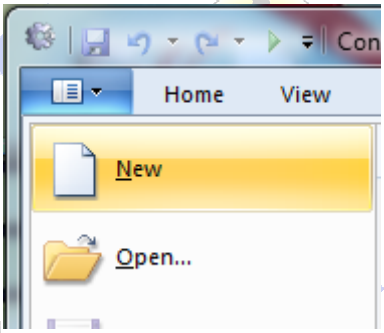
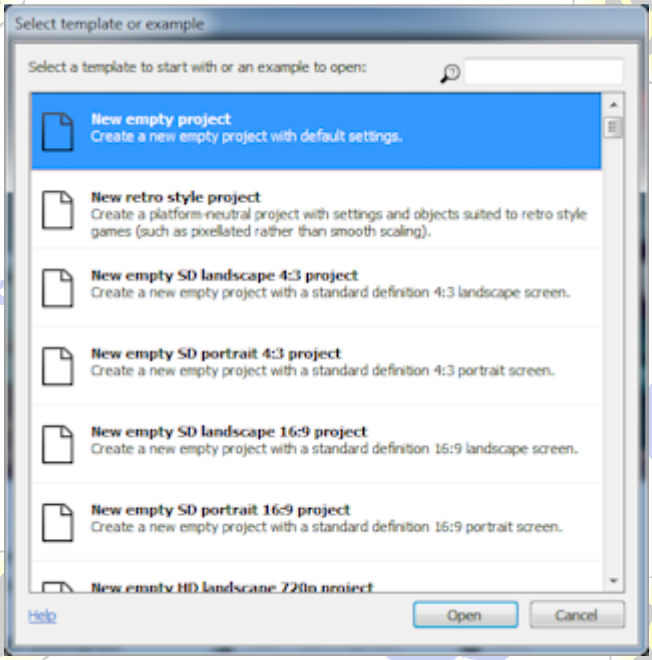
Getting started

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



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



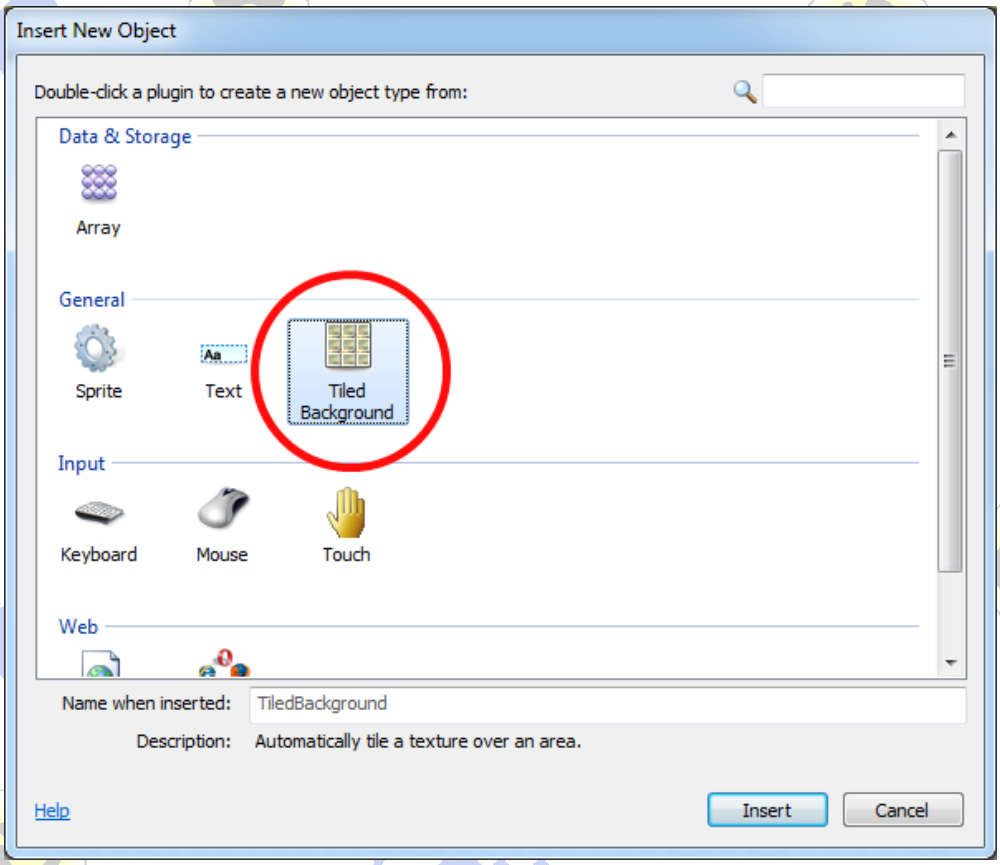
This shows a list of examples and templates that you can investigate at your leisure. For now, just click on 'Open' at the bottom of the box to create a blank, empty new project. Construct 2 will keep the entire project in a single *.capx* file for us. You should now be looking at an empty *layout* - 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

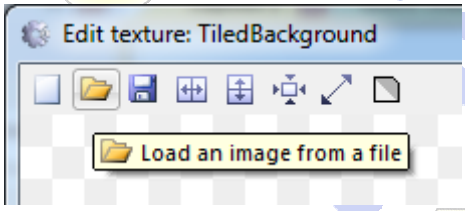
Tiled Background

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:

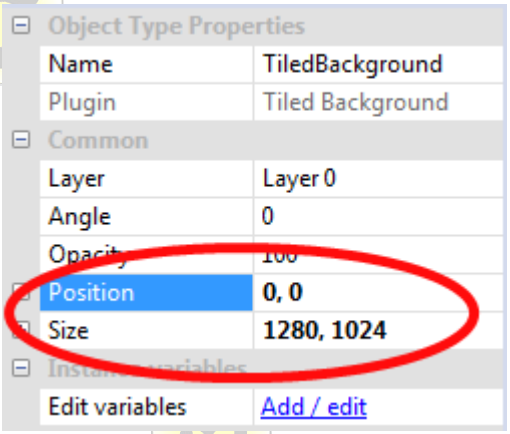
Now, **double click** a space in the layout to insert a new object. (Later, if it's full, you can also right-click and select *Insert new object*.) Once the *Insert new object*dialog appears, **double click the Tiled Background object** to insert it.



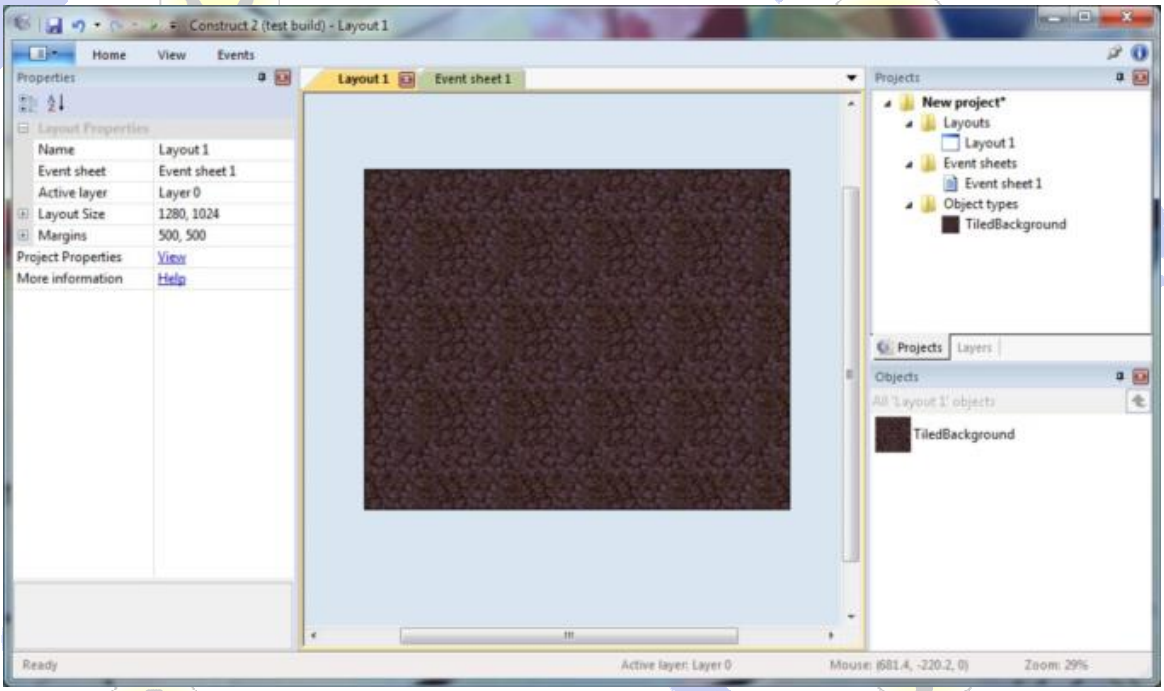
A crosshair will appear for you to indicate where to place the object. Click somewhere near the middle of the layout. The *texture editor* now opens, for you to enter the texture to tile. Let's import the tile image you saved earlier. Click the folder icon to load a texture from disk, find where you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



