

Triggers and Events

Video

<http://vimeo.com/2609584>

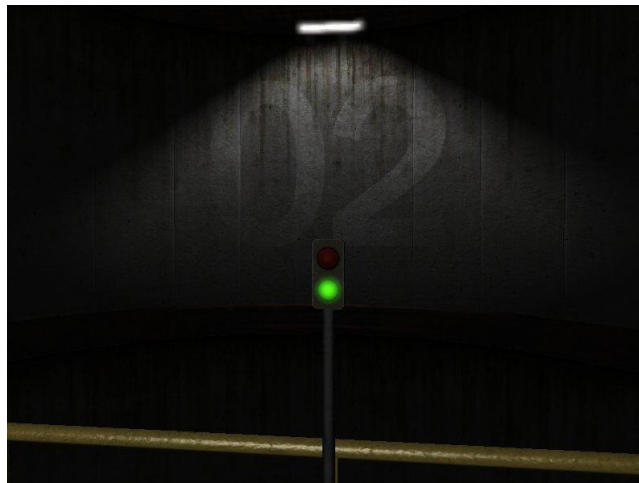
Required Files

http://developer.leadwerks.com/Tutorials/CPP/Triggers_And_Events_Files.zip

Recommended Reading

[Loading a Scene](#)

[Post-Processing Effects](#)



Introduction

Leadwerks Engine provides a set of commands and callbacks that can be used to handle a wide variety of game events and actions. These simple interactions form the basis of advanced gameplay mechanics.

Targets

Each entity can link to one or more other entities with targets. The `SetEntityTarget` command will set an entity's target value, and the `GetEntityTarget` command will retrieve it. When either entity is freed, the engine automatically removes the link, so that `GetEntityTarget` will never return an invalid entity. An optional index parameter can be used to set multiple targets for an entity. In our example, we are going to set up a switch that targets a light:



Messages

We can send a custom message to any entity. An entity message contains a string name for the message, an optional custom data pointer, and an optional delay. The name is used to tell the entity what action is being performed. The custom data pointer can store an entity, another string, or a custom data structure. The delay value can be used to send messages that should not be received for a set amount of time.

What happens when an entity receives a message? By default, nothing happens. However, if the user specifies an entity callback using the `ENTITYCALLBACK_MESSAGERECEIVE` constant, this callback function will be performed. The protocol for the message received callback function is as follows:

```
void MessageReceive ( TEntity entity, str name, byte* extra )
```

Note that the delay value is not passed to the callback function, because it is not needed. Typically, the message receive callback function will examine the name string and perform an action based on its value.

Putting it All Together

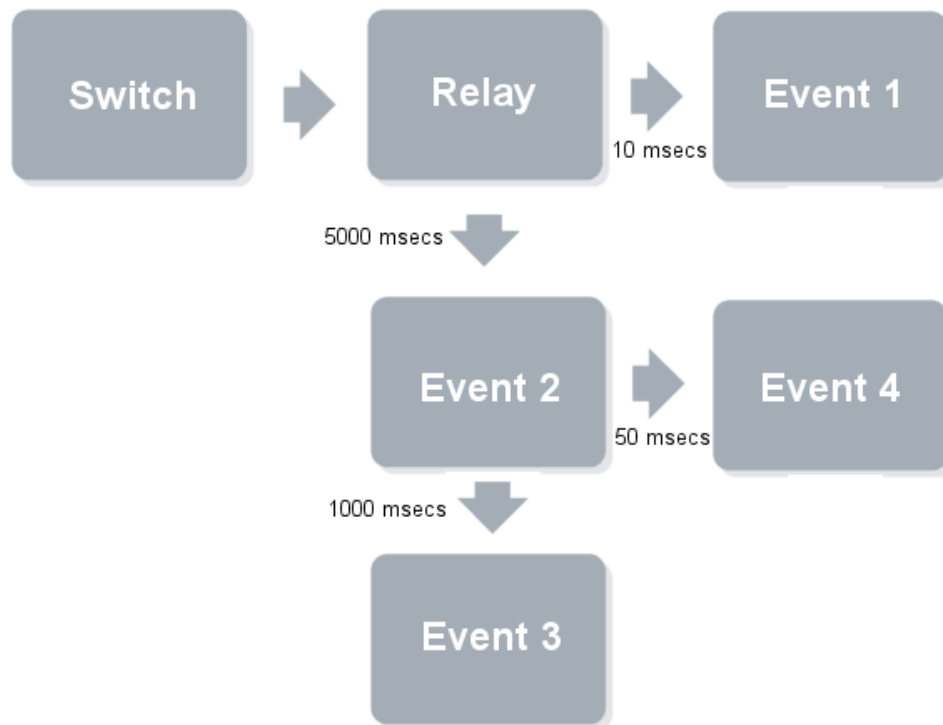
In the example for this lesson, the user activates a switch by looking at it and pressing the E key. A raycast is performed, and if the switch is hit, the switch model's target entity is retrieved. A message is then sent to the target entity, which in our example is a spotlight along the wall. The message name is either "hide" or "show", depending on the hidden state of the entity. The user can turn the light on and off by toggling the switch.

Note that the entire system is set up by reading a Sandbox scene instead of hard-coding the relationships. By designing abstract classes and actions, we can create a system that allows a new level

or even a new game to be designed, without modifying our game source code. Please see the video for this lesson for a more detailed explanation of the example.

Expanding the System

Because every entity can have multiple targets, and each message can have an optional delay, we can set up very complicated sequences of triggered events. The diagram below shows one theoretical system we could create:



The switch activates a relay entity, which is just a model or pivot with a message receive callback that relays a message to all of its targets, with different delay values. Event 1 might be a sound an entity plays when a message is received. Event 2 might be a door opening, and forwarding its received messages to its own targets. Event 3 could be a light changing color, while event 4 might be a giant mutant chicken spawning behind the player. Designing abstract classes and actions is the bulk of game programming. A well-designed system of potential interactions gives a map designer lots of options to play with, so be creative and have fun.