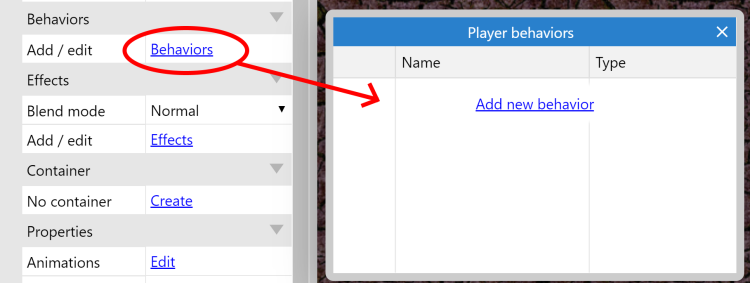
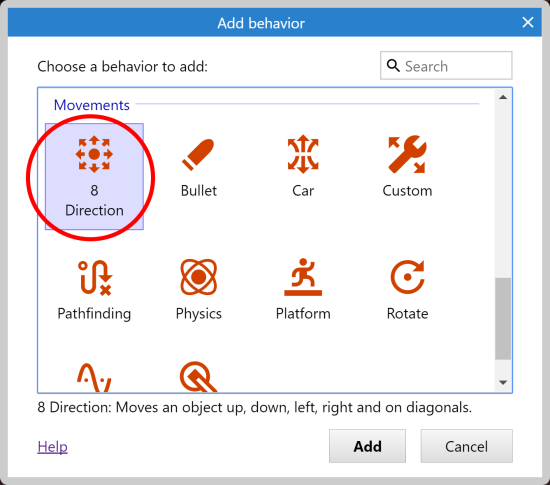
**How to add a behavior**

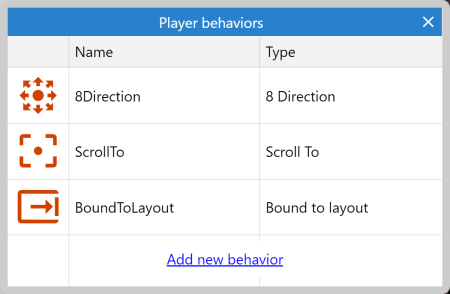
Let's add the **8 direction** movement behavior to the player. Click the player object to select it. In the Properties Bar, notice the Behaviors category. Click the **Behaviors** link there. The Behaviors dialog for the player will open.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1108/behaviors-dlg.png)The Behaviors dialog

Click **Add new behavior** in the behaviors dialog. Double-click the **8 direction** movement to add it.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1109/add-8dir.png)

Do the same again and this time add the **Scroll To** behavior, to make the screen follow the player. Then also add the **Bound to layout** behavior, to keep them inside the layout. The behaviors dialog should now look like this:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1110/player-behaviors.png)

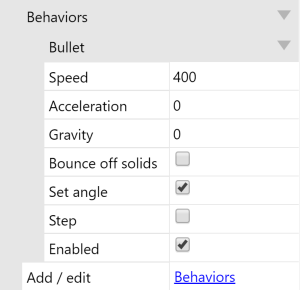
Close the behaviors dialog. Now try pressing **Preview** to run the game so far! Once the preview starts, notice you can already move around with the arrow keys, and the screen follows the player. You also can't walk outside the layout area, thanks to the Bound to Layout behavior. This is what behaviors are good for - quickly adding common features. We'll be using the event system soon to add custom features.

**Adding the other behaviors**

We can add behaviors to the other objects by the same method - select it, click the *Behaviors* link to open the behaviors dialog, and add some behaviors. Let's add these other behaviors:

1. Add the **Bullet movement** and **Destroy outside layout** to the **Bullet** object (no surprises there)
2. Add the **Bullet movement** to the **Monster** object (because it just moves forwards as well, just at a slower speed)
3. Add the **Fade** behavior to the **Explosion** object (so it gradually disappears after appearing). By default the Fade behavior also destroys the object after it has faded out, which also saves us having to worry about invisible Explosion objects clogging up the game.

If you run the game, you might notice the only thing different is any monsters you can see suddenly shoot off rather quickly. Let's slow them down to a leisurely pace. Select the **Monster** object. Notice how since we added a behavior, some extra properties have appeared in the properties bar:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1111/bullet-properties.png)

This allows us to tweak how behaviors work. Change the speed from 400 to **80** (this is in pixels travelled per second).

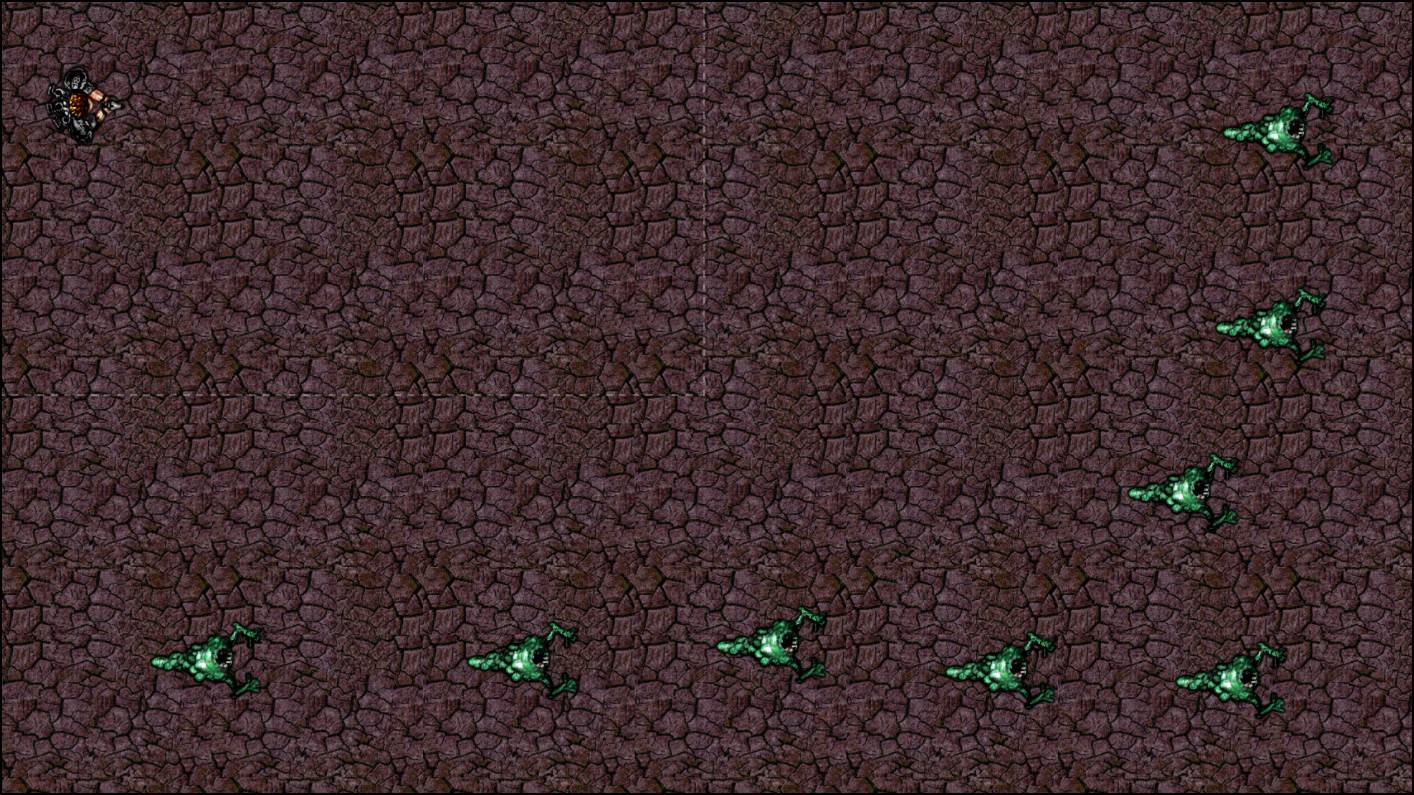
Similarly, change the **Bullet** object's speed to **600**, and the **Explosion** object's Fade behavior's **Fade out time** to **0.5** (that's half a second).

# Create some more monsters

Holding Control, click and drag the **Monster** object. You'll notice it creates another *instance*. This is simply another object of the Monster *object type*.

Object types are essentially 'classes' of objects. In the event system, you mainly deal with object types. For example, you might make an event that says "Bullet collides with Monster". This actually means "*Any instance of the* Bullet *object type* collides with *any instance of the* Monster *object type*" - as opposed to having to make a separate event for each and every monster. We'll cover more on object types vs. instances later. For now, a good example to think about is different types of enemy are different object types, then the actual enemies themselves (which there might be several of) are instances of those object types.

Using Control + drag, **create 7 or 8 new monsters.** Don't place any too close to the player, or they might die straight away! Remember you can zoom out with Control + mouse wheel down if it helps, and spread them over the whole layout. You should end up with something a bit like this.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1112/monsters-layout.jpg)

Now it's time to add our custom logic with Construct's visual method of programming - events!

# Events

First, click the **Event sheet 1** tab at the top to switch to the *Event Sheet View*. A list of events is called an *Event sheet*, and you can have different event sheets for different parts of your game, or for organisation. Event sheets can also "include" other event sheets, allowing you to reuse events on multiple levels for example, but we won't need that right now.

