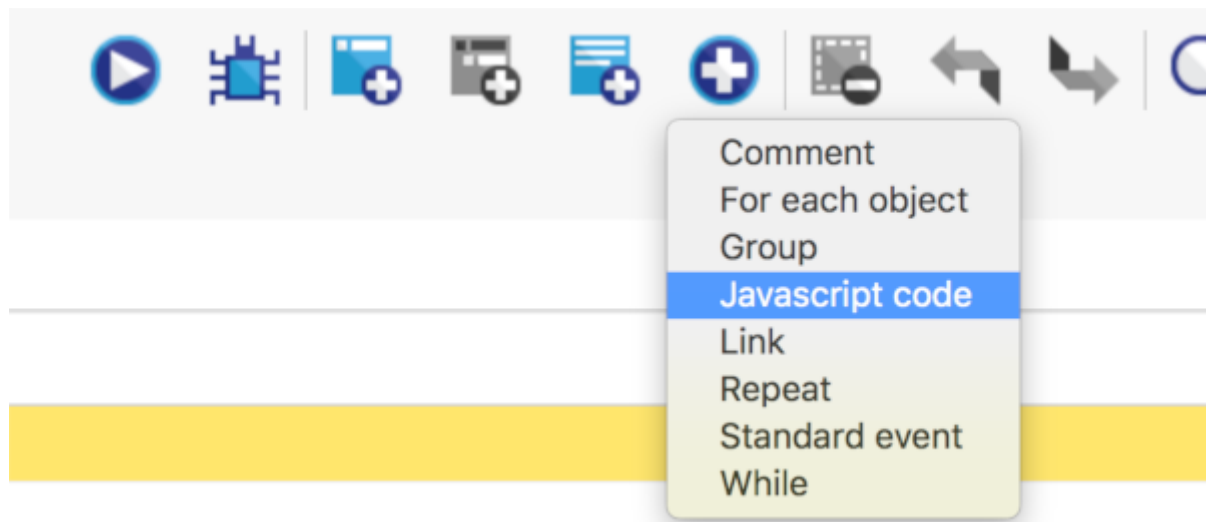


JavaScript Code events

Special Event Description

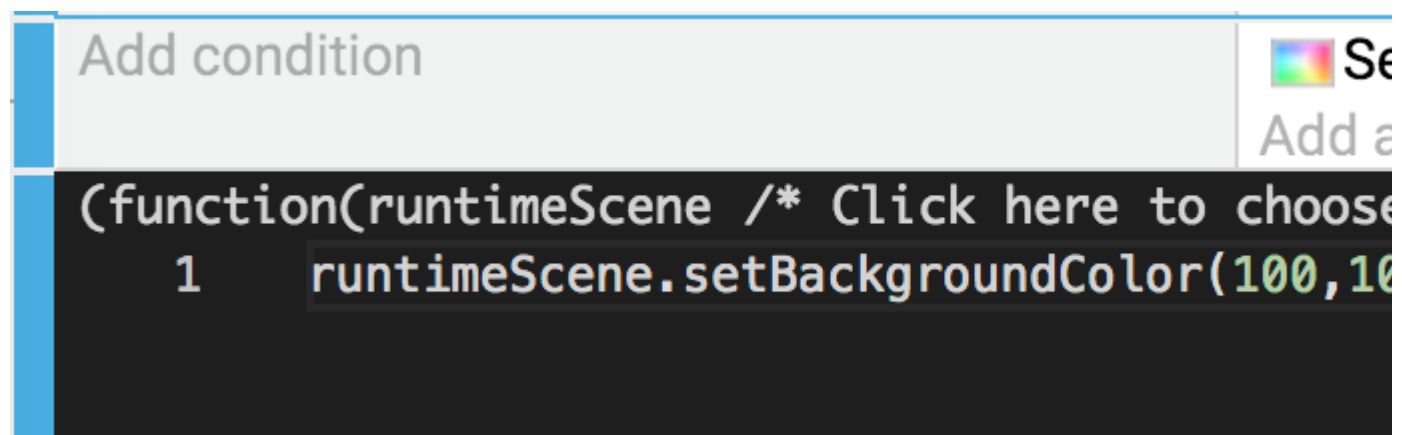
This event allows you to add custom **JavaScript code** in your game. It's only meant to be used by advanced users who are familiar with JavaScript programming.

