Prof. Monster‘s

BGE Guide to

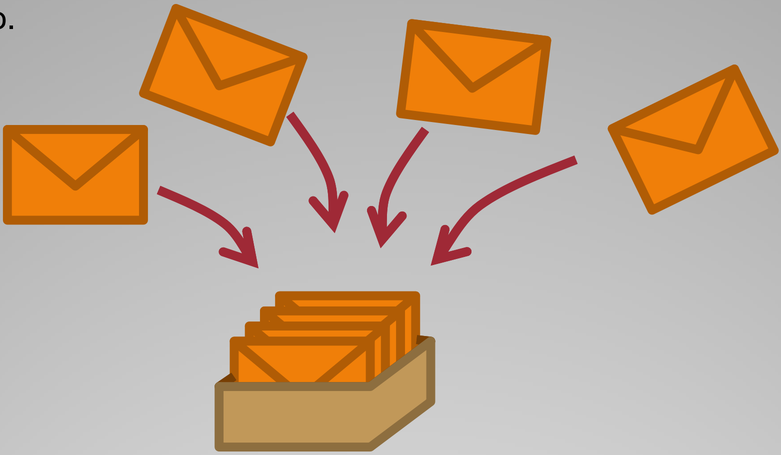
Messages

by Monster
Version 1.0
Date January 12, 2012

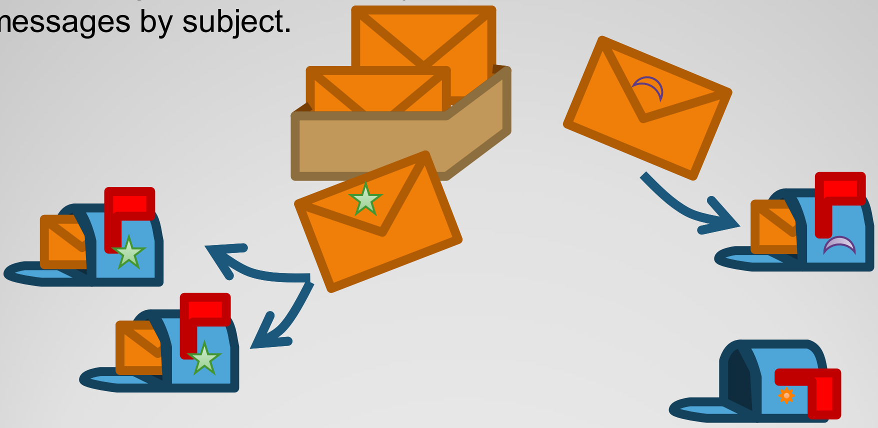
Messages can be an important part of your game. They are quite easy to use but they have their issues as well.

Introduction

The BGE message system is like a news paper service. It is a central instance that collects all messages regardless where they go to.



The BGE message system delivers copies of all messages to all readers regardless where they come from. Readers can filter the messages by subject.



Overview

Benefits

- **Easy** to setup via GUI or Python

- **Flexible** to use
  - no need to worry if there is a recipient and how many
  - no need to worry who sends a message from where
  - no need to worry inter-scene communication
  - filtered by subject

- It is **save** - messages do not get lost
  - even after scene change or
  - when multiple messages are received

- **Up-to-date** - messages lives for one frame only
  - no queuing of outdated messages

- Transfer ***strings only***

- *Multiple messages* can be received at once
  - might require Python code

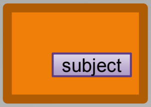
- The GUI allows to send the value of a property - but not to save it in another *property*
  - it requires Python code

- It is difficult to track what messages are in the queues
  - especially when the are targeted to a specific object (avoid that)

- It is not possible to *save messages* in a save/load system

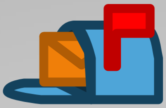
- Relatively *slow* compared to property copy

Drawbacks



A message carries a **subject string**. It acts as filter to differentiate several message types.

Different message types can be used for different purposes.



Just the fact that a message arrived is a **Boolean** information regardless of it's content:

- True = message received
- False = no message

Use cases of True/False values:

- can be used to trigger specific tasks (e.g. a controller)
- can be used for simple counters
- can be used to synchronize multiple independent objects

A message carries a **string body**. You need Python to process it.