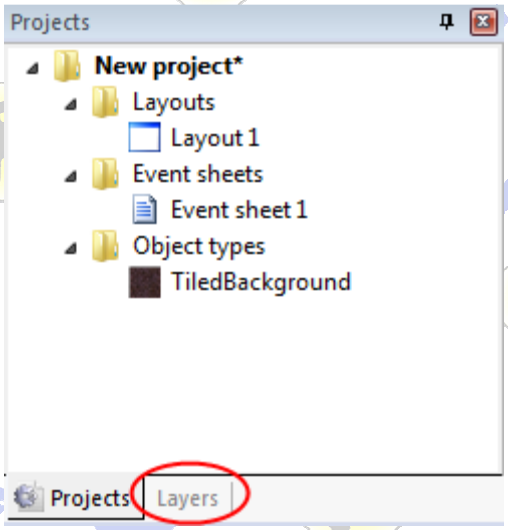
Hit control+0 or click *view - zoom to 100%* to return to 1:1 view.

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.

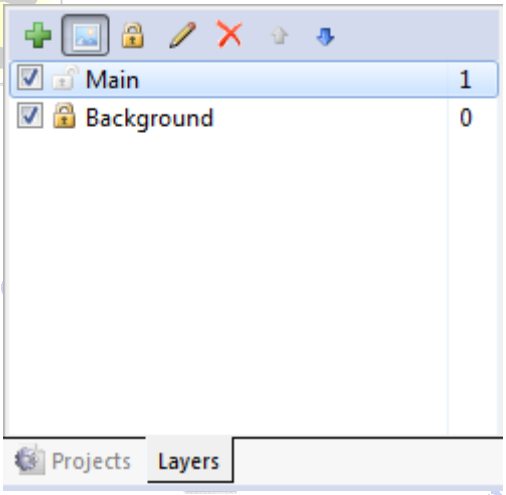
Layouts can consist of multiple *layers*, which you can use to group objects. Imagine layers like sheets of glass stacked on top of each other, with objects painted on each sheet. It allows you to easily arrange which objects appear on top of others, and layers can be hidden, locked, have parallax effects 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:

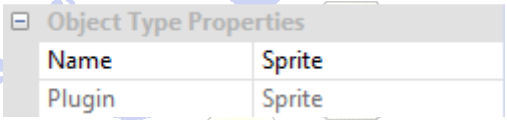
1. **Double click** to insert a new object
2. **Double click** the 'Sprite' object.
3. When the mouse turns to a crosshair, click somewhere in the layout. The tooltip should be 'Main'. (Remember this is the active layout.)
4. The texture editor pops up. Click the open icon, and **load one of the four 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

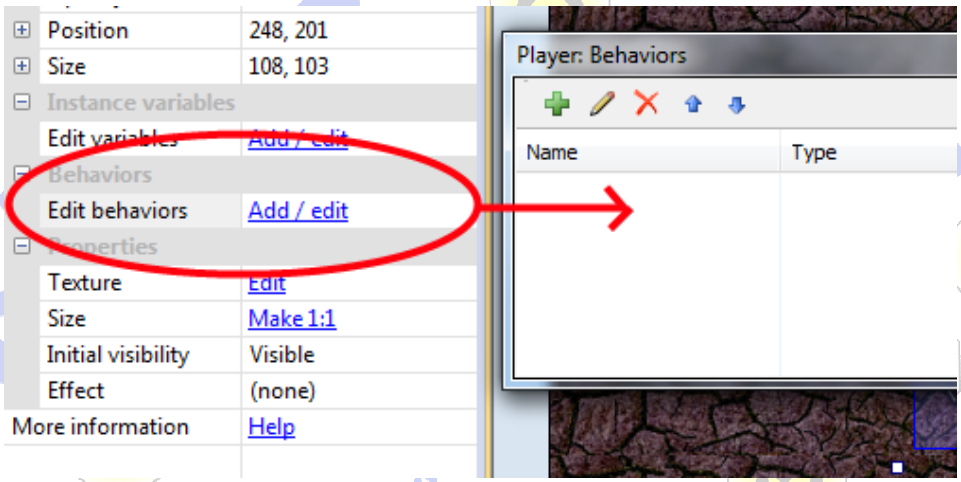
Behaviors are pre-packaged functionality in Construct 2. For example, you can add a *Platform* behavior to an object, and the *Solid* behavior to the floor, and you instantly can jump around like a platformer. You can do the same in events, but it takes longer, and there's no point anyway if the behavior 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.

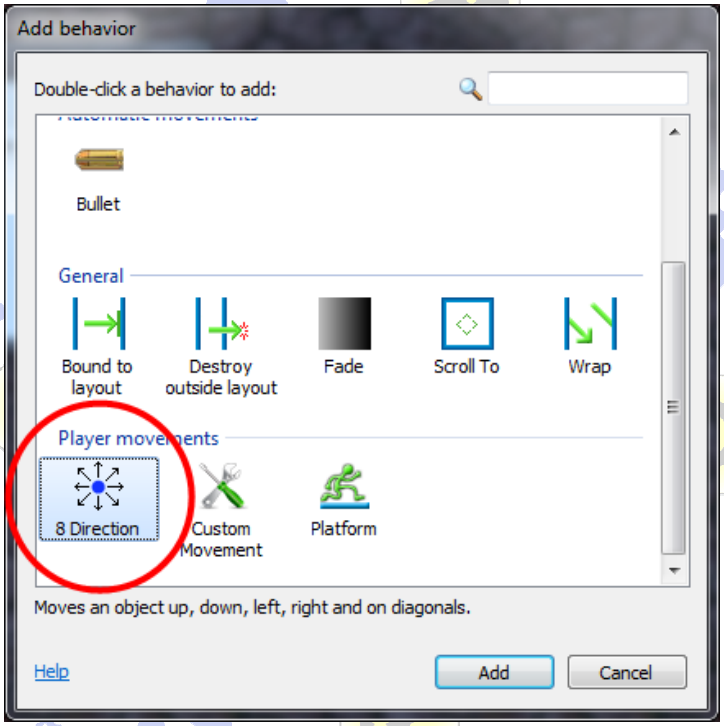
Let's add these behaviors to the objects that need them.

