HOW EVENTS WORK

View online: https://www.construct.net/en/make-games/manuals/construct-3/project-primitives/events/how-events-work

If you're new to Construct's events, this section will outline how they work. This is essential reading for beginners! You will be able to make much better and more reliable games with a thorough understanding of how events work.

To learn how to add and edit events, see Event Sheet View.

Events are designed to be easily readable and to intuitively "just work". However, they have specific, well-defined ways of working which is described here.

Events work by **filtering** specific instances that meet some conditions. The actions then run for *those instances only*. For example, consider the following event:

In this example, when a *Bullet* collides with a *Monster* the event condition is met. The specific instances of *Bullet* and *Monster* that collided in the game are "picked" by the event. Actions only run on the "picked" instances. If there are other instances of *Bullet* and *Monster* in the layout, they won't be affected by the *Subtract 1 from health* and *Destroy* actions. It would be very difficult to make good games if every bullet hurt every monster!

Another way to think about an event is "**If** all conditions are met **then** run actions on the instances meeting the conditions".

Multiple conditions

Adding more conditions to an event progressively filters the instances to run actions on. For example:

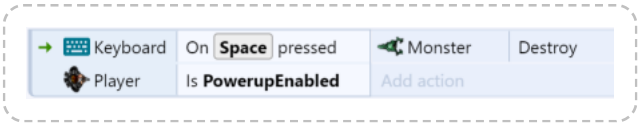
This event runs like this:

1. First all *Monsters* that are on-screen are picked.
2. Then, of those on-screen, it is reduced to those with less than 5 health.
3. The action makes all monsters that are **both** on-screen **and** have less than 5 health look directly at the player. Monsters that are off-screen or have 5 or more health are not affected.

Therefore, using multiple conditions you can run actions on just the instances meeting several criteria. Users from programming languages or other tools might recognise this as a logical "AND". All conditions of an event must be met for the actions to run. If no monsters are on-screen or none of those on-screen have less than 5 health, the actions do not run at all.

Unreferenced objects

Have a look at the following event:



If the user presses `Spacebar` and the Player's *PowerupEnabled* flag is set, the action does *Monster: Destroy*. Note that there aren't any conditions that filter or pick Monsters in this event. In this case, *all* Monster instances are destroyed. In other words, if an event doesn't reference an object in its conditions, actions apply to all the instances of that object.

Think of conditions as starting with all instances being picked, and progressively filtering them from there. If there were no conditions, there are still all instances picked, so the action affects all of them.

Picking resets between events

After an event ends, the next event begins from scratch. Its conditions will start picking from all instances again.

On the other hand, sub-events (which appear indented) **carry on from where its parent event left off**. A sub-event will further filter the instances left over by the event that came before it. If an event has two sub-events, they both pick from the same set of instances the parent left behind - the second sub-event is not affected by the first. In other words, events at the same indentation level always pick from the same set of instances, and events at a lower indentation level are always working with the instances handed down from above.

The System object

In Construct the System object represents built-in functionality. It has no instances. This means most system conditions do not pick any instances: they are either true or false. If they are false the event stops running, otherwise the event continues without the picked instances having been changed. There are exceptions, though: if a system condition uses an object, such as *Pick random instance*, that will affect the picked objects.

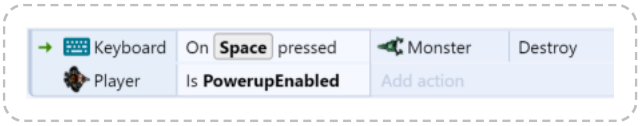
System actions do not run on any picked objects: they simply run if all of the event's conditions were met.

Events run top to bottom

The order of events is important. Every event is checked once per tick (about 60 times a second on most computers), and they are run from top to bottom in the event sheet. The screen is drawn once every event has been run, then the process starts again. This means if one event does something and the next event undoes it, you'll never see that anything happened.

The same applies within events: conditions are checked from top to bottom, and the actions run from top to bottom.

However, **triggers** are an exception. See the green arrow to the left of *Keyboard: On Space pressed* from the previous example:



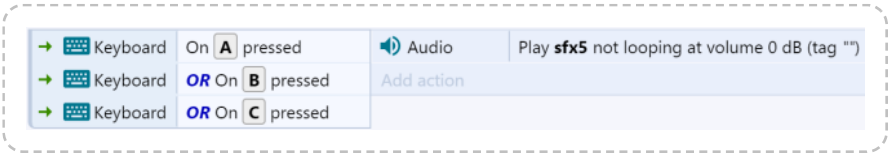
This indicates the event is **triggered**. Rather than running once per tick, this event simply runs (or "fires") upon something actually happening. In this case, the event runs when the user hits the Spacebar key on the keyboard. It is never checked any other time. Since triggers run upon an event happening, they aren't checked in top-to-bottom order like other events. This means the ordering of triggers relative to other events is not important (except relative to other triggers of the same type, since triggers still fire top-to-bottom).

There can only be one trigger in an event, because two triggers cannot fire simultaneously. However, multiple triggers can be placed in 'Or' blocks (see the next section).

'Or' blocks

As mentioned before, all conditions have to be met for an event to run. This is called a 'Logical AND', because "condition 1 AND condition 2 AND condition 3..." must be true. However, you can change an event to run when any condition is true. This is called a 'Logical OR', because the event will run if "condition 1 OR condition 2 OR condition 3..." are true.

Normally blocks work as 'AND' blocks. To make an 'OR' block, right-click the block and select Make OR block. It will then display with - or - between each condition, as shown below.



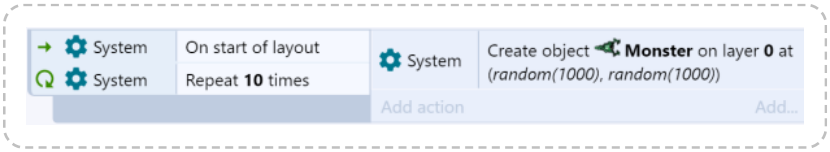
Note that because OR blocks run if any condition is true, it's possible the event will still run if some conditions were false and did not pick any instances. In this case the actions will still run, but possibly with zero instances picked for any objects where no instances met the condition. If any actions are run for objects with no instances picked, nothing happens.

Also note normally you can only put one trigger in an event, but you can put multiple triggers in an 'Or' block, and the event will run if any of triggers run.

You can combine the block types by using sub-events. This allows you to build up more advanced logic, such as an 'Or' block followed by an 'And' block.

Looping conditions

Some events **loop**, which simply means they repeat more than once. Note the green circular arrow in the below example to indicate this.



This means when the layout starts, the *Create object* action repeats 10 times. The end result is 10 monsters are created at random positions in the layout on startup.

There can also be more conditions following the *Repeat* condition. These are tested on each of the repeats as well, and must be true for the actions to run. There can even be more than one loop in an event, but this is rare.

Families and containers

Note families Paid plans only pick their instances entirely separately from any of the object types in the family. For more information, see the section *Picking families in events* in the manual entry on Families.

Containers are an advanced feature that can also make groups of instances always be picked together. For more information see the manual entry on Containers.

Summary

Using this event system it's possible to make sophisticated logic for games quickly and easily. It is a very powerful alternative to scripting or programming languages but much easier for non-technical people to use.

Although this section has described the essential parts of the event system, it still has not covered everything. The rest of this manual section covers more features you can use in events. The reference sections also cover all the conditions, actions and expressions in Construct.