Message Content

- The **recipient is not known** - or we do not care
  - example: send the health of the player
    - there can be one or more listening GUI elements
    - there can be a listening game flow elements (e.g. re-spawn)

- The **sender is not know** - or we do not care
  - example: collecting coins
    - each coin is sending a message when collected
    - a counter object is counting the received messages

- The sender and the recipient are in **different scenes**.

- To carry temporary information over to the **next frame**
  - (e.g. on scene switching)
  - example: you can send a message "game continues" - scenes from a newly started game wouldn't get such a message

- It is important to evaluate **each** single information
  - when using messages all messages are available and can be used to produce the desired results
  - Opposite: When copying properties the previous value of the property is overwritten (no history)

When to use

- If it is a **1:1** communication with known participants
  - (better use property copy)
- If **Complex data** should be transferred like arrays, dictionaries or object references.

When to avoid

- Write down what messages will be used and what purpose they have (you can use a text block for this documentation). If you use Python code to send messages such documentation is a MUST DO. Otherwise no one knows where messages are coming from and why they are there.

- Always use subject filters!

- Always use meaningful subjects content that shows the reader the purpose of the message.

- Message sensors should always have True Pulse on

- Use the **MessageAnalyzer** for debugging and tracking messages.

Recommendations

As an example lets design an health bar better. We better call it **Value Bar** as it can be used for other values too.

A Value Bar consists of two parts:

- **Value provider** which provides a value
- **Value display**(s) which shows a representation of the provided value (e.g. textual, graphical, sound)

These parts do not need to know each other. They just need to know the interface which is a message subject and the message body.

To allow multiple different value types you use different subjects.

The following implementation assumes the value to be displayed is in the body of the message.

The value provider places the value in the message.

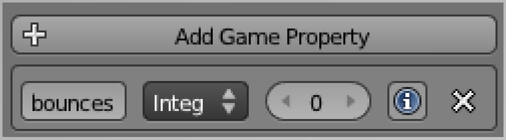
The value display takes it from the message. Our value display assume one value provider only. If multiple messages arrive at the same time (e.g. by different value providers) we pick the body of the first message.

Example: Health bar

As described the Value Provider sends a **value**.

Before it can do that we need a value. A simple option is a property. It is easy to configure and easy to manipulate.

In your example we count the bounces of a tennis ball.

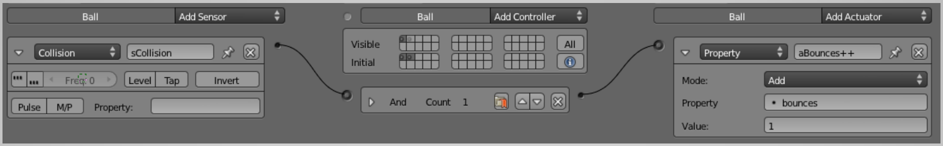


Add Game Property

bounces | Integ | 0

The counting logic is simple:

When colliding we increase the property



Ball | Add Sensor

Collision | sCollision

Freq: 0 | Level | Tap | Invert

Pulse | M/P | Property:

Ball | Add Controller

Visible | Initial | All

And | Count 1

Ball | Add Actuator

Property | aBounces++

Mode: Add

Property: bounces

Value: 1

Value

Value Provider

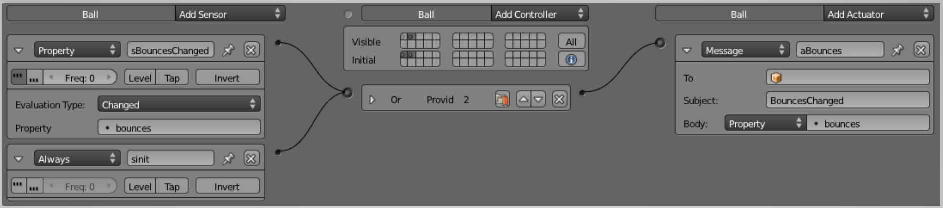
We place the logic of the **Value Provider** at another state.

That makes it easier to keep the overview of the complete logic. *Do not forget to enable all involved states!*

The Value Provider sends a new message at object creation and after the value changed.

That is simply done with an Always Sensor (no Pulses) and a Property Sensor (True Pulse) that senses for changes of the value.

Combine the output of both sensors with an OR controller as the message should be send in both cases.