How to add a behavior

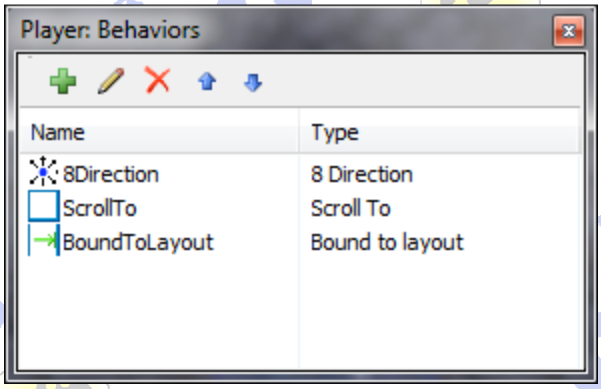
Let's add the **8 direction movement** behavior to the player. Click the player to select it. In the properties bar, notice the *Behaviors* category. Click *Add / Edit*there. The Behaviors dialog for the player will open.



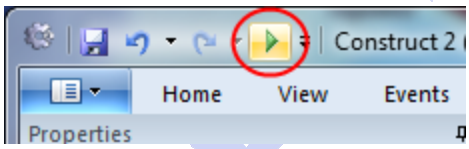
Click the green 'add behavior' icon in the behaviors dialog. Double-click the **8 direction movement** to add it.



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

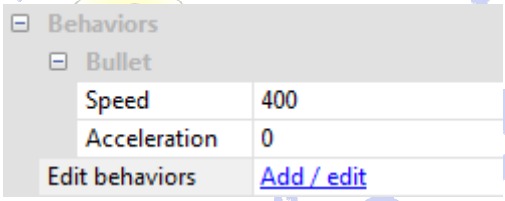
We can add behaviors to the other objects by the same method - select it, click *Add / Edit* to open the behaviors dialog, and add some behaviors. Let's add those other behaviors:

- 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).

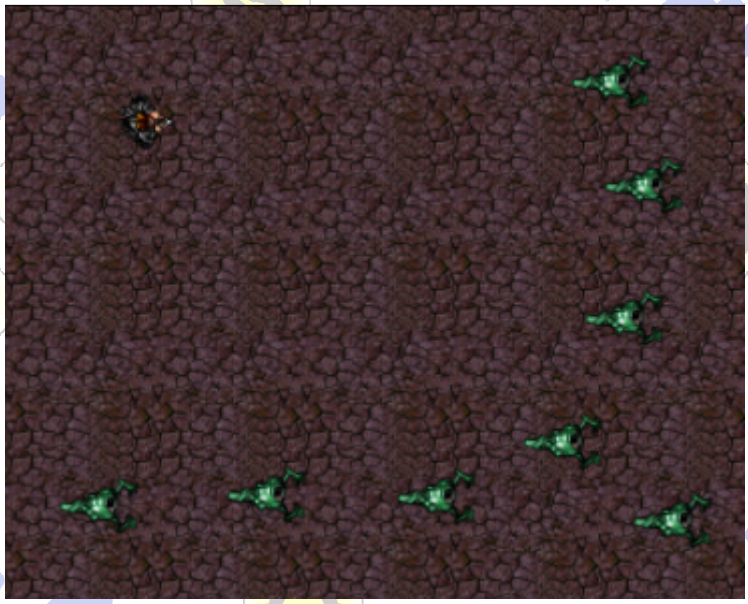
Similarly, change the **Bullet object's** speed to 600, and the **Explosion** object's Fade behavior's *Fade out time* to **0.5** (that's half a second).

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.

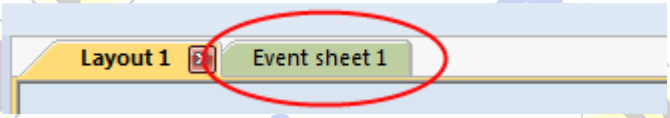
Using control + drag, **create 7 or 8 new monsters**. Don't place any too close to the player, or they might die straight away! You can zoom out with control + mouse wheel down if it helps, and spread them over the whole layout. You should end up with something a bit like this.



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.