[Event sheet tab](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1113/event-sheet-tab.png)

**About events**

As the text in the empty sheet indicates, Construct runs everything in the event sheet once per tick. Most screens update their display 60 times per second, so Construct will try to match that for the smoothest display. This means the event sheet is usually run 60 times per second, each time followed by redrawing the screen. That's what a tick is - one unit of "run the events then draw the screen".

Events run top-to-bottom, so events at the top of the event sheet are run first.

**Conditions, actions and sub-events**

Events consist of **conditions**, which test if certain criteria are met, e.g. "Is spacebar down?". If all these conditions are met, the event's **actions** are all run, e.g. "Create a bullet object". After the actions have run, any **sub-events** are also run - these can then test more conditions, then run more actions, then more sub-events, and so on. Using this system, we can build sophisticated logic for our games and apps. We won't need sub-events in this tutorial, though.

Let's go over that again. In short, an event basically runs like this:

Are all conditions met?

* **Yes:** run all the event's actions.
* **No:** go to next event (not including any sub-events).

That's a bit of an oversimplification. Construct provides a lot of event features to cover lots of different things you might need to do. However, for now, that's a good way to think about it.

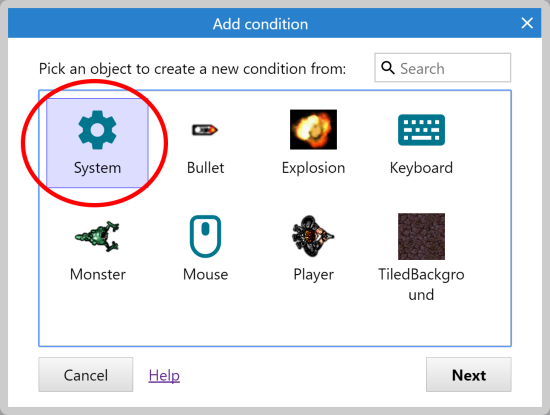
**Your first event**

We want to make the player always look at the mouse cursor. It will look like this when we're done:

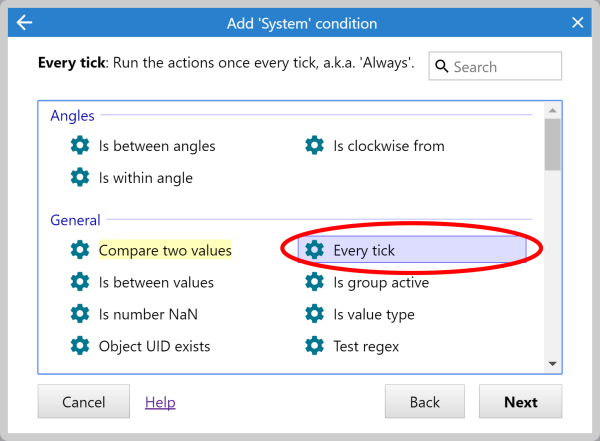
[The first event](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1114/gstut-event1.png)

Remember a tick runs every time the screen is drawn, so if we make the player point at the mouse cursor position every tick, they'll always appear to be facing the mouse cursor.

Let's start making this event. Double-click a space in the event sheet. This will prompt us to add a **condition** for the new event.

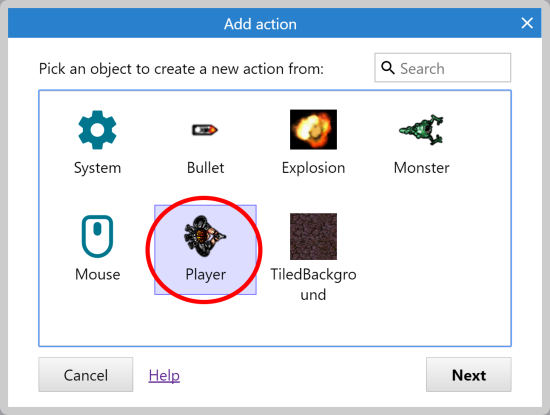
[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1115/gstut-sysobj.png)

Different objects have different conditions and actions depending on what they can do. There's also the **System object**, which represents Construct's built-in features. **Double-click** the System object as shown. The dialog will then list all of the System object's conditions:

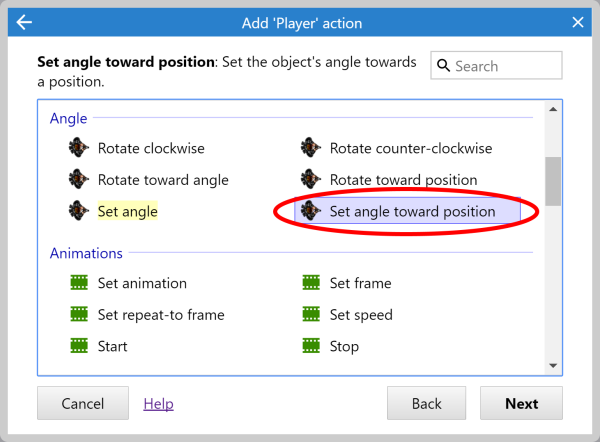
[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1116/gstut-everytick.png)

**Double-click** the *Every tick* condition to create an event with it. The dialog will close and the event is created, with no actions.

