other opposite sides can be any length. This combined with Left, Down, Right, Up causing right angles will be a rectangle. The letter L (Down, Left) could be done with segment length 2, 1 saying the Right is half the Down.



This gesture could be defined as Down with ratio 1, Left with ratio 3, Minus 45 with ratio 2.

Relative size can also be specified by indicating a segment is bigger than a ratio, biggest, smaller than a ratio, and smallest.

Gesture possibility is calculated in the order that the gestures are in the gesture list. This should be taken into account when two gestures replicate each other in direction but not in size or with optional directions. The first gesture matched will be returned. For example, a square is a rectangle. If the rectangle was located on the list before the square, then the square would never be returned since the rectangle would always be matched before the square. Putting square before rectangle would allow squares to be found.

A line gesture can be defined to contain multiple lines. Relationship properties are specified to indicate how the lines interact with each other. For example consider the letter T. T is two straight lines with the start of the vertical (biggest) line attached to the middle of the (smallest) horizontal line .

Line identification relates to the amount of effort the code will pursue when evaluating line gestures. The **LineIdentification** enumerated type has the following values:

Precise	The line segments drawn must match what the user defines segment for segment. This setting is not very useable.
Clean	The line segments drawn basically match what the user defines. Very small segments are ignored. Multiple segments that are friendly will be coalesced to match a target segment only if the line direction made from the start point on a primary segment to the end of the coalesced segment matches the target segments direction(s). Precise is included in Clean.
Sloppy	<p>Sloppy gives everything more leeway. Segment directions can match the friendly directions of the target segment bringing in the friendly directions of the friendly ones to do so. Matching locations and sizes are greatly relaxed. Clean is included in Sloppy.</p> <p>Sloppy is very good at matching lines gestures done in many different ways but may have unintended matches on similar gestures. It is recommended on a touch screen. Clean may be sufficient on a mouse-based interface.</p>

Class: LineGesture

Input Properties

restrictLineSwipe Direction	Accepted line gestures direction. One of: Forward – the direction of the components are defined. Backward – reverse direction that the components are defined. Either – forward or backward Anywhere – forward or backward, or on a closed gesture at any start/end point or in the middle of any side.
startsOnObject	Required location of the down finger relative to the object. See <i>FingerLocation</i>.

endsOnObject	Required location of the up finger relative to the object. See <i>FingerLocation</i>.
returnSwipeAlways	Boolean causing all gesture attempts to be returned regardless of matching success. This is provided for testing gestures.
targetCollider	The collider that the gesture will interact with. If not set the collider on the game object where the gesture resides will be used as a default.
targetMessageObjects	An array of gameobjects where gesture messages will be sent. If not set then messages will be sent to the game object where the gesture resides as a default.
lineIdentification	LineIdentification specifies the level of line matching that will be performed. One of: Precise, Clean or Sloppy.
matchPositionDiff	A float specifying the permissible distance for matched line positions. In screen coordinate system pixels.
matchLengthDiffPercent	A float specifying the maximum difference percentage in segment lengths. Float where percentages under 100% are less than one, 100% is one.
maxTimeBetween-Lines	The maximum time that will be waited for between line segments. Only used if multiple segment line types are selected. This value must be long enough but not too long since types with fewer segments will wait this long before being evaluated. A float in seconds.
AddLineFactoryType method	<p>Adds a predefined line type to be looked for by the gesture.</p> <pre>public void AddLineFactoryType(LineFactory.LineType lineType, bool clearList)</pre> <p>lineType : provide a linetype from the LineType enumerated type. ClearList: Boolean that will clear list making the added type the only one. If false multiple types can exist.</p>
HaveLineFactoryType method	<p>Query the defined types about the existence of a type.</p> <pre>bool HaveLineFactoryType(LineFactory.LineType lineType)</pre> <p>lineType: A LineType specifying the line type to be queried. Return a Boolean indicating whether or not the type exists in the type list.</p>
RemoveLineFactory-Type method	<p>Remove a line type from the searched line types list.</p> <pre>RemoveLineFactoryType(LineFactory.LineType lineType)</pre>



alternateCamera	All gesture default to using the Camera marked Camera.main. If there is no Camera.main then a Camera will be used. alternateCamera overrides this selection and sets the camera to be used. It is important the right camera to be used for any gestures that interact with game object colliders.
topColliderOnly	A boolean controlling if FingerLocation collider checks will only look for the first collider from the camera or look through all colliders to find the gesture's collider.