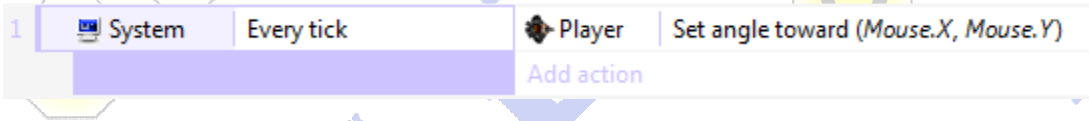
Let's go over that again. In short, an event basically runs like this:

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.

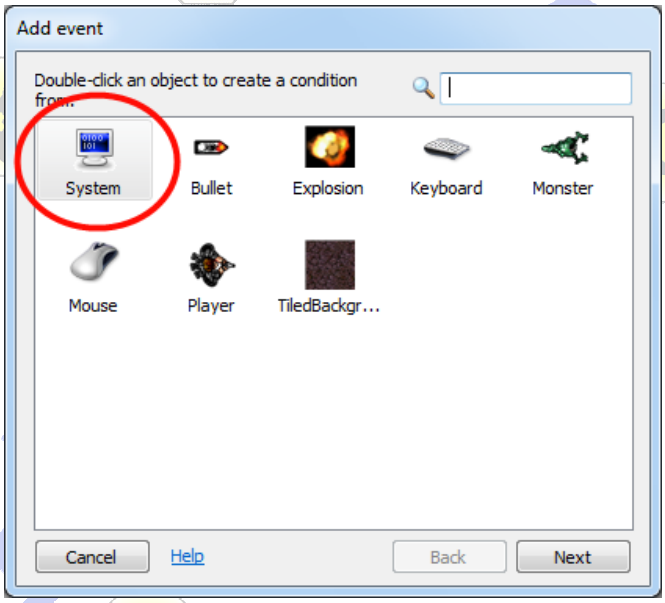
Your first event

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

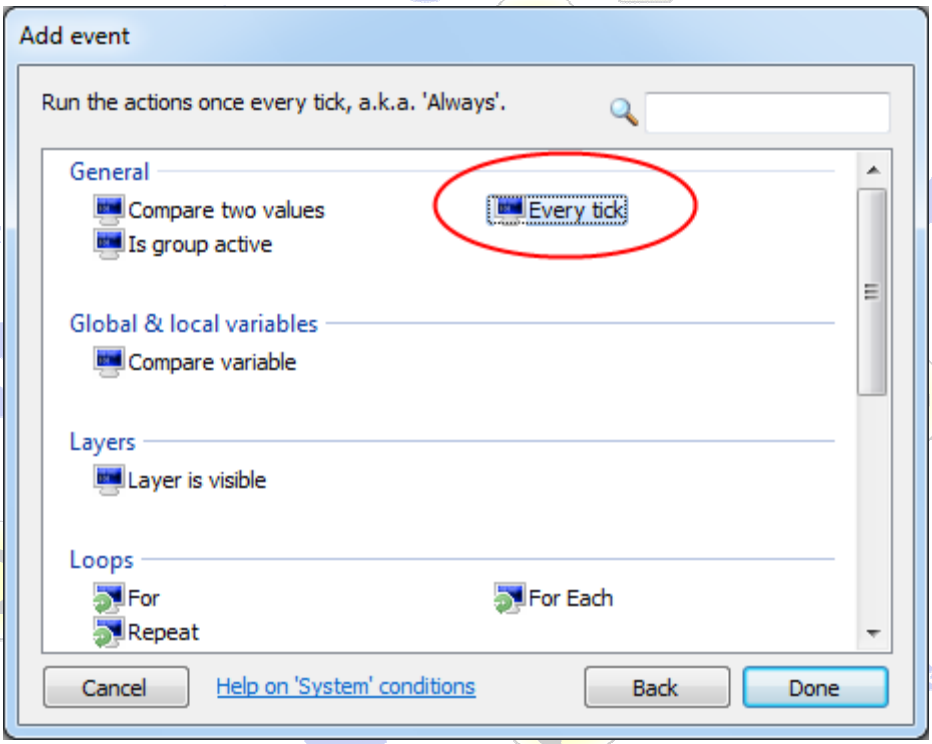


Remember a tick runs every time the screen is drawn, so if we make the player face the mouse every tick, they'll always appear to be facing the mouse.