Now we want to add an action to make the player look at the mouse cursor. Click the **Add action** link to the right of the event. (Make sure you get the *Add action* link, **not** the *Add event* link underneath it which will add a whole different event again.) The Add Action dialog will appear:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1117/gstut-playerobj.png)

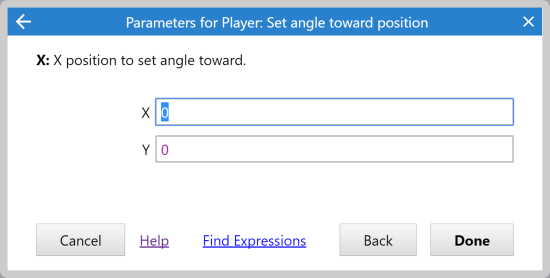
As with adding an event, we have our same list of objects to choose from, but this time for adding an action. Try not to get confused between adding conditions and adding actions! As shown, **double-click** the *Player* object, for it is the player we want to look at the mouse cursor. The list of actions available in the Player object appears:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1118/gstut-playeract.png)

Notice how the player's 8-direction movement behavior has its own actions. We don't need to worry about that for now, though.

Rather than set the player's angle to a number of degrees, it's convenient to use the **Set angle towards position** action. This will automatically calculate the angle from the player to the given X and Y co-ordinate, then set the object's angle to that. **Double-click** the *Set angle towards position* action in the list.

Construct now needs to know the X and Y co-ordinate to point the player at:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1119/gstut-paramsdlg.png)

The X and Y fields are called the **parameters** of the action. Conditions can have parameters too, but *Every tick* doesn't need any.

We want to set the angle towards the mouse position. The Mouse object can provide this. Enter Mouse.X for **X**, and Mouse.Y for **Y**. These are called *expressions*. They're like sums that are calculated. For example, you could also enter Mouse.X + 100 or sin(Mouse.Y) (although those particular examples might not be very useful!). This way you can use any data from any object, or any calculation, to work out parameters in actions and conditions. It's very powerful, and a sort of hidden source of much of Construct's flexibility.

If you get an error that says "Mouse is not an object name", make sure you added the Mouse object! Go back to page 2 and check under *Add the input objects*.

You might be wondering how you'd remember all the possible expressions you could enter. Firstly, you might notice Construct shows some lists as you type. This is called *autocomplete* and helps show what you can type at each point in an expression. Secondly, there's also the *Expressions Dictionary* which lists them all. If there's room on your screen, it will appear faded out, so it doesn't distract you until you need it. If it's not there, you can click *Find expressions* to open it. You can double-click an object in the Expressions Dictionary to see a list of all its expressions. If you double-click an expression, it will also insert it for you, saving you from having to type it out.

Anyway, click **Done** on the parameters dialog. The action is added! As you saw before, it should look like this:

[The first event](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1114/gstut-event1.png)

There's your first event! Try running the game, and the player should now be able to move around as before, but always facing the mouse cursor. This is our first bit of custom logic.

# More game logic

If each event is described in as much detail as before, it's going to be quite a long tutorial. Let's make the description a little briefer for the next events. Remember, the steps to add a condition or action are:

1. Double-click to insert a new event, or click an *Add action* link to add an action.
2. Double-click the object the condition/action is in.
3. Double-click the condition/action you want.
4. Enter parameters, if any are needed.

From now on, events will be described as the object, followed by the condition/action, followed by any parameters. For example, the event we have just inserted could be written:

Add condition System►Every tick

Add action Player►Set angle towards position, and for X: Mouse.X, Y: Mouse.Y

**Get the player to shoot**

When the player clicks, they should shoot a bullet. This can be done with the *Spawn an object* action in Player, which creates a new instance of an object at the same position and angle. The Bullet movement we added earlier will then make it fly out forwards. Make the following event:

Condition: Mouse►On click►Left clicked (the default)

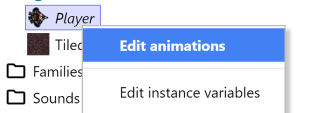
Action: Player►Spawn another object, then for *Object*, choose the *Bullet* object. Leave the other parameters as they are.

Your event should now look like this:

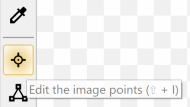
[The event to shoot a bullet](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1120/gstut-event2.png)

If you run the game, you can shoot bullets! However you may notice the bullets shoot from the middle of the player, rather than from the end of the gun. Let's fix that by placing an image point at the end of the gun. An image point is just a position on an image that you can spawn objects from, and we can reference it in the *Spawn another object* action.