Messages

All messages return a LineGesture object.

GestureLineSwipe	Sent when a line gesture happens.
GestureLineSwipeFailure	Sent when a potential line gesture evaluation fails. This is useful if you are tracking finger movements relative to the line gesture.

Return Properties

swipeSegments	A list of the segments the user performed. Given as list of SwipeSegment objects. See SwipeSegment. If multiple lines are defined they are returned in swipeList.
swipedLineType	A LineFactory.LineType indicating the line swipe done. If returnSwipeAlways is true, then a type of None is returned if no swipe was matched.
finger	A Finger object describing the finger that was used. See Finger Used.
swipeList	List<SwipeSegmentList>, a list of the segment list that the user defined.
performingSwipe	A Boolean indicating whether or not a line swipe is currently happening.
lineIdentificationUsed	A LineIdentification value indicating the line identification used on a successful line swipe.

Long Press Gesture

The Long Press Gesture is used to detect the holding down of some number of fingers for a specified amount of time.

Class: LongPressGesture

Input Properties

longPressTime	The number of seconds the press must hold until the message is sent. Float, positive rational number in seconds.
maxPressDistance	<p>The distance in pixels the finger is permitted to move and still be considered motionless. A mouse is very easy to hold still but fingers on a device tend move a little while held still.</p> <p>On a device a value of around 20 pixels is recommended. Float, a positive rational number representing pixels.</p>
fingerLocation	Required location of the fingers relative to the object. See <i>FingerLocation</i>.
restrictFingerCount	Restrict the number of fingers the gesture can use. See <i>FingerCountRestriction</i>.
targetCollider	The collider that the gesture will interact with. If not set the collider on the game object where the gesture resides will be used as a default.
targetMessageObjects	An array of gameobjects where gesture messages will be sent. If not set then messages will be sent to the game object where the gesture resides as a default.
alternateCamera	All gesture default to using the Camera marked Camera.main. If there is no Camera.main then a Camera will be used. alternateCamera overrides this selection and sets the camera to be used. It is important the right camera to be used for any gestures that interact with game object colliders.
topColliderOnly	A boolean controlling if FingerLocation collider checks will only look for the first collider from the camera or look through all colliders to find the gesture's collider.

Messages

All messages return a LongPressGesture object.

GestureLongPress	A long press has occurred.
-------------------------	----------------------------

Return Properties

timeDifference	The time the finger was down. Float, positive rational in seconds.
FingerCount	The number of fingers used in the gesture, Integer.
finger	A Finger object describing the finger that was used. The location can be found in fingerUsed.endPosition. See Finger Used.
fingers	Array of finger objects giving multiple fingers when finger count is greater than 1.

Pinch Gesture

The Pinch Gesture uses two fingers to scale down and scale up the size of the object. This gesture modifies the Scale property on the Transform of the object. Scaling is performed in the orientation of the fingers. Vertical pinching will change the height, horizontal pinching will change the width and diagonal pinching will change height and width equally.

A mouse wheel can be used to pinch vertically.

The pinch gesture uses PinchDirection enumerated type to define pinch direction.

PinchDirection is one of:

All	any of the directions
Vertical	Up and down
LeftDiagonal	Minus45 and Plus135
Horizontal	Left and right
RightDiagonal	Plus45 and Minus135