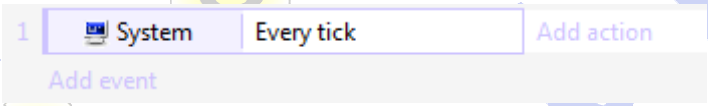
Let's start making this event. Double-click a space in the event sheet. This will prompt us to add a **condition** for the new event.



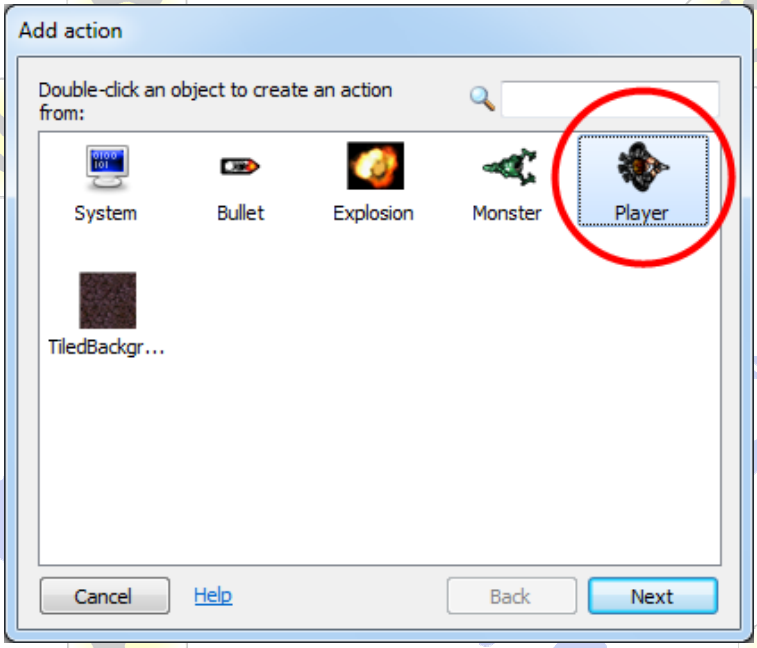
Different objects have different conditions and actions depending on what they can do. There's also the **System object**, which represents Construct 2's built-in functionality. **Double-click** the System object as shown. The dialog will then list all of the System object's conditions:



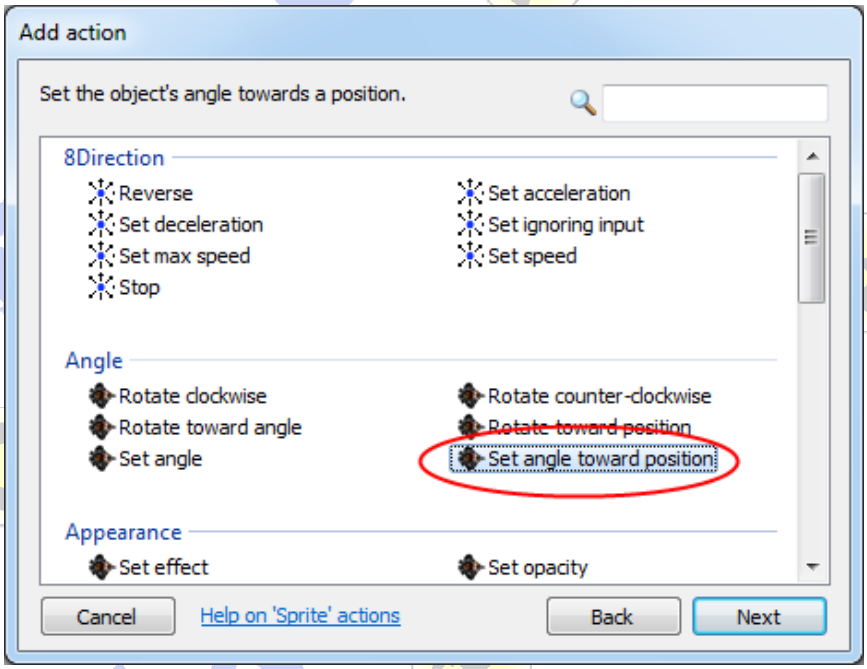
Double-click the *Every tick* condition to insert it. The dialog will close and the event is created, with no actions. It should now look like this:



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:



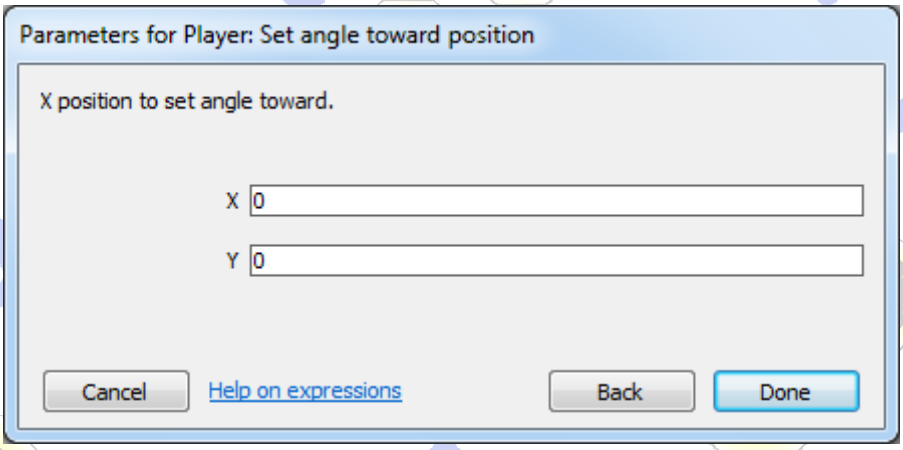
As with adding an event, we have our same list of objects to choose from, but this time for adding an *action*. Try not to get confused between adding conditions and adding actions! As shown, **double-click** the *Player* object, for it is the player we want to look at the mouse. The list of actions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

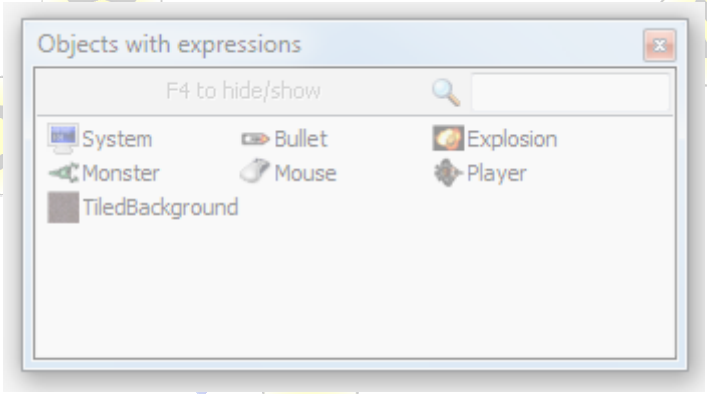


These are called the **parameters** of the action. Conditions can have parameters too, but *Every tick* doesn't need any.

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.

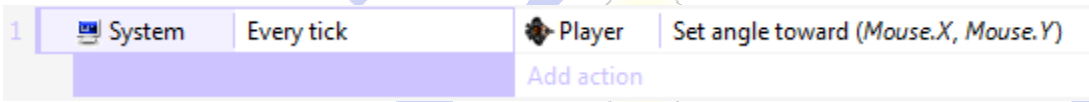
Did you get an error that said "Mouse is not an object name"? Make sure you added the Mouse object! Go back to page 2 and check under "Add the input objects".

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.