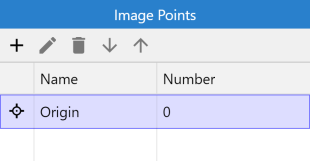
**Right-click** the player in the Project Bar and select **Edit animations**.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1121/edit-animations.png)

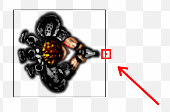
The image editor for the player reappears. Click the origin and image points tool:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1122/imagepoints-tool.png)

...and then the side pane turns in to a list of image points:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1123/imagepoints-pane.png)

Notice the object origin appears in the list. That's the "hotspot" or "pivot point" of the object. If you rotate the object, it spins around the origin. That's also what's used when you spawn an object at image point 0, as our action did. We want to add another image point to represent the gun, so **right-click** in the list and select Add a new image point. A new item appears in the list, and an icon appears over the image to indicate where this image point is. Left-click at the end of the player's gun to place the image point there:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1124/placing-imagepoint.png)

Close the image editor. **Double-click** the *Spawn an object* action we added earlier, and change the *Image point* to 1. The event should now look like below - note it says *Image point 1* now:

[Event spawning at an image point](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1125/gstut-event2b.png)

Preview the game again. The bullets now shoot from the end of your gun! The bullets don't do anything yet, though. Hopefully, however, you'll start to realise that once you get to grips with the event system, you can put together your game's logic very quickly.

Let's make the bullets kill monsters. Add the following event:

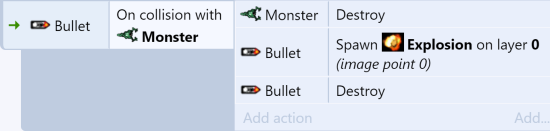
Condition: Bullet►On collision with another object►Monster.

Action: Monster►Destroy

Action: Bullet►Spawn another object►Explosion

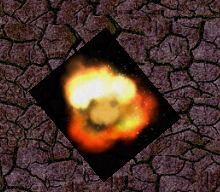
Action: Bullet►Destroy

Here's what the finished event looks like.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1126/gstut-event3.png)

**The explosion effect**

Preview the game, and try shooting a monster. Oops, the explosion has that big black border!

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1127/explosionnoadditive.png)

You might have predicted it'd look like that from the start, and wondered if our game was really going to end up like that! Don't worry, it won't. **Click the Explosion object** in the Project Bar. Its properties appear in the Properties Bar on the left. In the *Effects* section, set its **Blend mode** to **Additive**. Now try the game again.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1128/explosionadditive.png)

Why does this work? Without going too much in to the nuts and bolts, ordinary images are pasted on top of the screen. With the additive blend mode, each pixel is instead added (as in, summed) with the background pixel behind it. Black is a zero pixel value, so nothing gets added - you don't see the black background. Brighter colors add more, so appear more strongly. It's great for explosions and lighting effects.

**Making monsters a little smarter**

Right now the monsters just wander off the layout to the right. Let's make them a bit more interesting. First of all, let's start them at a random angle.

Condition: System►On start of Layout

Action: Monster►Set angle to random(360)

They will still wander off forever when they leave the layout, never to be seen again. Let's keep them inside. What we'll do is point them back at the player when they leave the layout. This does two things: they always stay within the layout, and if the player stands still, monsters end up coming right for them!

Condition: Monster►Is outside layout

Action: Monster►Set angle toward position X: Player.X Y: Player.Y

Here are what the two finished events look like.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1129/gstut-event4.png)

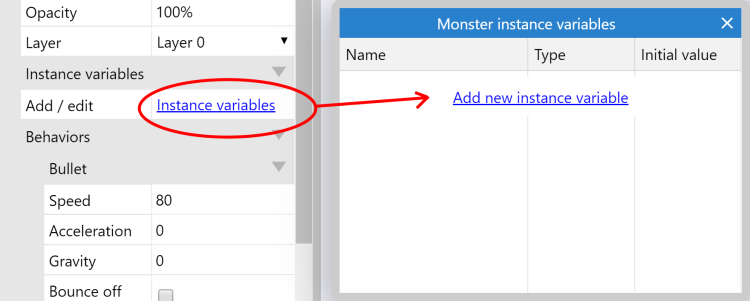
Run the game. If you hang around for a while, you'll notice the monsters stay around the layout too, and they're going in all kinds of directions. It's hardly AI, but it'll do!