Class: PinchGesture

Input Properties

doPinch	Perform the scaling actions on the object, Boolean. If false the gesture will only send pinch messages and not alter the object. Pinch values are sent so your code can implement the pinch if desired.
keepAspectRatio	Keep the aspect ratio of the object during scaling, Boolean. If true then all scaling will maintain height and width ratio of the object. If false then vertical and horizontal pinching will distort the shape of the object.
fingerLocation	Required location of the fingers relative to the object. See FingerLocation.
pinchAction	The pinch in or out scaling that can be done One of: Both – Close and Open Close – scale smaller Open – scale bigger
restrictDirection	The pinch directions that will be accepted. See PinchDirection.
pinchScaleFactor	Numeric scaling factor is multiplied on the scaling to control the amount that the object is scaled relative to the motion of the fingers. Float, positive rational number.
targetCollider	The collider that the gesture will interact with. If not set the collider on the game object where the gesture resides will be used as a default.
targetMessageObjects	An array of gameobjects where gesture messages will be sent. If not set then messages will be sent to the game object where the gesture resides as a default.
alternateCamera	All gesture default to using the Camera marked Camera.main. If there is no Camera.main then a Camera will be used. alternateCamera overrides this selection and sets the camera to be used. It is important the right camera to be used for any gestures that interact with game object colliders.
topColliderOnly	A boolean controlling if FingerLocation collider checks will only look for the first collider from the camera or look through all colliders to find the gesture's collider.

Messages

All messages return a PinchGesture object.

GestureStartPinch	Sent when a pinch starts.
GestureMovePinch	Sent for each move during a pinch.
GestureEndPinch	Sent when the pinch ends.

Return Properties

pinchDirection	The pinch direction that is being done. <i>See PinchDirection.</i>
pinchMagnitudeDelta	The difference in pixels of the pinch. Float, rational number. A positive value is the two fingers moving apart, negative is closer. This value which pinchDirection defines the pinch.
pinchFinger1	A Finger object describing one of the fingers being used. <i>See Finger.</i>
pinchFinger2	A Finger object describing the other finger being used. <i>See Finger.</i>

Rotate Gesture

The Rotate Gesture uses any number of fingers to rotate the object on one of the X, Y or Z axis. This gesture modifies the Rotation property on the Transform of the object.

Class: RotateGesture

Input Properties

doRotate	Perform the rotation actions on the object, Boolean. If false the gesture will only send rotate messages and not alter the object. Rotate values are sent so your code can implement the rotation if desired.
fingerLocation	Required location of the fingers relative to the object. See FingerLocation.

rotateAxis	The axis the rotation should use One of: X – rotate on the X axis Y – rotate on the Y axis Z – rotate on the Z axis
restrictFingerCount	Restrict the number of fingers the gesture can use. See FingerCountRestriction.
targetCollider	The collider that the gesture will interact with. If not set the collider on the game object where the gesture resides will be used as a default.
targetMessageObjects	An array of gameobjects where gesture messages will be sent. If not set then messages will be sent to the game object where the gesture resides as a default.
alternateCamera	All gesture default to using the Camera marked Camera.main. If there is no Camera.main then a Camera will be used. alternateCamera overrides this selection and sets the camera to be used. It is important the right camera to be used for any gestures that interact with game object colliders.
topColliderOnly	A boolean controlling if FingerLocation collider checks will only look for the first collider from the camera or look through all colliders to find the gesture's collider.

Messages

All messages return a RotateGesture object.

GestureStartRotate	Sent when a rotate starts.
GestureMoveRotate	Sent for each move during a rotate.
GestureEndRotate	Sent when the rotate ends.

Return Properties

rotationAngleDelta	The angle change represented by the change in direction. Float, rational number. A positive value is clockwise, negative counter clock-wise.
---------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------

rotateFinger1	A Finger object describing one of the fingers being used. See Finger.
rotateFinger2	A Finger object describing the other finger being used. See Finger.

Slice Gesture

The Slice Gesture uses one finger to swipe a straight line across the object. The swipe must start off the object, cross the object and finish off the object. The position and direction of the slice is returned.

Class: SliceGesture

Input Properties

restrictDirection	The swipe directions that will be accepted for the slice. See SwipeDirection.
targetCollider	The collider that the gesture will interact with. If not set the collider on the game object where the gesture resides will be used as a default.
targetMessageObjects	An array of gameobjects where gesture messages will be sent. If not set then messages will be sent to the game object where the gesture resides as a default.
alternateCamera	All gesture default to using the Camera marked Camera.main. If there is no Camera.main then a Camera will be used. alternateCamera overrides this selection and sets the camera to be used. It is important the right camera to be used for any gestures that interact with game object colliders.
topColliderOnly	A boolean controlling if FingerLocation collider checks will only look for the first collider from the camera or look through all colliders to find the gesture's collider.

Messages

All messages return a SliceGesture object.

GestureSlice	A slice has happened on this object.
---------------------	--------------------------------------

Return Properties

sliceDirection	The swipe direction the slice was in. See SwipeDirection.
sliceStartPosition	A global position on the object where the slice started, Vector3.
sliceEndPosition	A global position on the object where the slice ended, Vector3.
fingerCount	Number of fingers used, Integer.
finger	A Finger object describing the finger that was used. See Finger Used.
fingers	Array of finger objects giving multiple fingers when finger count is greater than 1.

Swipe Gesture

The Swipe Gesture is used to swipe a straight line with some number of fingers. A swipe has direction with down and up positions relative to the object.

Note: On Apple touch devices four and five finger gestures are taken by the operating system for it's own functionality. The swipe gestures cannot return four and five finger gestures on Apple touch devices.

Class: SwipeGesture

Input Properties

restrictDirection	The swipe directions that will be accepted for the swipe. See SwipeDirection.
restrictFingerCount	The number of required fingers.. See FingerCountRestriction.
startsOnObject	Required location of the down finger relative to the object. See FingerLocation.

endsOnObject	Required location of the up finger relative to the object. See FingerLocation.
targetCollider	The collider that the gesture will interact with. If not set the collider on the game object where the gesture resides will be used as a default.
targetMessageObjects	An array of gameobjects where gesture messages will be sent. If not set then messages will be sent to the game object where the gesture resides as a default.
alternateCamera	All gesture default to using the Camera marked Camera.main. If there is no Camera.main then a Camera will be used. alternateCamera overrides this selection and sets the camera to be used. It is important the right camera to be used for any gestures that interact with game object colliders.
topColliderOnly	A boolean controlling if FingerLocation collider checks will only look for the first collider from the camera or look through all colliders to find the gesture's collider.
minGestureLength	Reject swipes that are too short using this value; float.
maxTime	Reject swipes that are take to long; float in seconds

Messages

All messages return a SwipeGesture object.

GestureSwipe	A swipe has happened on this object.
GestureSwipeMove	A swipe move has happened. The gesture has the direction of the move.

Return Properties

swipeDirection	The swipe direction of the swipe. See <i>SwipeDirection</i>.
pressFingerCount	The number of fingers used in the gesture, Integer.
sliceStartPosition	A global position on the object where the slice started, Vector3.
sliceEndPosition	A global position on the object where the slice ended, Vector3.
swipeSegment	A SwipeSegment object describing the swipe which is a single segment. See <i>SwipeSegment</i>.
fingerCount	<i>Number of fingers used, Integer.</i>

finger	A Finger object describing the finger that was used. <i>See Finger Used.</i>
fingers	Array of finger objects giving multiple fingers when finger count is greater than 1.

Tap Gesture

The Tap Gesture is triggered after a set number of single finger taps has occurred relative to the object. A tap is a finger down and up in the same position within a short amount of time. The tap can also be configured to allow a move with the start, move and end on or off the object which is useful to perform a button click.

The button click can be created by setting taps property to one, maxTapDistance to a big number like 9999, enforceStationary is false, startsOnObject and endsOnObject both to Over and movesOnObject to Always.