To add a JavaScript event, click on the  button in the Events Sheet toolbar, and select JavaScript code:

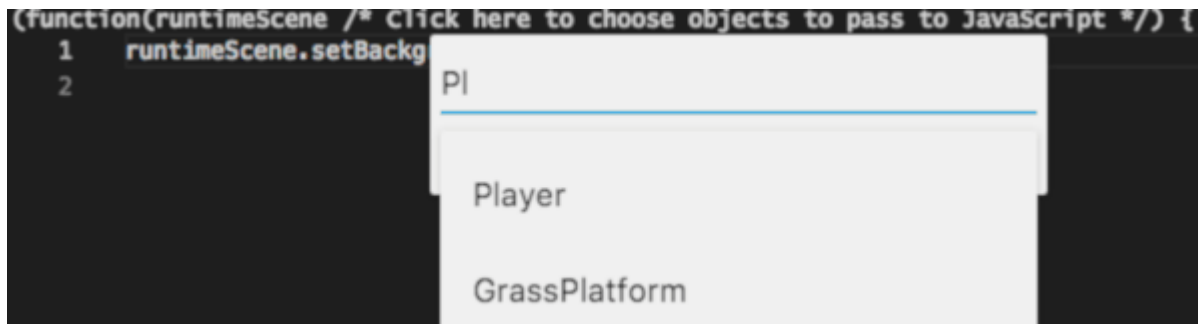


When added, click on the code to edit it. The code will be executed each time the event is reached by GDevelop.

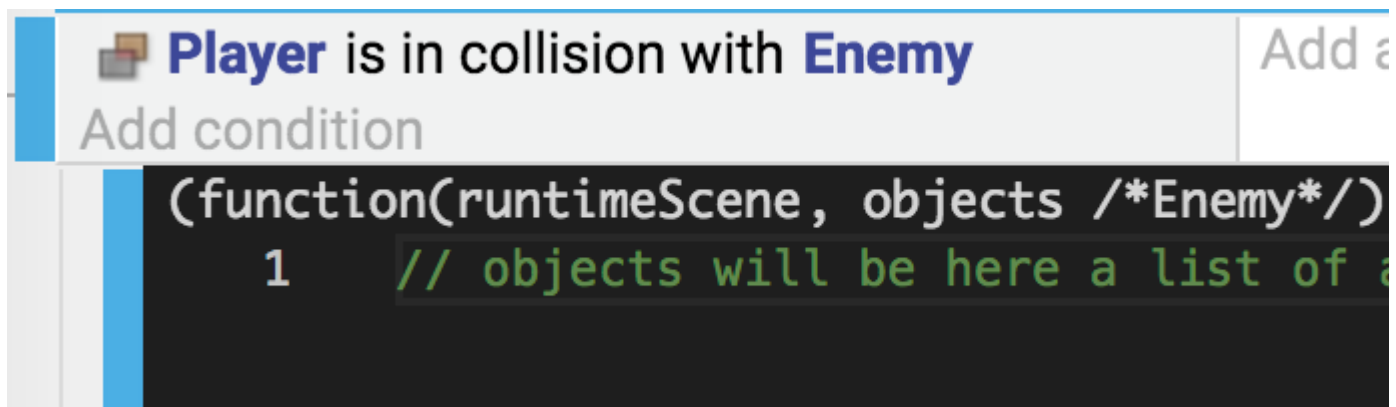
Note that in the code, you are in strict mode, but you have access to a variable named *runtimeScene*. The variable, "runtimeScene", represents the scene being played. Below is an example of a standard GDevelop event, and an equivalent event that uses a JavaScript:



If you click on "Click here to choose objects to pass to Javascript" next to the *runtimeScene* parameter, an array variable called "objects" will be available to use. A menu of your game objects will pop up. You can select the game object of our choice. That object will be added to your javascript object array.



“objects” is an array that contains the selected instances of the object that you choose. It is exactly like what an action or a condition would use). For example, the illustration below is an event using a JavaScript event as a sub-event. The javascript event will manipulate the enemies that are colliding with the player:

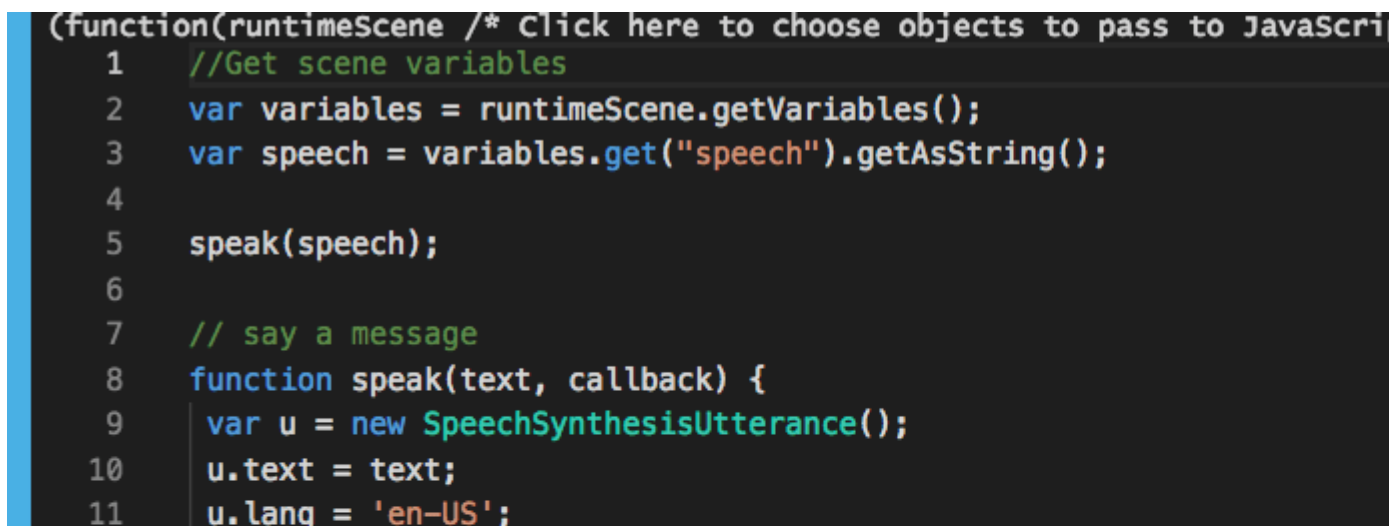


Adding special events to the project is completed the same way as [adding a standard event, as described here](#).

Examples

See it in action! 🎮

Open these examples online. You can do this with the links given below. There is a number of more examples available when you ask GDevelop to create a new project, then click the examples button.



```

12
13   u.onend = function () {
14     if (callback) {
15       callback();
16     }
17   };
18
19   u.onerror = function (e) {
20     if (callback) {
21       callback(e);
22     }
23   }
24 }
25 })(runtimeScene /* Click here to choose objects to pass to JavaScript */);

```

- [Text to Speech Example](#)

Fading out:


```

1 (function(runtimeScene, objects /*Enemy*/) {
2   objects.forEach(object => {
3     /** @type {gdjs.SpriteRuntimeObject} */
4     const enemy = object;
5
6     /** @type {gdjs.PlatformerObjectRuntimeBehavior} */
7     const platformerBehavior = object.getBehavior("PlatformerObject");
8
9     if (enemy.getAnimation() === 1 && platformerBehavior.isOnFloor())
10      object.activateBehavior("PlatformerObject", false);
11     enemy.setOpacity(enemy.getOpacity() - 50 * object.getElapsedT
12
13     if (enemy.getOpacity() === 0) {
14       object.deleteFromScene(runtimeScene);
15     }
16   }
17 })

```

```
})(runtimeScene, objects /*Enemy*/); // Read the documentation and help
```

 The number of the current animation of **Enemy** is = 1

 **Enemy** is on floor

 **Enemy** is moving

 Activate behavior

 Do - 50 * TimeD

Add action

- [Javascript Blocks in Platformer Example](#)

Documentation

- Read [the game engine documentation](#) to get started.
 - You can also browse directly the [GDJS game engine source code](#).

Code examples

Read and change the value of a variable

```
var myVar = runtimeScene.getVariables().get("MyVar");
var myVar2 = runtimeScene.getVariables().get("MyVar2");

var currentValue = myVar.getAsNumber();
myVar.setNumber(currentValue+1);

myVar2.setString("Hello, world");
```

See the documentation of [gdjs.Variable](#) and [gdjs.VariablesContainer](#).




Move object at the position of another, check if the game is rendering the first frame

```
const players = runtimeScene.getObjects("Player");
const playerHitBoxes = runtimeScene.getObjects("PlayerHitBox");
// Stop early if for some reason the player or the hitbox object is not found.
if (playerHitBoxes.length === 0 || players.length === 0) return;

// At the first frame only (i.e: at the beginning of the scene).
if (runtimeScene.getTimeManager().isFirstFrame()) {
  // Hide the first instance of PlayerHitBoxes.
  playerHitBoxes[0].hide();
}

// Set the position of the Player object.
players[0].setX(playerHitBoxes[0].getX() - 12);
players[0].setY(playerHitBoxes[0].getY());
```

Equivalent events would be:

 At the beginning of the scene	 Hide
Add condition	Add action
Add condition	 Do
	Add action

Change animation according to some conditions on the behavior of an object

In the code below, we use a comment which is called an annotation.

By writing the **annotation** `/** @type {gdjs.XXX} */` just before the declaration of a variable in JavaScript, you let the code editor know that the variable has the type

gdjs.XXX. The editor will be able to assist you by providing **autocompletion** while you type (or when you hover over a word).