Ball | Add Sensor

Property | sBouncesChanged
Freq: 0 | Level | Tap | Invert
Evaluation Type: Changed
Property: bounces

Always | sinit
Freq: 0 | Level | Tap | Invert

Ball | Add Controller

Visible | All
Initial

Or | Provid 2

Ball | Add Actuator

Message | aBounces
To:
Subject: BouncesChanged
Body: Property | bounces

The Message Actuator should get a meaningful subject. We choose “*bouncesChanged”* which describes its usage pretty well.

The Message Actuator provides the option to get feet by a property. So we can configure our value property.

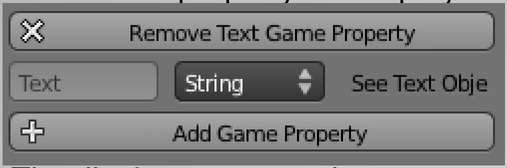
As you see you can apply this logic to nearly any object you like. Please remember to uses different subjects for different purposes.

Now values are delivered via the BGE message system. We need some **Value Displays** to present the values to the user. There are several options. We will implement two of them. You will see they are pretty simple.

We start with a **Text Display**. A Text Display presents the value as written text.

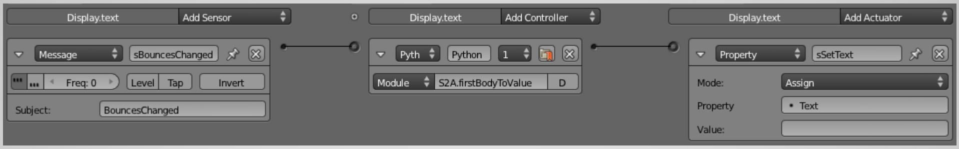
The BGE provides several options for text we use a Text object.

Since 2.62 the text object has a special property *Text.* When you write to this property the displayed text changes to the new value.



Remove Text Game Property
Text | String | See Text Obje
Add Game Property

The display reacts on the new message with a specific subject. Please note the subject should match the subject of a value provider. A message sensor (True Pulse) does this for us.



Display.text | Add Sensor
Message | sBouncesChanged
Freq: 0 | Level | Tap | Invert
Subject: BouncesChanged

Display.text | Add Controller
Pyth | Python | 1
Module | S2A.firstBodyToValue | D

Display.text | Add Actuator
Property | sSetText
Mode: Assign
Property: Text
Value:

We need a bit of Python to read the body of the message. Do not worry the S2A module already does that. The module

`S2A.firstBodyToValue`

places the body value of the first message at the value field of the connected Property Actuator and activates it.

That makes the received value displayed as Text.

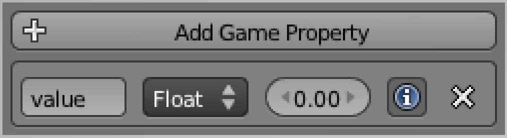
Text display

Text Displays are important but can be a bit boring.

So we add a **Graphical Display** too.

There are multiple options to do that too. The basics are the same.

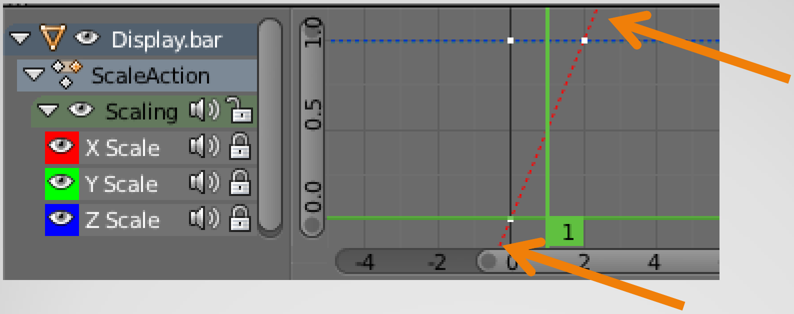
We place the received value in a property. The Display transforms that into the according representation. Here we create a simple bar that increases with larger values. First we need a property:



Now we need logic that transforms the value of the property into a graphical representation.

We choose a simple implementation. We add a plane with a scale action along the X axis. One key frame is at frame 0 and has a scale of 0. Another key frame is at frame one and has a scale of 1 (or whatever with you want for a value of 1).

Make sure the X-Scale curve gets a linear interpolation and a linear extrapolation (Curve Editor/F-Curve Editor).