Class: TapGesture

Input Properties

taps	The number of taps required. Integer greater than zero.
maxTimeBetween-Taps	The maximum amount of time in seconds between two taps for the second tap to be considered in the same tap count. Float, rational number greater than zero representing seconds.
maxTapDistance	The maximum distance between tap locations. Each tap must be no further than this from the previous tap. Float, rational number greater than zero.
tapRateTapsCount	The number of taps included in the tap rate calculation. The tap rate is the average time per tap across tapRateTapsCount taps. Integer, greater then zero.
targetCollider	The collider that the gesture will interact with. If not set the collider on the game object where the gesture resides will be used as a default.
targetMessageOb-jects	An array of gameobjects where gesture messages will be sent. If not set then messages will be sent to the game object where the gesture resides as a default.

alternateCamera	All gesture default to using the Camera marked Camera.main. If there is no Camera.main then a Camera will be used. alternateCamera overrides this selection and sets the camera to be used. It is important the right camera to be used for any gestures that interact with game object colliders.
topColliderOnly	A boolean controlling if FingerLocation collider checks will only look for the first collider from the camera or look through all colliders to find the gesture's collider.
startsOnObject	Required location of the finger relative to the object for the finger down. If it fails to activate, the tap will fail. See FingerLocation (eg : Always,Over, etc)
movesOnObject	Required location of the finger relative to the object for all the moves.If any move fails the tap will fail. See FingerLocation.
endsOnObject	Required location of the finger relative to the object for the finger up. If it fails to activate, the tap fails. See FingerLocation.
endsOnObject	Boolean to make sure the tap moves very little, a true tap. This will subvert movesOnObject, and startsOnObject and endsOnObject should have values that co-exist, not Over with Not Over.

Messages

All messages return a TapGesture object.

GestureTap	The target number of taps has happened on this object.
-------------------	--------------------------------------------------------

Return Properties

tapsPerMinute	The taps per minute rate. Float, rational number.
fingerCount	Number of fingers used, Integer.
finger	A Finger object describing the finger that was used. See Finger Used.
fingers	Array of finger objects giving multiple fingers when finger count is greater than 1.

Touch Gesture

The Touch Gesture follows a single finger around the screen sending messages for down, move and up.

The up, down and move messages can each be controlled with an associated FingerLocation value. Somewhat strange behaviour can happen and should be taken into account when setting up this gesture. All down, up and moves will have a message IF their related FingerLocation succeeds. For example if down, up and move FingerLocation are all Over, then if the finger went down off the object - down not sent, the finger moved across the object - move messages when over object, and then went up on the object - up sent, then you would get no down, some of the moves and the up.

Class: TouchGesture

Input Properties

startsOnObject	Required location of the finger relative to the object for the finger down. If it fails to activate, the down message will not be sent. See FingerLocation.
movesOnObject	Required location of the finger relative to the object for all the moves. Each move that fails to activate will not be sent but moves that do activate will be sent. This means you can get the moves that happen on or off your object. See FingerLocation.
endsOnObject	Required location of the finger relative to the object for the finger up. If it fails to activate, the up message will not be sent. See FingerLocation.
targetCollider	The collider that the gesture will interact with. If not set the collider on the game object where the gesture resides will be used as a default.
targetMessageObjects	Another GameObject that will be sent messages.
alternateCamera	All gesture default to using the Camera marked Camera.main. If there is no Camera.main then a Camera will be used. alternateCamera overrides this selection and sets the camera to be used. It is important the right camera to be used for any gestures that interact with game object colliders.



topColliderOnly

A boolean controlling if FingerLocation collider checks will only look for the first collider from the camera or look through all colliders to find the gesture's collider.

Messages

All messages return a TouchGesture object.

GestureStartTouch	Sent when a finger goes down
GestureMoveTouch	Sent for each move by finger.
GestureEndTouch	Sent when finger is lifted.

Return Properties

fingerCount	Number of fingers used, Integer.
finger	A Finger object describing the finger that was used. See Finger Used. Use position for finger position.
fingers	Array of finger objects giving multiple fingers when finger count is greater than 1.

Gesture Engine

The Gesture Engine is for declaring the gesture internal engine in a place that will exist and will always be enabled. It does not perform an gesture functionality beyond declaration.

Normally, the gesture engine is declared on the first gesture that is started in Unity when a scene is started. If this gesture gets removed or disabled then all the other gesture stop working because the engine is lost. The root gesture can be used to fix this. As well, this can be used on static scene so the gesture engine is declared only once.

PLEASE BUY OUR SOFTWARE so we can continue to make great tools for the Unity 3D community

BC Gesture Library ©2012 BlackCherry Digital Media Inc.