Most of the time annotations are not needed. The methods that you are using are already <https://docs.gdevelop-app.com/GDJS%20Runtime%20Documentation/> with type annotations. But when you use a list of objects, you may want to access this object specific method (for example, methods to modify the animation of a sprite object). In this case, you could write `/** @type {gdjs.SpriteRuntimeObject} */`.

If you don't do this, the code will still work, but the editor will only be able to provide you with the methods of `gdjs.RuntimeObject` (the base class).

When you get the behavior from an object, you also know what kind of behavior you're getting. Let the editor know this using an annotation (otherwise, you'll only get autocompletion for the base class, `gdjs.RuntimeBehavior`).

```
const players = runtimeScene.getObjects("Player");
const playerHitBoxes = runtimeScene.getObjects("PlayerHitBox");
if (playerHitBoxes.length === 0 || players.length === 0) return;

















/** @type {gdjs.SpriteRuntimeObject} */
const player = players[0];

/** @type {gdjs.PlatformerObjectRuntimeBehavior} */
const platformerBehavior = playerHitBoxes[0].getBehavior("PlatformerObject");

if (platformerBehavior.isJumping() || platformerBehavior.isFalling()) {
    player.setAnimation(1);
} else if (platformerBehavior.isOnFloor()) {
    if (!platformerBehavior.isMoving()) {
        player.setAnimation(0);
    } else {
        player.setAnimation(2);
    }
}

const LEFTKEY = 37;
const RIGHTKEY = 39;
if (runtimeScene.getGame().getInputManager().isKeyPressed(LEFTKEY)) {
    player.flipX(true);
} else if (runtimeScene.getGame().getInputManager().isKeyPressed(RIGHTKEY)) {
    player.flipX(false);
}
```

Equivalent events would be:

	PlayerHitBox is jumping		De
	Add condition		Add a
	PlayerHitBox is falling		De
	Add condition		Add a
	PlayerHitBox is on floor		Add a
	Add condition		Add a
	PlayerHitBox is moving		De
	Add condition		Add a

 PlayerHitBox is moving	 Do
Add condition	Add a
 Left key is pressed	 Fl
Add condition	Add a
 Right key is pressed	 Fl
Add condition	Add a


Set the position of the camera to the position of an object

```
if (!objects.length) return;
```

```
// Here, "objects" refer to a list of "Player", which should be selected
// in the configuration of the function.
```

```
runtimeScene.getLayer("").setCameraX(objects[0].getX());
```

This equivalent event would be:

Add condition	 Do
	Add a

Fade out a sprite object after it is on the floor (using the Platformer Object behavior)






```
objects.forEach(object => {
  /** @type {gdjs.SpriteRuntimeObject} */
  const enemy = object;

  /** @type {gdjs.PlatformerObjectRuntimeBehavior} */
  const platformerBehavior = object.getBehavior("PlatformerObject");

  if (enemy.getAnimation() === 1 && platformerBehavior.isOnFloor() && !platformerBehavior.isMoving)
    object.activateBehavior("PlatformerObject", false);
  enemy.setOpacity(enemy.getOpacity() - 50 * object.getElapsedTime(runtimeScene) / 1000);

  if (enemy.getOpacity() === 0) {
    object.deleteFromScene(runtimeScene);
  }
});
```

The equivalent events would be:

 The number of the current animation of Enemy is = 1	 Add
 Enemy is on floor	 Do
 Enemy is moving	Add a
Add condition	

The opacity of **Enemy** is = 0

X Do

Add condition

Add a

Use Javascript to get a value from parameters

In Javascript, you can get the value from your function easily. Here let see how to handle an object. Type is *Objects* as you can see *Objects* is plural, this means the values returned will be an array of instances of your object pass into parameters.

Function Configuration

CONFIGURATION

PARAMETERS

OBJECT GROUPS

Parameter #1: RotateObjects

Type
Objects

Object type
Any object

Label
The objects to be rotated

+ ADD A PARAMETER

? HELP

For get the array of instances we use this function:

```
eventsFunctionContext.getObjects("myParameter");
```

Then in your case with a parameter named *RotateObjects* if we want rotate the object by 45 degrees we do

```
const objects = eventsFunctionContext.getObjects("Rotate0bjects");  
const firstInstance = objects[0];  
firstInstance.setAngle(45);
```

Let see now how to make the angle a parameter, angle is number typed.

Parameter #2: degrees

Type
Number

Label
Angle in degrees

```
const angle = eventsFunctionContext.getArgument("degrees");
```

The same code can be used for String type and others.

If you see

```
GetArgumentAsString()  
GetArgumentAsNumber()
```

These are two expressions, these are not used in Javascript.