Event Editor

Inhalt

[Summary 1](#_Toc23438237)

[Target System 1](#_Toc23438238)

[Features 1](#_Toc23438239)

[Detailed Features 1](#_Toc23438240)

[Event Behavior 1](#_Toc23438241)

[Page 1](#_Toc23438242)

[Condition 1](#_Toc23438243)

[Trigger 2](#_Toc23438244)

[Setup 2](#_Toc23438245)

[Functions 2](#_Toc23438246)

[Mockup 2](#_Toc23438247)

[Scene View 2](#_Toc23438248)

[Inspector 2](#_Toc23438249)

[Classes 2](#_Toc23438250)

[Use Cases 2](#_Toc23438251)

[Architecture 2](#_Toc23438252)

[Milestones 2](#_Toc23438253)

[Task List 3](#_Toc23438254)

# Initialize

## Situation

Unity is a free engine to create games with. Unity’s code to object relation is a composition. A game object has mono behaviors in form of attached components. This enables many variations of interaction between other objects and the engine itself. This makes Unity easy to use for a broad genre of games.

This “openness” however has two sides to it. Unity does not have specific systems for a specific genre. Good thing here that we have Asset Stores, which cover this problem for us for most of the cases. But there is still a lot of potential left.

One such needed potential is a tool designed for adventure like RPG that simplifies the handling with events and objects.

## RPG Makers Event Editor

There is an engine out there called the RPG Maker. The RPG Maker series are all known for their simplicity because their Development Kit requires no knowledge of coding. They use

## Solution

This Tool enables game objects to have an event editor in their inspector. This event editor makes game objects easier to handle, because all objects are handled in a uniform matter.

## Target System

This tool is designed for Top-Down RGP like games.

## Features

The event editor handles its game object based on the settings done in the event inspector. Features are:

* Pages: Each event can have one or many pages.
* Conditions: Determines whether a page is active or not.
* Trigger: Determines how the event can be interacted with.
* Processing: Determines how the event is processed when it is active.
* Setup: Each Page has a setup section which sets up the game object.
* Functions: This is a list of function which are called when the event is triggered.

# Detailed Features

## Event Behavior

This is essentially the Event Editor found on the game object itself. It handles the game object depending on its values set up in a custom inspector. The Event behavior can have one or multiple pages.

## Page

An event can have one or multiple pages set up in the inspector. Each page has its own set of values, which can change the behavior of its game object. Only one Page can be active at a time. This depends which page first meets their condition. Pages can be added, copied or removed.

## Condition

Each Page has a condition section. The first page which all conditions are met is active for the event. A page can have none to many conditions. Following conditions can be set up in the inspector:

* Global Switch: This is a global dictionary of type <string, bool> value. This dictionary can be accessed anywhere and anytime. It is stored on the hard disk.
* Local Switch: This switch exists only on the event itself and on all its pages. It cannot survive scene changes.
* Game object variables: All specified variables exposed with the tag “[ConditionalField]” are listed from the specified game object found in the same scene. Depending on the type of the variable the condition can be set as following:
  + Integer: value is **equal**, **greater than**, **greater or equal than**, **less than**, **less or equal than** or **not** target value. Target value can be given by the user or be compared to another variable inside a game object.
  + Bool: value is either **true** or **false**.

## Trigger

This determines how the event can be interacted with. There are:

* Parallel: Automatically processes function calls in the function section in a coroutine.
* Autorun: Automatically processes function calls in the function section in a coroutine and loops when the list is finished.
* Interaction: Triggers only on interaction calls from an Interactor.
* Collision: Triggers on collision controls. **OnTriggerEtc.** or **OnCollsionEtc.**

## Setup

These changes setup the event from the moment where when a page becomes active. Changes are:

* Sprite: What sprite is shown.
* Animator: Which animator to use.
* Character State: **IsDirectionFix**, **DefaultState**
* Sorting Layer: Sets Sorting Layer for the object.
* Animation Speed: How fast the animator plays animations.

## Functions

This list of function is called when the event is being triggered. It’s essentially a UnityEvent list where all objects inside the scene can subscribe their functions to.

In later iterations this list might get revamped to or completely replaced by an own Custom Event List, which has new inspector elements for better functionality. These elements can then be attached to one and other. The following Mini Tools will be a part of this new list.

* Flow Control: Conditional Branches, Loops, etc.
* In-depth Object Control: Custom tools for managing object variables like health, buffs, inventory, Camera Control, Shaders, Map changes, Tile Set handling etc.
* Game Control: Custom tools for managing game flow like battle processing, Scene management, etc.
* Message integration: A custom message tool to create dialogs.

# Mockup

## Scene View

## Inspector

# Classes

## Use Cases

## Architecture

# Milestones

|  |  |  |
| --- | --- | --- |
| Id | Date | Name |
|  |  |  |
|  |  |  |

# Task List

Task broken down from features. A Category represents the feature the task is used for.

|  |  |  |
| --- | --- | --- |
| Id | Category | Description |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |