This is the Construct 2 Ghost shooter tutorial from the Scirra.com website.
It's by Ashley, and was originally published in 2011.

https://www.construct.net/en/tutorials/beginners-guide-construct-47

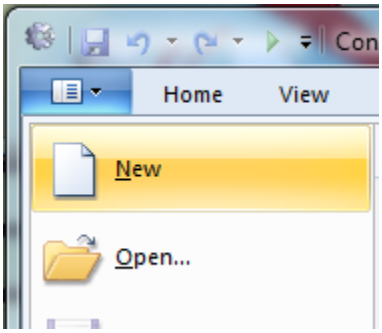
You can get all the images you need in the GHOST SHOOTER folder in the handout folder.

Installing Construct 2

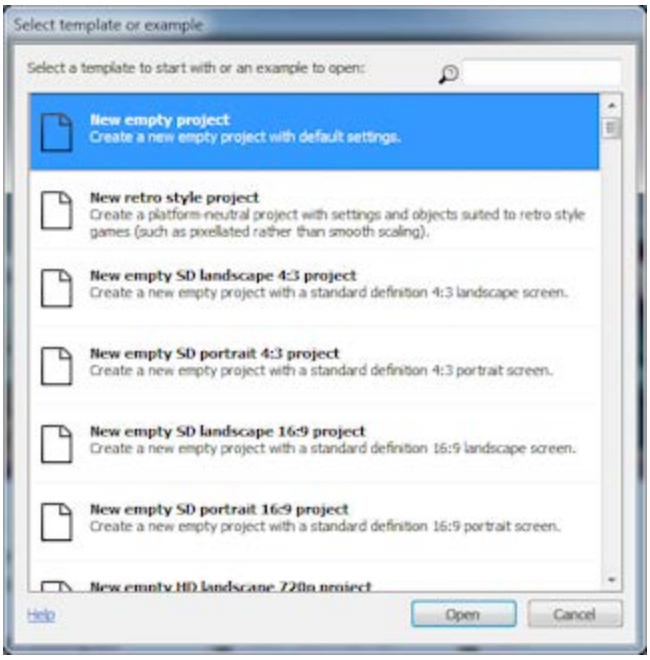
If you haven't already, grab a copy of the latest release of Construct 2 here. The Construct 2 editor is for Windows only, but the games you make can run anywhere, such as Mac, Linux or iPad. Construct 2 can also be installed on limited user accounts. It's also portable, so you can install to a USB memory stick for example, and take it with you!

Getting started

Now you're set up, launch Construct 2. Click the *File* button, and select *New*.



You will see the 'Template or Example' dialog box.



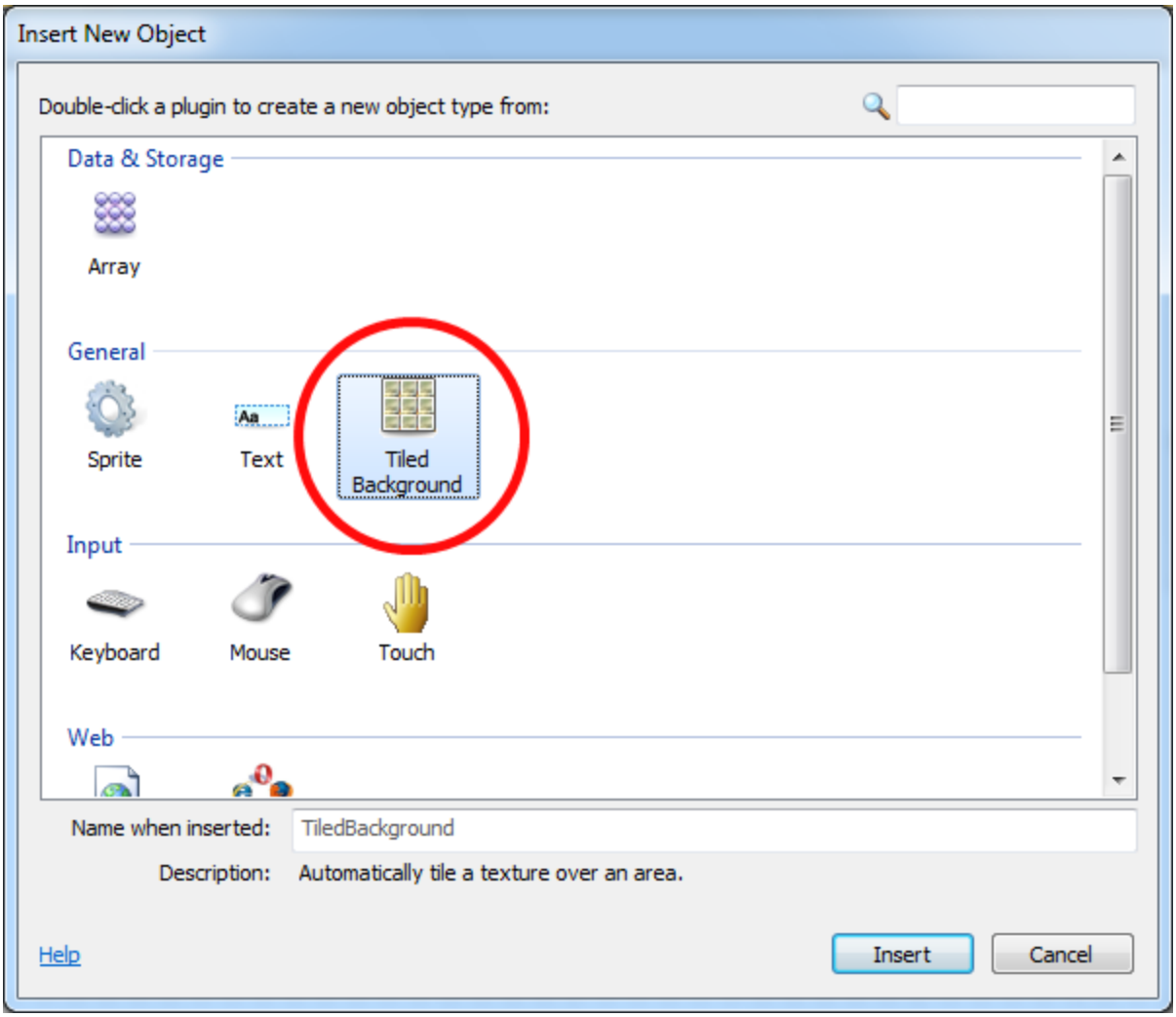
This shows a list of examples and templates that you can investigate at your leisure. For now, just click on 'Open' at the bottom of the box to create a blank, empty new project. Construct 2 will keep the entire project in a single *.capx* file for us. You should now be looking at an empty *layout* - the design view where you create and position objects. Think of a layout like a game level or menu screen. In other tools, this might have been called a *room*, *scene* or *frame*.

Inserting objects

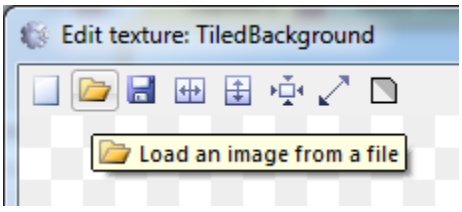
Tiled Background

The first thing we want is a repeating background tile. The *Tiled Background* object can do this for us. First, here's your background texture - right click it and save it to your computer somewhere:

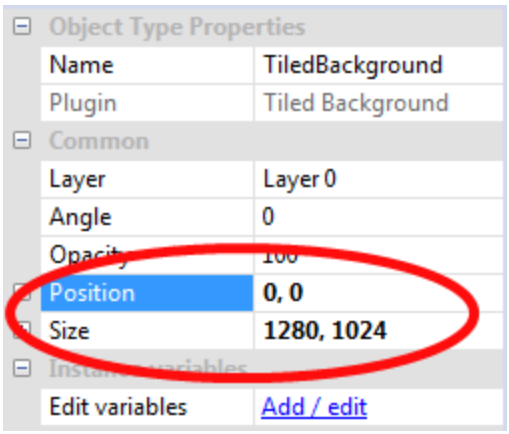
Now, **double click** a space in the layout to insert a new object. (Later, if it's full, you can also right-click and select *Insert new object*.) Once the *Insert new object* dialog appears, **double click the Tiled Background object** to insert it.



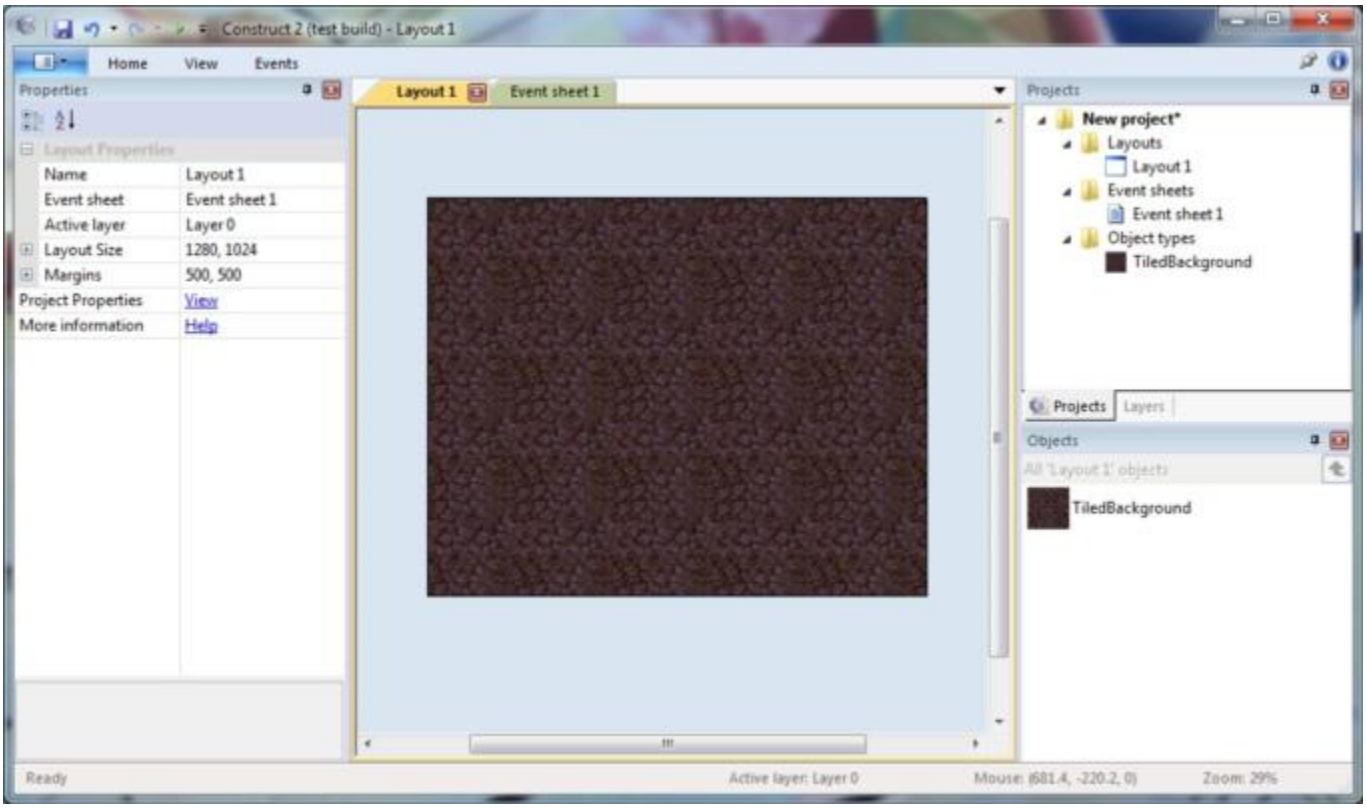
A crosshair will appear for you to indicate where to place the object. Click somewhere near the middle of the layout. The *texture editor* now opens, for you to enter the texture to tile. Let's import the tile image you saved earlier. Click the folder icon to load a texture from disk, find where you downloaded the file to, and select it.



Close the texture editor by clicking the X in the top right. If you're prompted, make sure you save! Now you should see your tiled background object in the layout. Let's resize it to cover the entire layout. Make sure it's selected, then the *Properties Bar* on the left should show all the settings for the object, including its size and position. Set its position to 0, 0 (the top left of the layout), and its size to 1280, 1024 (the size of the layout).



Let's survey our work. Hold **control** and scroll the **mouse wheel down** to zoom out. Alternatively, click *view - zoom out* a couple of times. You can also hold space, or the middle mouse button, to pan around. Neat, huh? Your tiled background should cover the entire layout now:



Hit control+0 or click *view - zoom to 100%* to return to 1:1 view.

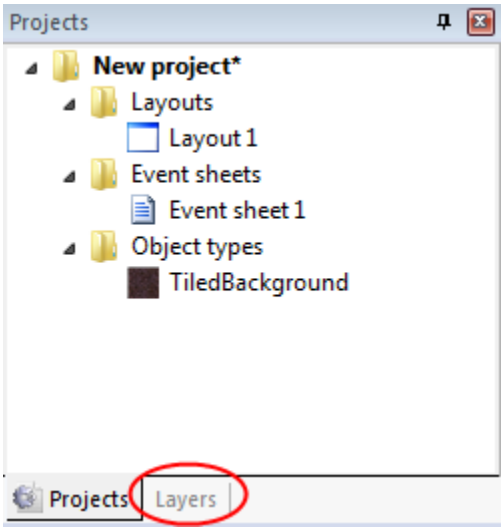
(If you're impatient like me, click the little 'run' icon in the window title bar - a browser should pop up showing your tiled layout! Woo!)

Adding a layer

Okay, now we want to add some more objects. However, we're going to keep accidentally selecting the tiled background unless we *lock* it, making it unselectable. Let's use the layering system to do this.

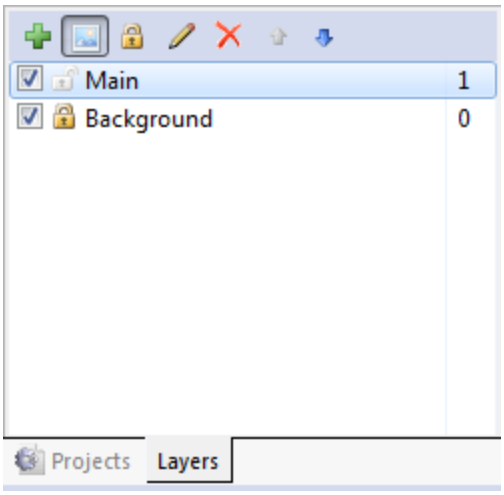
Layouts can consist of multiple *layers*, which you can use to group objects. Imagine layers like sheets of glass stacked on top of each other, with objects painted on each sheet. It allows you to easily arrange which objects appear on top of others, and layers can be hidden, locked, have parallax effects applied, and more. For example, in this game, we want everything to display above the tiled background, so we can make another layer on top for our other objects.

To manage layers, click the **Layers tab**, which usually is next to the *Project bar*:



You should see *Layer 0* in the list (Construct 2 counts starting from zero, since it works better like that in programming). Click the pencil icon and **rename it** to *Background*, since it's our background layer. Now click the green 'add' icon to add a new layer for our other objects. Let's call that one *Main*. Finally, if you click the little padlock icon next to *Background*, it will become *locked*. That means you won't be able to select anything on it. That's quite convenient for our tiled background, which is easy to accidentally select and won't need to be touched again. However, if you need to make changes, you can just click the padlock again to unlock.

The checkboxes also allow you to hide layers in the editor, but we don't need that right now. Your layers bar should now look like this:



Now, **make sure the 'Main' layer is selected in the layers bar**. This is important - the selected layer is the *active* layer. All new inserted objects are inserted to the *active* layer, so if it's not selected, we'll be accidentally inserting to the wrong layer. The active layer is shown in the status bar, and also appears in a tooltip when placing a new object - it's worth keeping an eye on.

Add the input objects

Turn your attention back to the layout. Double click to insert another new object. This time, select the **Mouse** object, since we'll need mouse input. Do the same again for the **Keyboard** object.

Note: these objects don't need placing in a layout. They are hidden, and automatically work project-wide. Now all layouts in our project can accept mouse and keyboard input.

The game objects

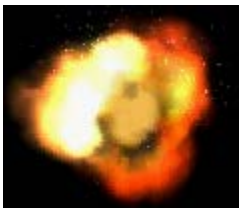
It's time to insert our game objects! Here are your textures - save them all to disk like before.

Player:

Monster:

Bullet:

and Explosion:



For each of these objects, we will be using a *sprite* object. It simply displays a texture, which you can move about, rotate and resize. Games are generally composed mostly out of sprite objects. Let's insert each of the above four objects as sprite objects. The process is similar to inserting the Tiled Background:

1. **Double click** to insert a new object

2. **Double click** the 'Sprite' object.

3. When the mouse turns to a crosshair, click somewhere in the layout. The tooltip should be 'Main'. (Remember this is the active layout.)

4. The texture editor pops up. Click the open icon, and **load one of the four textures**.

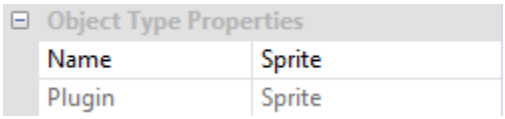
5. **Close** the texture editor, saving your changes. You should now see the object in the layout!

Note: another quick way to insert sprite objects is to drag and drop the image file from Windows in to the layout area. Construct 2 will create a Sprite with that texture for you. Be sure to drag each image in one at a time though - if you drag all four in at once, Construct 2 will make a single sprite with four animation frames.

Move the *bullet* and *explosion* sprites to somewhere off the edge of the layout - we don't want to see them when the game starts.

These objects will be called *Sprite*, *Sprite2*, *Sprite3* and *Sprite4*. That's not very useful - things will quickly get confusing like this. Rename them to *Player*, *Monster*, *Bullet* and *Explosion* as

appropriate. You can do it by selecting the object, then changing the **Name** property in the properties bar:



Adding behaviors

Behaviors are pre-packaged functionality in Construct 2. For example, you can add a *Platform* behavior to an object, and the *Solid* behavior to the floor, and you instantly can jump around like a platformer. You can do the same in events, but it takes longer, and there's no point anyway if the behavior is already good enough! So let's have a look at which behaviors we can use. Amongst others, Construct 2 has these behaviors;

- **8 Direction movement**. This lets you move an object around with the arrow keys. It will do nicely for the player's movement.

- **Bullet movement**. This simply moves an object forwards at its current angle. It'll work great for the player's bullets. Despite the name, it'll also work nicely to move the monsters around - since all the movement does is move objects forwards at some speed.

- **Scroll to**. This makes the screen follow an object as it moves around (also known as *scrolling*). This will be useful on the player.

- **Bound to layout**. This will stop an object leaving the layout area. This will also be useful on the player, so they can't wander off outside the game area!

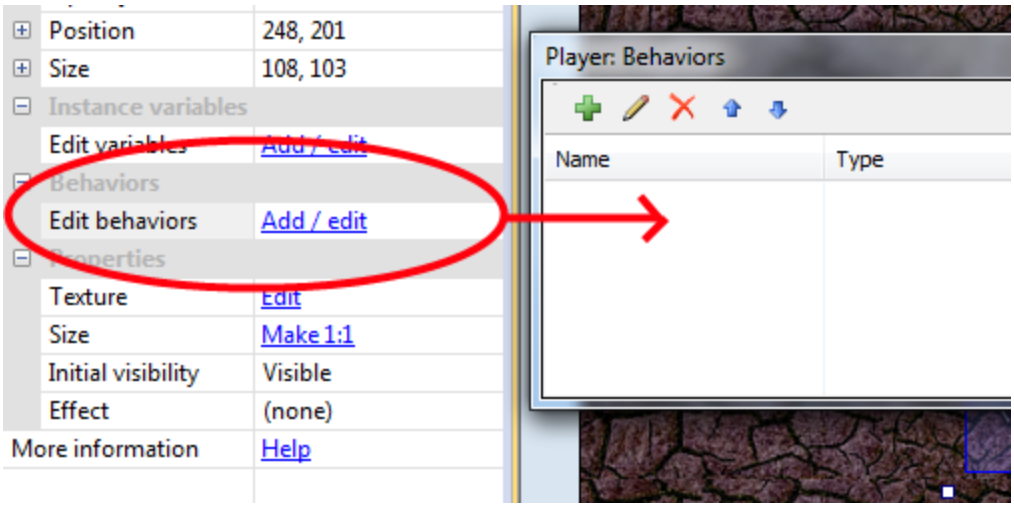
- **Destroy outside layout**. Instead of stopping an object leaving the layout area, this destroys it if it does. It's useful for our bullets. Without it, bullets would fly off the screen forever, always taking a little bit of memory and processing power. Instead, we should destroy the bullets once they've left the layout.

- **Fade**. This gradually makes an object fade out, which we will use on the explosions.

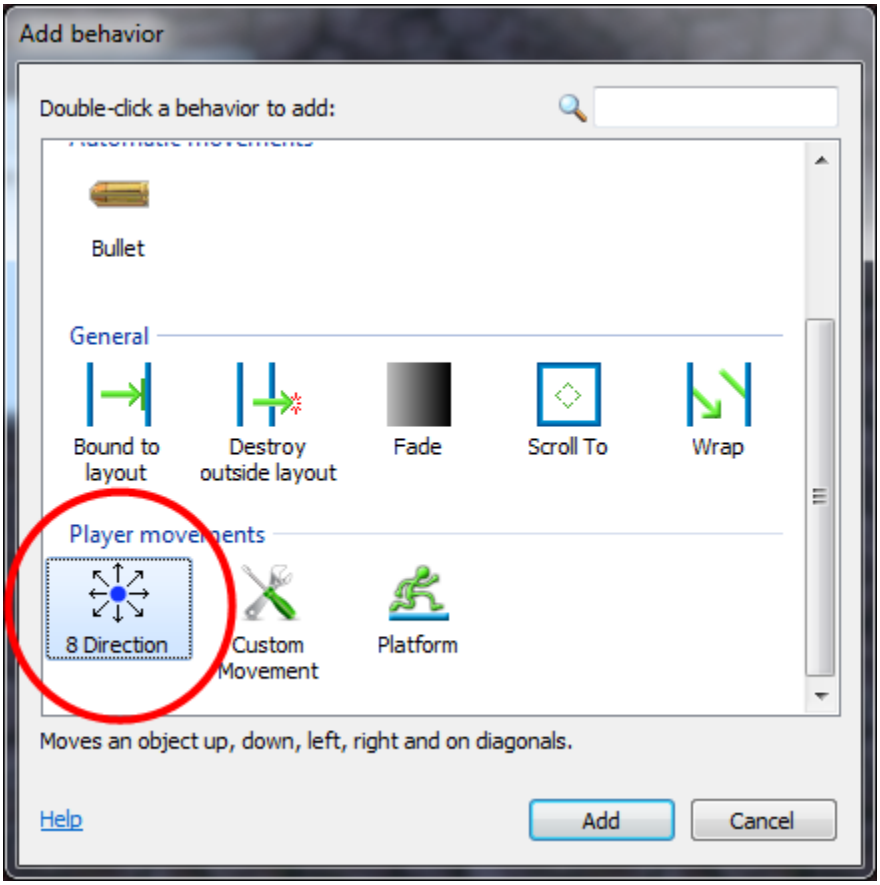
Let's add these behaviors to the objects that need them.

How to add a behavior

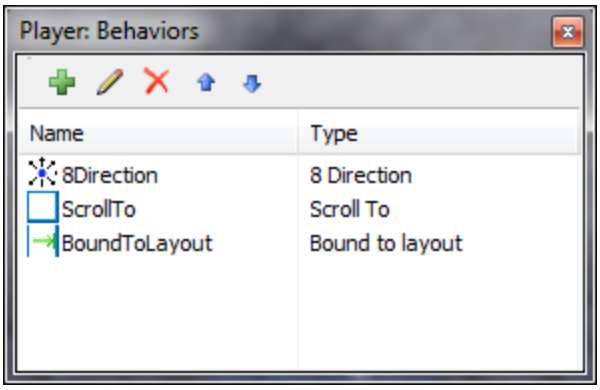
Let's add the **8 direction movement** behavior to the player. Click the player to select it. In the properties bar, notice the *Behaviors* category. Click *Add / Edit* there. The Behaviors dialog for the player will open.



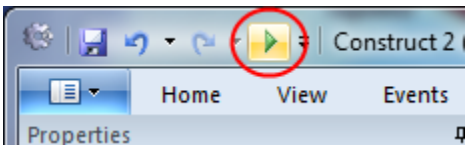
Click the green 'add behavior' icon in the behaviors dialog. Double-click the **8 direction movement** to add it.



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



Close the behaviors dialog. Hit Run to try the game!



Hopefully you have a HTML5 compatible browser installed. Otherwise, be sure to get the latest version of Firefox or Chrome, or Internet Explorer 9 if you're on Vista and up. Once you have the game running, 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 bits of functionality. We'll be using the event system soon to add customised functionality.

Adding the other behaviors

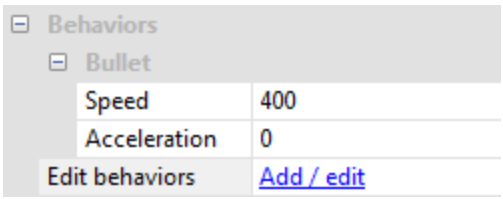
We can add behaviors to the other objects by the same method - select it, click *Add / Edit* to open the behaviors dialog, and add some behaviors. Let's add those other behaviors:

- Add the **Bullet movement** and **Destroy outside layout** to the **Bullet** object (no surprises there)

- Add the **Bullet movement** to the **Monster** object (because it just moves forwards as well)

- 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:



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

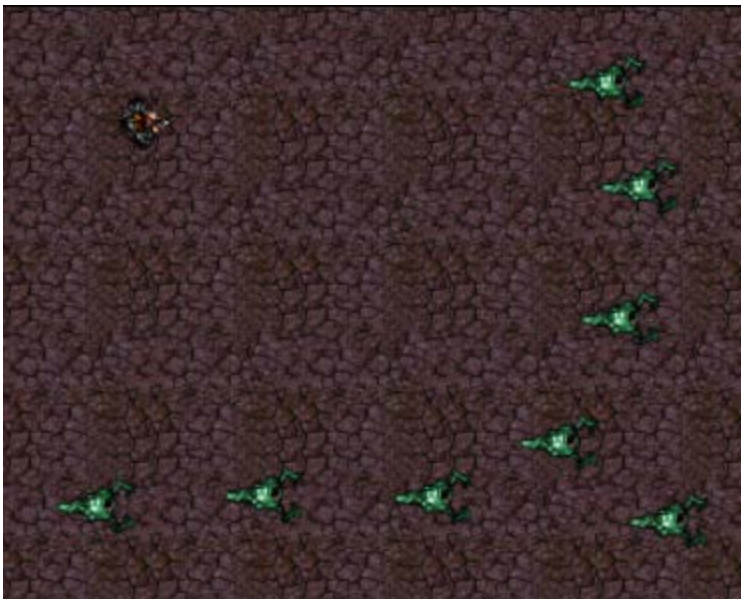
Similarly, change the **Bullet object's** speed to 600, and the **Explosion** object's Fade behavior's *Fade out time* to **0.5** (that's half a second).

Create some more monsters

Holding control, click and drag the **Monster** object. You'll notice it spawns 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. With Sprites, all instances of an object type also share the same texture. This is great for efficiency - when players play your game online, rather than having to download 8 monster textures for 8 monsters, they only need to download one monster texture and Construct 2 repeats it 8 times. 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.

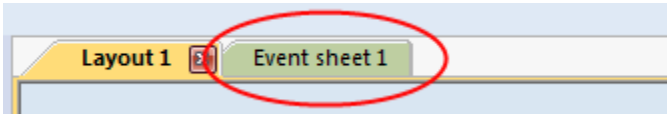
Using control + drag, **create 7 or 8 new monsters**. Don't place any too close to the player, or they might die straight away! You can zoom out with control + mouse wheel down if it helps, and spread them over the whole layout. You should end up with something a bit like this.



Now it's time to add our custom functionality via Construct 2's visual method of programming - the *event system*.

Events

First, click the *Event sheet 1* tab at the top to switch to the *Event sheet editor*. 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.



About events

As the text in the empty sheet indicates, Construct 2 runs everything in the event sheet once per tick. Most monitors update their display 60 times per second, so Construct 2 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 functionality 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?

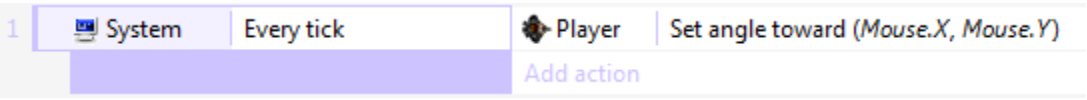
---> **Yes**: run all the event's actions.

---> **No**: go to next event (not including any sub-events).

That's a bit of an oversimplification. Construct 2 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.

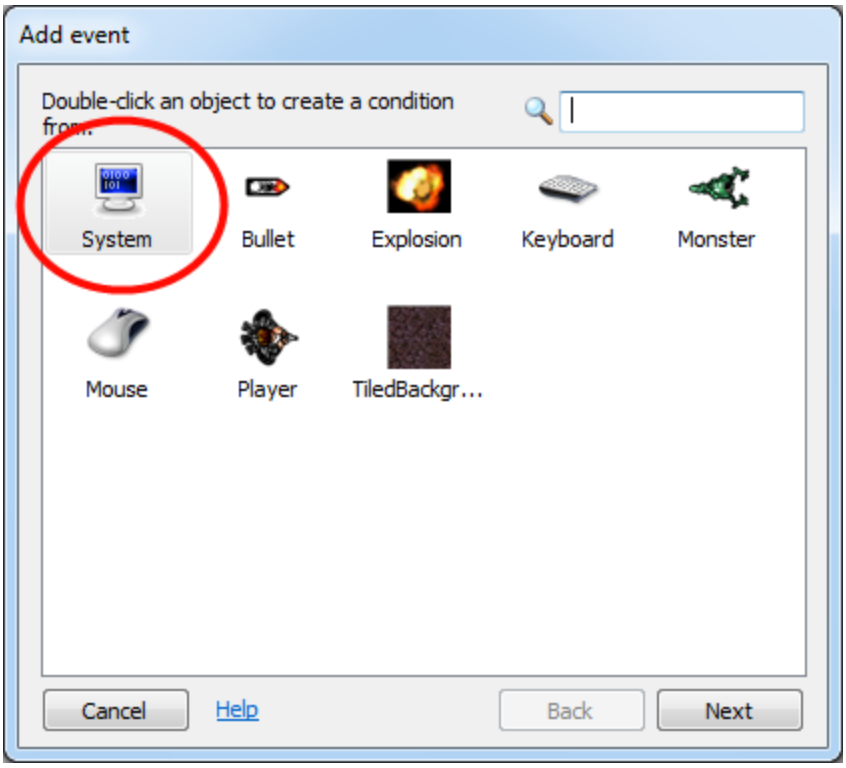
Your first event

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

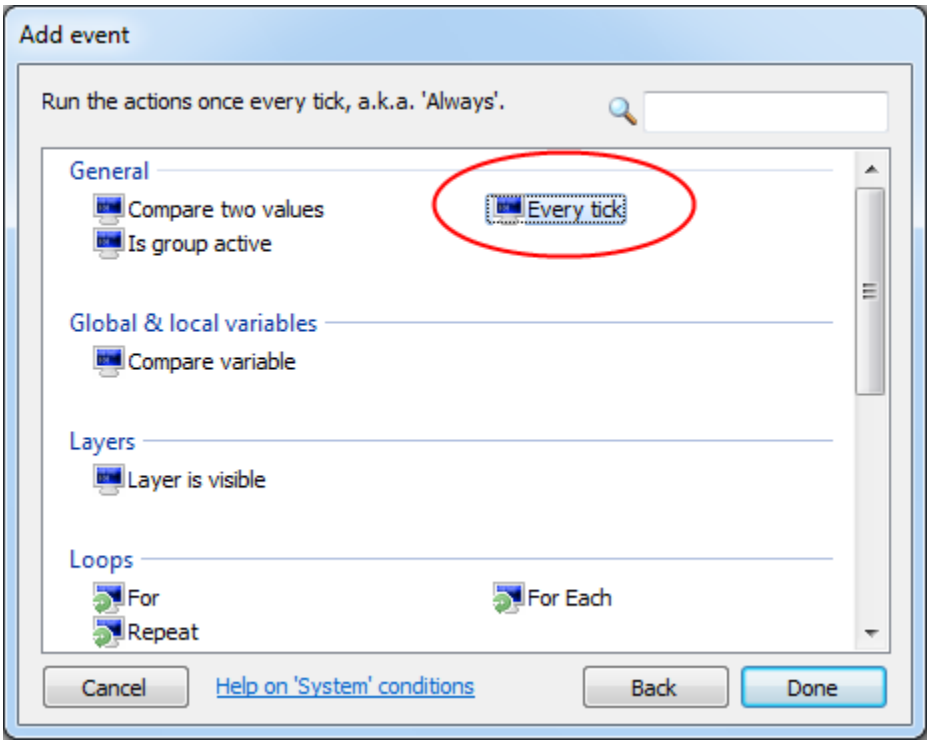


Remember a tick runs every time the screen is drawn, so if we make the player face the mouse every tick, they'll always appear to be facing the mouse.

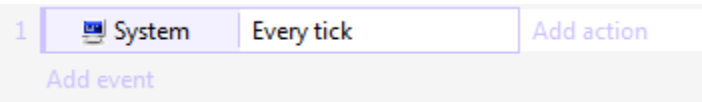
Let's start making this event. Double-click a space in the event sheet. This will prompt us to add a **condition** for the new event.



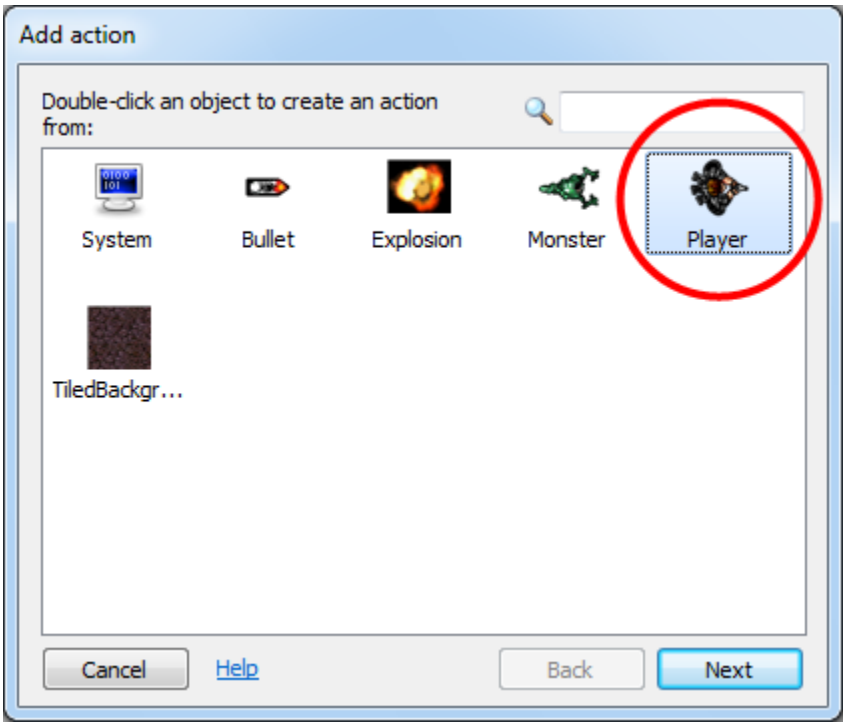
Different objects have different conditions and actions depending on what they can do. There's also the **System object**, which represents Construct 2's built-in functionality. **Double-click** the System object as shown. The dialog will then list all of the System object's conditions:



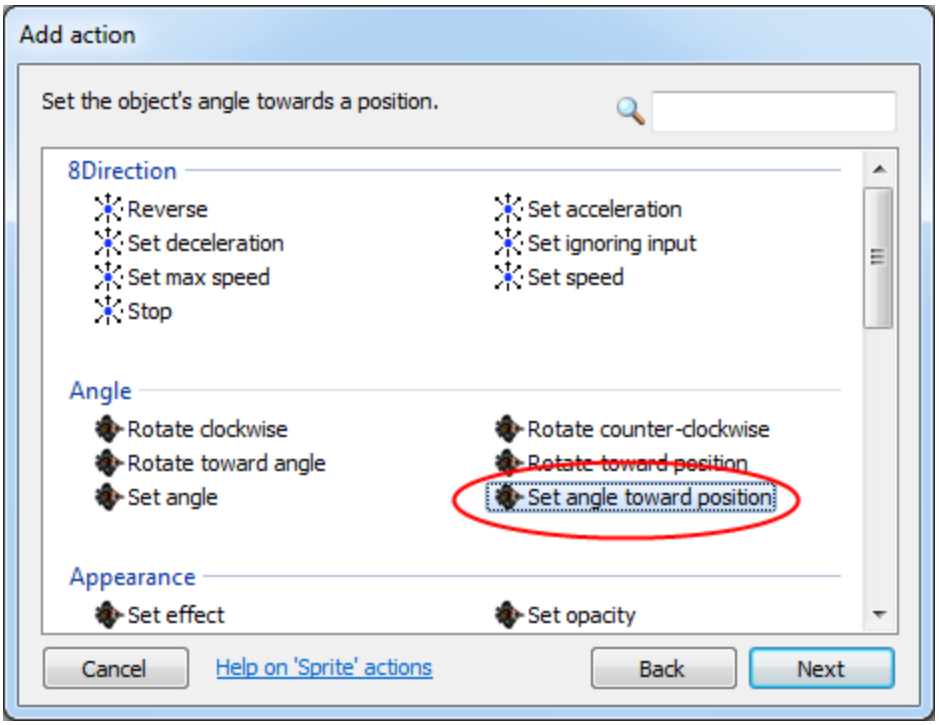
Double-click the *Every tick* condition to insert it. The dialog will close and the event is created, with no actions. It should now look like this:



Now we want to add an action to make the player look at the mouse. 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:



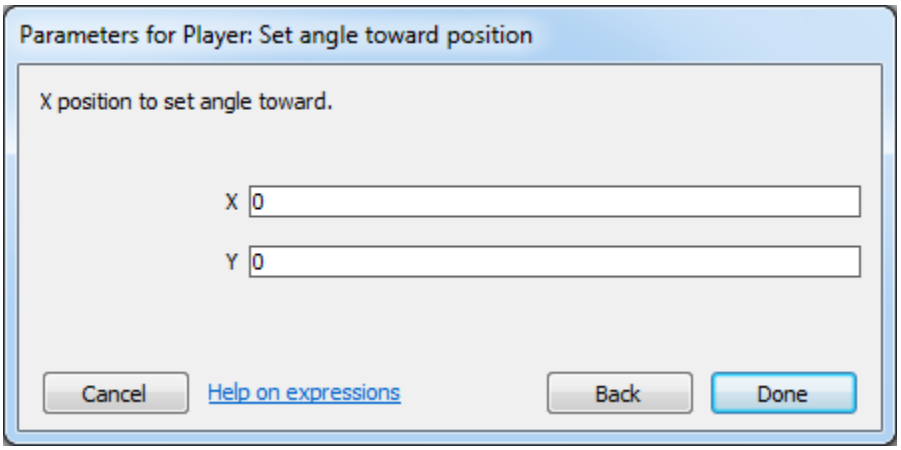
As with adding an event, we have our same list of objects to choose from, but this time for adding an *action*. Try not to get confused between adding conditions and adding actions! As shown, **double-click** the *Player* object, for it is the player we want to look at the mouse. The list of actions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



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.

Construct 2 now needs to know the X and Y co-ordinate to point the player at:

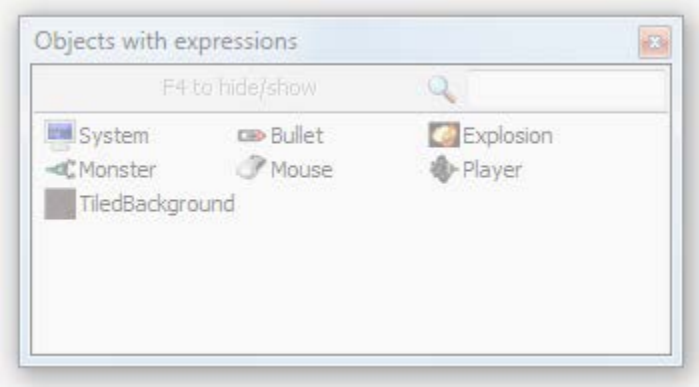


These are called the **parameters** of the action. Conditions can have parameters too, but *Every tick* doesn't need any.

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 2's flexibility.

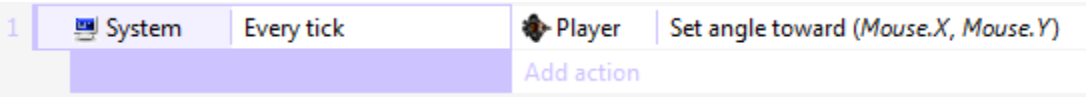
Did you get an error that said "Mouse is not an object name"? Make sure you added the Mouse object! Go back to page 2 and check under "Add the input objects".

You might be wondering how you'd remember all the possible expressions you could enter. Luckily, there's the "object panel" which you should see floating above it. By default, it's faded out so it doesn't distract you.



Hover the mouse over it, or click on it, and it'll become fully visible. This serves as a sort of dictionary of all the expressions you can use, with descriptions, to help you remember. If you double-click an object, you'll see all its expressions listed. 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:



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. This is our first bit of custom functionality.

Adding game functionality

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* -> X: *Mouse.X*, Y: *Mouse.Y*

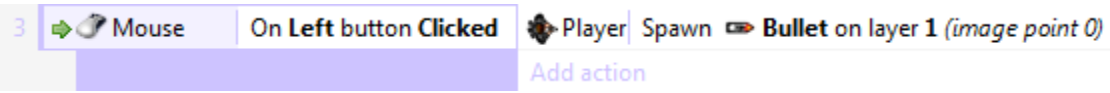
Get the player to shoot

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)

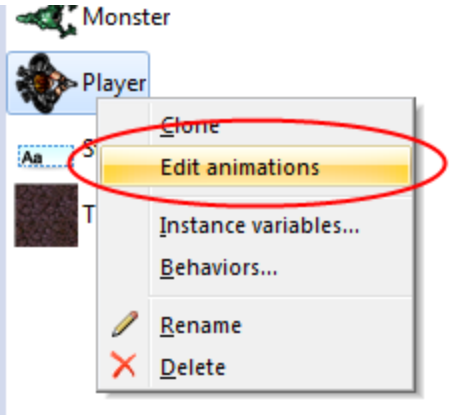
Action: *Player -> Spawn another object ->* For *Object*, choose the *Bullet* object. For *Layer*, put **1** (the "Main" layer is layer 1 - remember Construct 2 counts from zero). Leave *Image point* as 0.

Your event should now look like this:

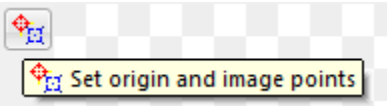


If you run the game, you'll notice the bullets shoot from the middle of the player, rather than from the end of the gun. Let's fix that by placing an **image point** at the end of the gun. (An image point is just a position on an image that you can spawn objects from.)

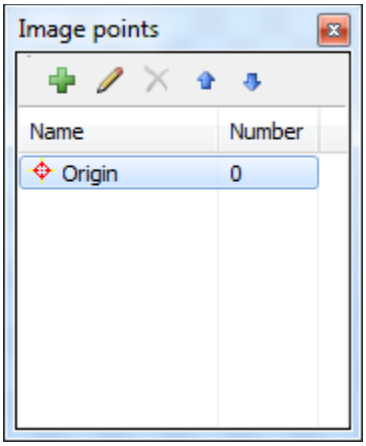
Right-click the player in the project or object bar and select **Edit animations**.



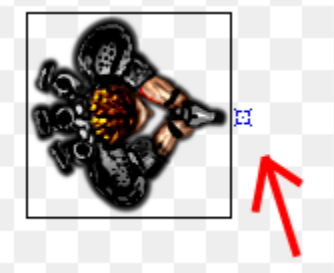
The image editor for the player reappears. Click the origin and image points tool:



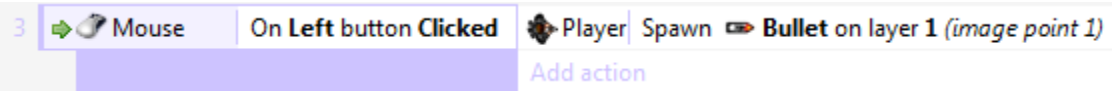
...and the image points dialog opens up:



Notice the object origin appears as a red spot. That's the "hotspot" or "pivot point" of the object. If you rotate the object, it spins around the origin. We want to add another image point to represent the gun, so click the green *add* button. A blue point appears - that's our new image point. Left-click at the end of the player's gun to place the image point there:



Close the image editor. Double-click the *Spawn an object* action we added earlier, and change the *Image point* to **1**. (The origin is always the first image point, and remember Construct 2 counts from zero.) The event should now look like below - note it says *Image point 1* now:



Run the game. 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 functionality together very quickly.

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

Condition: *Bullet -> On collision with another object ->* pick *Monster*.

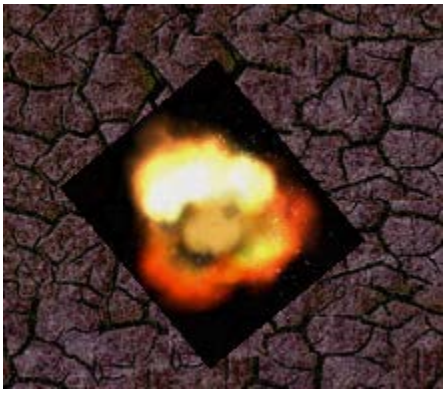
Action: *Monster -> Destroy*

Action: *Bullet -> Spawn another object -> Explosion*, layer **1**

Action: *Bullet -> Destroy*

The explosion effect

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



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 either the Object bar in the bottom right, or the Project bar (which was tabbed with the layers bar). Its properties appear in the properties bar on the left. At the bottom, set its **Blend mode** property to **Additive**. Now try the game again.

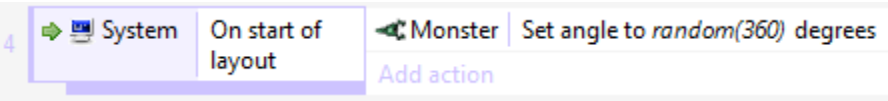
Why does this work? Without going in to the nuts and bolts, ordinary images are *pasted on top* of the screen. With the additive effect, each pixel is instead *added* (as in, summed) with the background pixel behind it. Black is a zero pixel value, so nothing gets added - you don't see the black background. 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 ->* 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 come right for them!

Condition: *Monster -> Is outside layout*

Action: *Monster -> Set angle toward position ->* For X, **Player.X** - for Y, **Player.Y**.

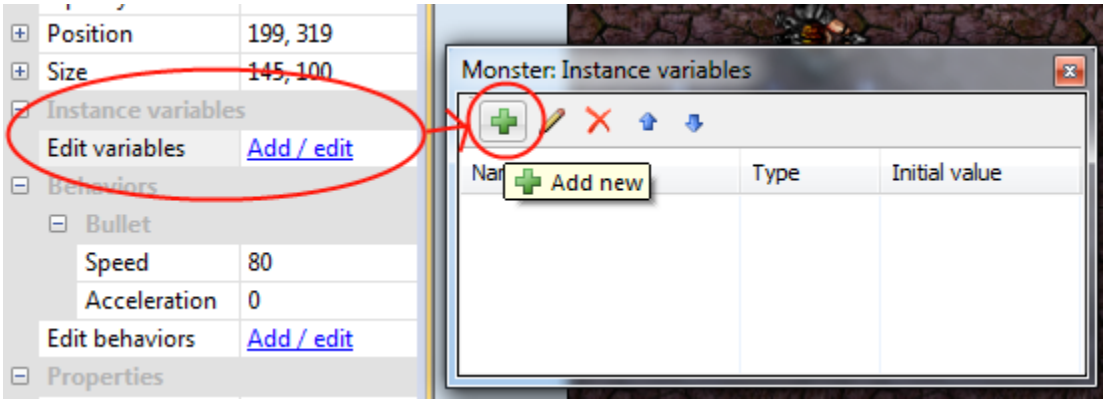
Run the game. If you hang around for a while, you'll notice the monsters stay around the layout too, and they're going in all kinds of directions. It's hardly AI, but it'll do!

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**.

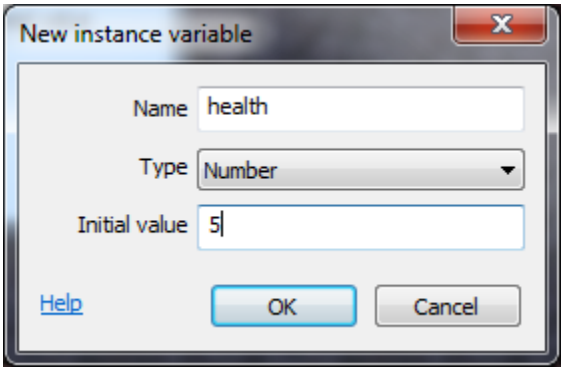
Instance variables

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. Click the monster in the project bar or object bar. Alternatively, you can switch back to the layout and select a monster object. This will show the monster's properties in the properties bar. Click **Add/edit** by **Edit variables**.

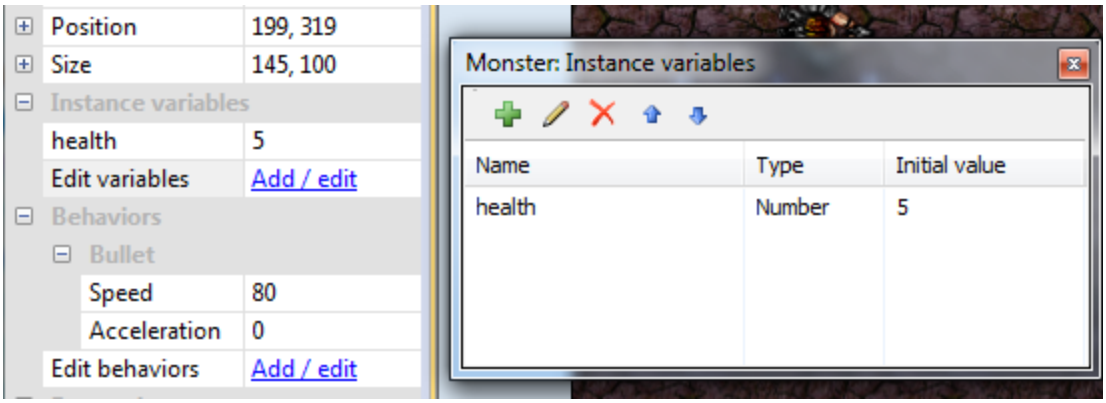


The Instance Variables dialog appears. It looks similar to the Behaviors dialog we saw earlier, but instead allows you to add and change instance variables for the object. Click the green **Add** button to add a new one.



In the dialog that pops up, 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.

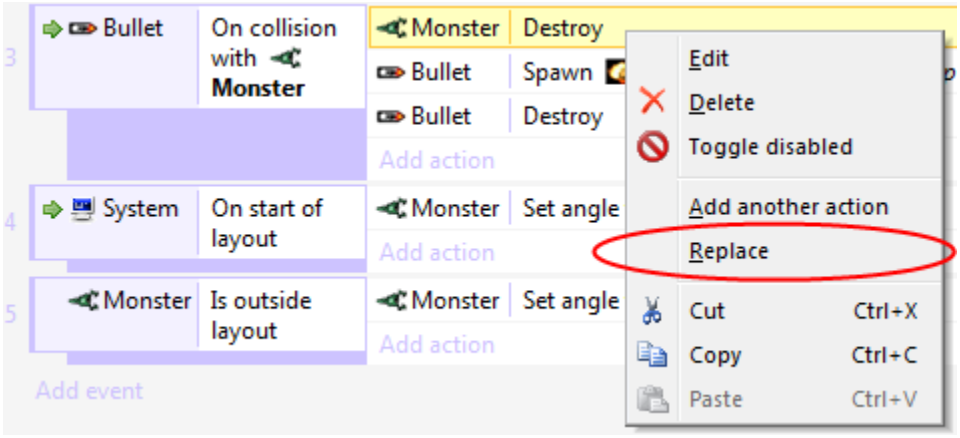
Once you're done click OK. Notice the variable now appears in the instance variables dialog and also in the properties for the monster as well. (You can quickly change initial values in the properties bar, but to add or remove variables you'll need to click the *Add / Edit* link.)



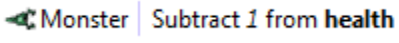
Changing the events

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**.



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) -> Instance variable "health", and enter **1** for *Value*. Click **Done**. The action should now appear like this:

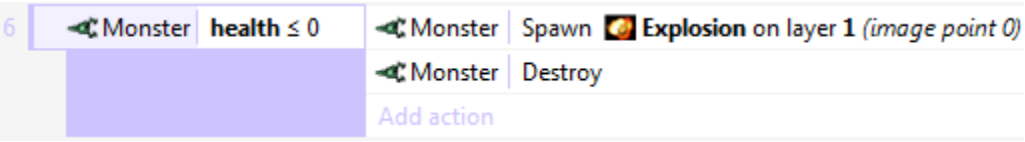


Now when we shoot monsters they lose 1 health and the bullet explodes, but we haven't made an event to kill monsters when their health reaches zero. Add another event:

Condition: *Monster -> Compare instance variable* -> Health, *Less or equal*, 0

Action: *Monster -> Spawn another object* -> Explosion, layer **1**

Action: *Monster -> Destroy*



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.

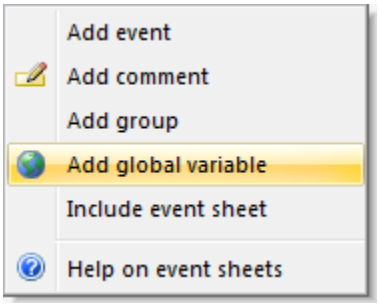
Run the game. You now have to hit monsters five times to kill them!

Keeping score

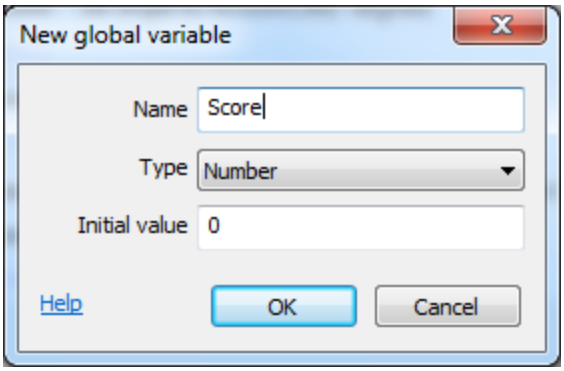
Let's have a score so the player knows how well they've done. We'll need another variable for this. You might think "lets put the score as one of the player's instance variables!". That's not a bad first idea, but remember the value is stored "in" the object. If there are no instances, there are no variables 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.

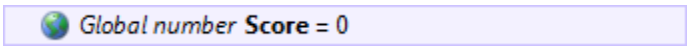
Right-click the space at the bottom of the event sheet, and select *Add global variable*.



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

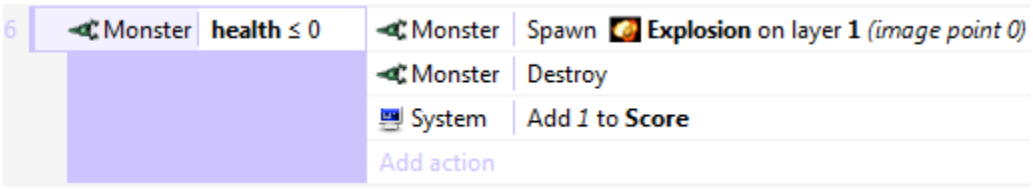


Now the global variable appears as a line in the event sheet. It's in this event sheet, but it can be accessed from any event sheet in any layout.



Note: 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) -> **Score**, value **1**. Now the event should look like this:



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.

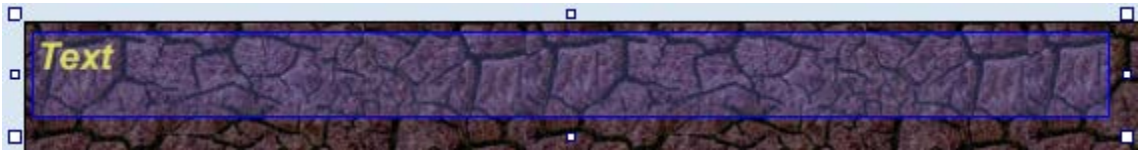
Creating a heads-up display (HUD)

A heads-up display (aka HUD) is the interface that shows the player's health, score and other information in-game. Let's make a really simple HUD out of a text object.

The HUD always stays the same place on the screen. If we have some interface objects, we don't want them to scroll away as the player walks around - they should stay on the screen. By default, layers scroll. 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.

Go back to the layers bar we used earlier. Add a new layer called *HUD*. Make sure it's at the top, and selected (remember this makes it the active layer). The Properties bar should now be displaying its properties. Set the **Parallax** property to **0, 0** (that's zero on both the X and Y axes).

Double-click a space to insert another object. This time pick the **Text** object. Place it 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:



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:*. 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 **1**, **1400** (for X), **random(1024)** (for Y)

1400 is an X co-ordinate just off the right edge of the layout, and *random(1024)* is a random Y co-ordinate the height of the layout.

Finally, let's have ghosts kill the player.

Condition: *Monster -> On collision with another object ->* **Player**

Action: *Player -> Destroy*

Conclusion

Congratulations, you've made your first HTML5 game in Construct 2! If you have a server and want to show off your work, click **Export** in the File menu. Construct can save all the project files to a folder on your computer, which you can upload or integrate to a web page. If you don't have your own server, you can share your games on Dropbox.

You've learnt some important basics about Construct2: inserting objects, using layers, behaviors, events and more. Hopefully this should leave you well prepared to learn more about Construct 2! Try exploring its features and see what it can do for you.

The finished thing

Try downloading the finished tutorial project. I've added in some extra features like some "Game over" text, and monsters which gradually speed up. Knowing what you know now, it shouldn't be hard to figure out how it works. There are also lots of comments describing how it works.

Well done! If you have any problems or you think any part of this tutorial could be improved, leave a comment or drop us a message on the forum. We'll see what we can do!

Finally, if you liked this tutorial and think someone you know might also like Construct 2, why not send them a link to this tutorial? Surely it can't hurt :)

Further reading

Want to add music and sound effects? See Sounds & Music in the manual for a quick overview.

You may be interested in our alternative platformer-based beginner's tutorial, How to make a platform game.

If you'd like to know more about how events work in Construct 2, see the section on How Events Work in the manual. It's highly recommended so you can get going quickly with your own projects! Then for even more information, don't forget there is complete documentation in the manual