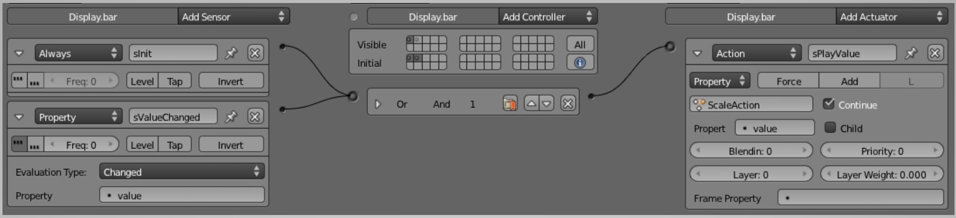


Graphical display

To scale the bar we simply play the according pose with an Action Actuator.

You activate the action actuator at object creation.

There is a bug in BGE 2.62 is a bug at the action actuator which forces you to activate the action actuator with each change of the property.



Display.bar | Add Sensor

Always | sInit
Freq: 0 | Level | Tap | Invert

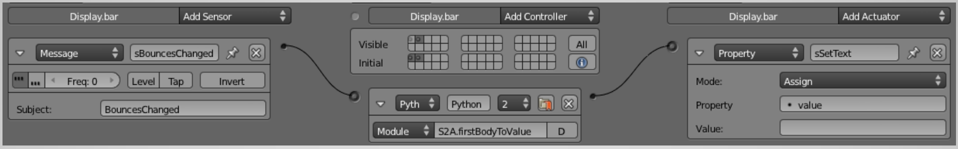
Property | sValueChanged
Freq: 0 | Level | Tap | Invert
Evaluation Type: Changed
Property: value

Display.bar | Add Controller
Visible
Initial
All
Or | And | 1

Display.bar | Add Actuator

Action | sPlayValue
Property | Force | Add | L
ScaleAction | Continue
Propert: value | Child
Blendin: 0 | Priority: 0
Layer: 0 | Layer Weight: 0.000
Frame Property:

Now we need the logic to change the value. We can simply replicate the logic we applied to the Text Display. This time we change property *value*.



Display.bar | Add Sensor

Message | sBouncesChanged
Freq: 0 | Level | Tap | Invert
Subject: BouncesChanged

Display.bar | Add Controller
Visible
Initial
All
Pyth | Python | 2
Module | S2A.firstBodyToValue | D

Display.bar | Add Actuator

Property | sSetText
Mode: Assign
Property: value
Value:

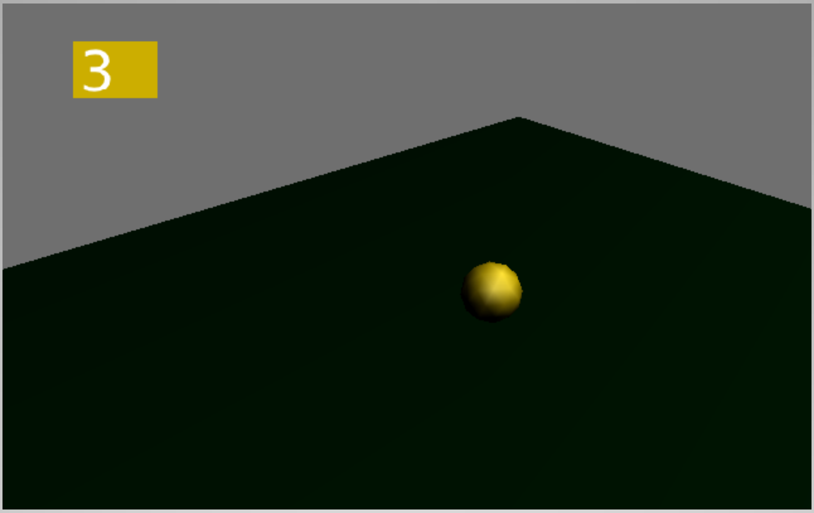
The module

`S2A.firstBodyToValue`

places the body value of the first message at the value field of the connected Property Actuator and activates it.

That changes the property value which updates the scale pose.

You see it is not that difficult to deal with messages. Additional you learned how to make a health bar as Text Display and as Graphical Display.



You can find the blend file at
http://blenderartists.org/forum/showthread.php?249078

Resume