Now, suppose we want to have to shoot a monster five times before it dies, rather than instant death like it is at the moment. How do we do that? If we only store one "Health" counter, then once we've hit a monster five times, all the monsters will die. Instead, we need *each* monster to remember its *own* health. We can do that with **instance variables.**

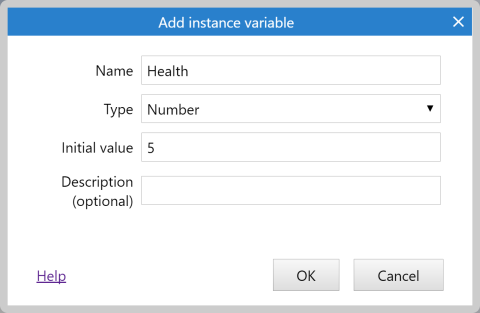
# Using instance variables

Instance variables allow each monster to store its own health value. A variable is simply a value that can change (or vary), and they are stored separately for each instance, hence the name instance variable.

Lets add a health instance variable to our monster. This works somewhat similarly to adding a behavior. Click the monster in the Project Bar. Alternatively, you can switch back to the layout using the tabs at the top and select a monster object. This will show the monster's properties in the Properties Bar. Click **Instance variables** to open the Instance Variables dialog.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1130/instvars-dlg.png)

You can add as many instance variables to an object as you like, but we only need one for the monster. Click **Add new instance variable**. The following dialog appears for adding an instance variable.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1131/add-health-var.png)

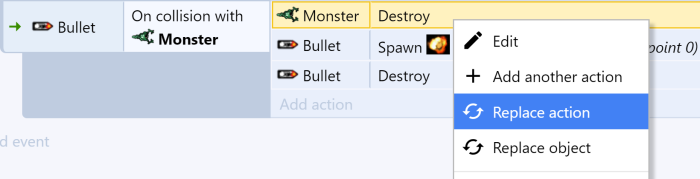
Type Health for the name, leave Type as **Number**, and for Initial value enter **5** (as shown). This starts every monster on 5 health. When they get hit we'll subtract 1 from the health, and then when health is zero we'll destroy the object.

Once you're done click OK. Notice the variable now appears in the instance variables dialog and also in the properties for the monster as well. You can quickly change initial values in the properties bar, but to add or remove variables you'll need to open the instance variables dialog. Also note every object in the layout can have unique instance variable values set as well, so you could for example start every monster with a different amount of health.

### Changing the events

Switch back to the event sheet. Right now, we're destroying monsters as soon as the bullet hits them. Let's change that to subtract 1 from its health.

Find the event that reads Bullet: On collision with Monster. Notice we've got a "destroy monster" action. Let's replace that with "subtract 1 from health". Right click the "destroy monster" action and click **Replace action**.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1132/replace-action.png)

The same dialog appears as if we were inserting a new action, but this time it'll replace the action we clicked instead. Choose Monster►Subtract from (in the Instance variables category), choose the instance variable "health", and enter 1 for the value. Click **Done**. The event should now look like this:

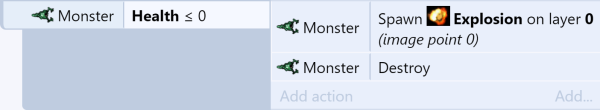
[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1133/gstut-event5.png)

Now when we shoot monsters they lose 1 health and the bullet explodes, but we haven't made an event to kill monsters when their health reaches zero. Add another event:

Condition: Monster►Compare instance variable►Health, Less or equal, 0

Action: Monster►Spawn another object►Explosion

Action: Monster►Destroy

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1134/gstut-event6.png)

Why "less or equal 0" rather than "equals 0"? Suppose we added another more powerful weapon which subtracted **2** from health. As you shot a monster, its health would go **5, 3, 1, -1, -3...** notice at no point was its health directly equal to zero, so it'd never die! Therefore, it's good practice to use "less or equal" to test if something's health has run out.

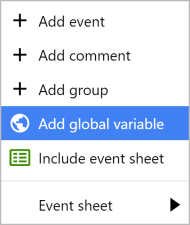
Run the game. You now have to hit monsters five times to kill them!

# Keeping score

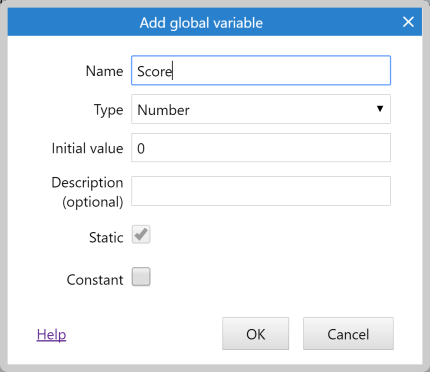
Let's have a score so the player knows how well they've done. We'll need another variable for this. You might think "lets put the score as one of the player's instance variables!". That's not a bad first idea, but remember the value is stored "in" the object. If there are no instances, there are no variables either! So if we destroy the player, we can no longer tell what their score was, because it was destroyed with the player.

Instead, we can use a **global variable.** Like an instance variable, a global variable (or just "global") can store text or a number. Each variable can store a single number or a single piece of text. Global variables are also available to the entire game across all layouts - convenient if we were to add other levels.

**Right-click** the space at the bottom of the event sheet, and select Add global variable.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1142/add-global.png)

Enter **Score** as the name. The other field defaults are OK, it'll make it a number starting at 0.

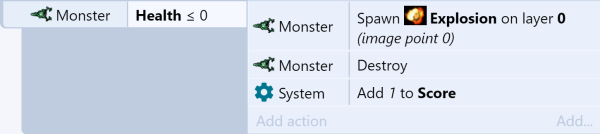
[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1143/adding-score.png)

Now the global variable appears as a line in the event sheet. It's in this event sheet, but it can be accessed from any event sheet in any layout.

[The 'Score' global variable](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1144/score-var.png)

There are also local variables which can only be accessed by a smaller "scope" of events, but we don't need to worry about that right now.

Let's give the player a point for killing a monster. In our "Monster: health less or equal 0" event (when a monster dies), click **Add action**, and select System►Add to (under Global & local variables), and choose **Score** with a value of 1. Now the event should look like this:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1145/gstut-event7.png)

Now the player has a score, which increases by 1 for every monster they kill - but they can't see their score! Let's show it to them with a text object.

# Displaying the score

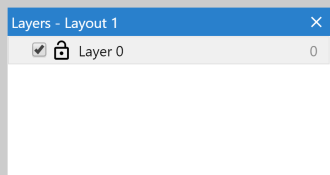
To display the player's score, we'll use a **Text** object. However we want to display it on a fixed position on-screen. The view follows the player, and we don't want the score to disappear as the player walks away! To fix this, we'll need to add a new **layer**.

### Adding a layer

Layouts can consist of multiple layers, which you can use to group objects. Imagine layers like sheets of glass stacked on top of each other, with objects painted on each sheet. It allows you to easily arrange which objects appear on top of others, and layers can be hidden, locked, have parallax effects applied, and more. We want our score to display on top of everything else, and also to stay in the same place on screen, so we can add a new layer to do this.

Try not to confuse **layouts** and **layers**. The words look quite similar.

Turn your attention to the **Layers bar**. By default this is in the lower-right corner of the screen.

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1146/layers-bar.png)The Layers Bar

You should see **Layer 0** in the list.

Construct counts starting from zero, since it works better like that in programming.

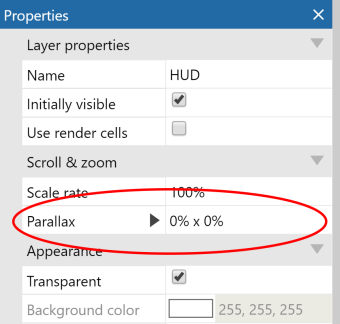
**Right-click** in the Layers Bar and select Add layer at top. (Be sure to add it at the top, not the bottom, because we want the score to display on top of everything else!) When you add it, you can immediately type in a name. Enter **HUD**, which stands for Heads-Up Display - which is a term used for all the on-screen information.

Now, **make sure the 'HUD' layer is selected in the Layers Bar.** This is important - the selected layer is the active layer. All newly added objects are added to the active layer, so if it's not selected, we'll later accidentally add our Text object to the wrong layer. The active layer is shown in the status bar in the bottom-right corner of the Layout View - it's worth keeping an eye on.

### Parallax

The HUD should always stay in the same place on the screen. By default, layers scroll as the view moves. To keep them on the screen, we can use the layer **Parallax** setting. Parallax allows different layers to scroll at different rates for a sort of semi-3D effect. If we set the parallax to zero, though, the layer won't scroll at all - ideal for a HUD.

Since you have the HUD layer selected, its properties should be showing in the Properties Bar. Set the **Parallax** property to **0 x 0** (that's zero on both the X and Y axes).

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1147/parallax-property.png)

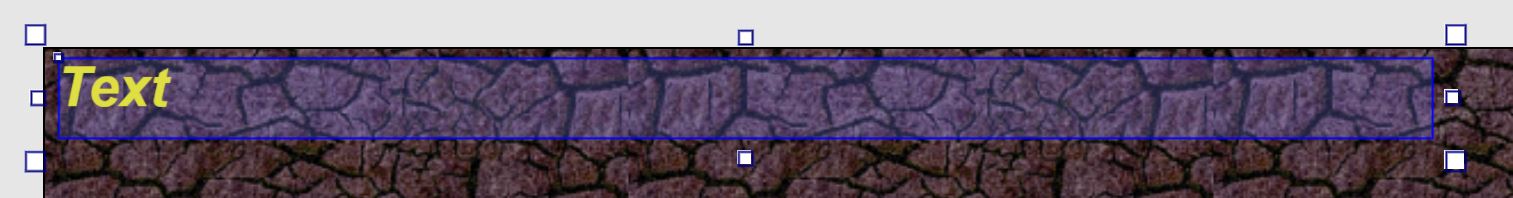
Now we have a layer where we can place objects that appear in a fixed place on-screen! However, we don't have any objects on it yet.

### Adding the Text object

**Switch back to the Layout View** using the tabs at the top. Make sure the HUD layer is selected in the Layers Bar, to ensure the Text object is added to the right layer. **Double-click** a space to add another object. This time pick the **Text** object.

Try not to confuse the **Text** object, which displays some text, with the **Text input** object, which is a box the user can type some text in to (e.g. for a form).

Place the Text object in the top left corner of the layout. It's going to be hard to see if it's black, so in the Properties Bar, make it bold, italic, yellow, and choose a slightly larger font size. Resize it wide enough to fit a reasonable amount of text. It should look something like this:

[](https://construct-static.com/images/v999/uploads/articleuploadobject/0/images/1148/text-hud.jpg)

Text objects won't display any text in the game if they are too small to fit the first word inside their box. If you don't see any text appear, try resizing the Text object larger.

Switch back to the event sheet. Let's keep the text updated with the player's score. In the **Every tick** event we added earlier, add the action Text►Set text.

Using the **&** operator, we can convert a number to text and join it to another text string. So for the text, enter:

"Score: " & Score

The first part ("Score: ") means the text will always begin with the phrase Score:. Pre-defined text has to appear in double-quotes in expressions. The second part (Score) is the actual value of the Score global variable. The **&** joins them together in to one piece of text.

Run the game, and shoot some monsters. Your score is displayed, and it stays at the same place on the screen!

# Finishing touches

We're nearly done. Let's add some final touches.

Firstly, let's have some monsters regularly spawning, otherwise once you've shot all the monsters there's nothing left to do. We'll create a new monster every 3 seconds. Add a new event:

Condition: System►Every X seconds►3

Action: System►Create object, **Monster**, layer **0** (that's the main layer), X: LayoutWidth + 100, Y: random(LayoutHeight)

Note the position is a random place along the right edge of the layout. That's where monsters will come from. We also create them 100 pixels further to the right so the player can't see them suddenly appear at the edge, they will instead come in from the outside.

Finally, let's have ghosts kill the player.

Condition: Monster►On collision with another object►Player

Action: Player►Destroy

**Some ideas to try**

Want to go a bit further? Here are some ideas for some extra things to add:

* Make the player get points for hitting monsters as well as killing them. You can adjust the score gained for each case.
* Make the monsters slowly speed up over time, so they're harder to hit and avoid.
* Add another kind of enemy!
* Add an alternative kind of weapon, which uses a different mouse button or keyboard control.
* Add the Audio object, import some sound files, and add sound effects or music.
* Add a title screen. Use the System object *Go to layout* action to switch between them.
* Introduce some scenery or obstacles in the level design.
* Add a "Game over" screen, or make something else happen when the player dies.

**Conclusion**

Congratulations, you've made your first game in Construct! If you want to show off your work, use Menu►Project►Export. You can publish to the Scirra Arcade, or upload to your own web server with the Web (HTML5) export. There are other platforms you can publish to, but you will need a [subscription](https://www.construct.net/make-games/buy-construct-3) to access the full features of Construct 3.

You've learnt some important basics about Construct: adding objects, using behaviors, events, layers and more. Hopefully this should leave you well prepared to learn more about Construct! Try exploring its features and see what it can do for you.

**The finished thing**

[Click here to open the finished game](https://editor.construct.net/#open=ghost-shooter-tut), or search for *Ghost shooter tutorial* in the Start Page. It's got some extra features added in like some "Game over" text, and monsters which gradually speed up. There are also lots of comments (simply notes you can add above events) describing how it works.

If you liked this tutorial and think someone you know might also like Construct, why not send them a link to this tutorial? You can help them out if they get stuck ;)

**What next?**

Want to go further? Here are some ideas on where to go next.

* Take a look at the tutorial [How to learn Construct 3? Next steps for beginners](https://www.construct.net/en/tutorials/how-to-learn-construct-3-next-steps-for-beginners-45). This is a great guide to follow to continue your journey learning Construct 3!
* Try some other tutorials in the [tutorials section](https://www.construct.net/tutorials) of the website!
* Take a look at the [Construct 3 manual](https://www.construct.net/make-games/manuals/construct-3), which includes a section with more information on [How events work](https://www.construct.net/make-games/manuals/construct-3/project-primitives/events/how-events-work).
* Browse the examples in the Start Page. They show off all kinds of features in Construct.
* Try out the demo games in the Start Page. You can also try to "hack" them, like trying to figure out what to change in the events to give yourself unlimited lives, faster movement or more powerful weapons.
* Get involved in the [community](https://www.construct.net/community)!
* [Purchase a subscription](https://www.construct.net/make-games/buy-construct-3) and try out the full features of Construct 3!