DIGITAL SCHOOLHOUSE together with NINTENDO SWITCH

CONSTRUCT

Step by Step Guide.

Bernie Crumbs

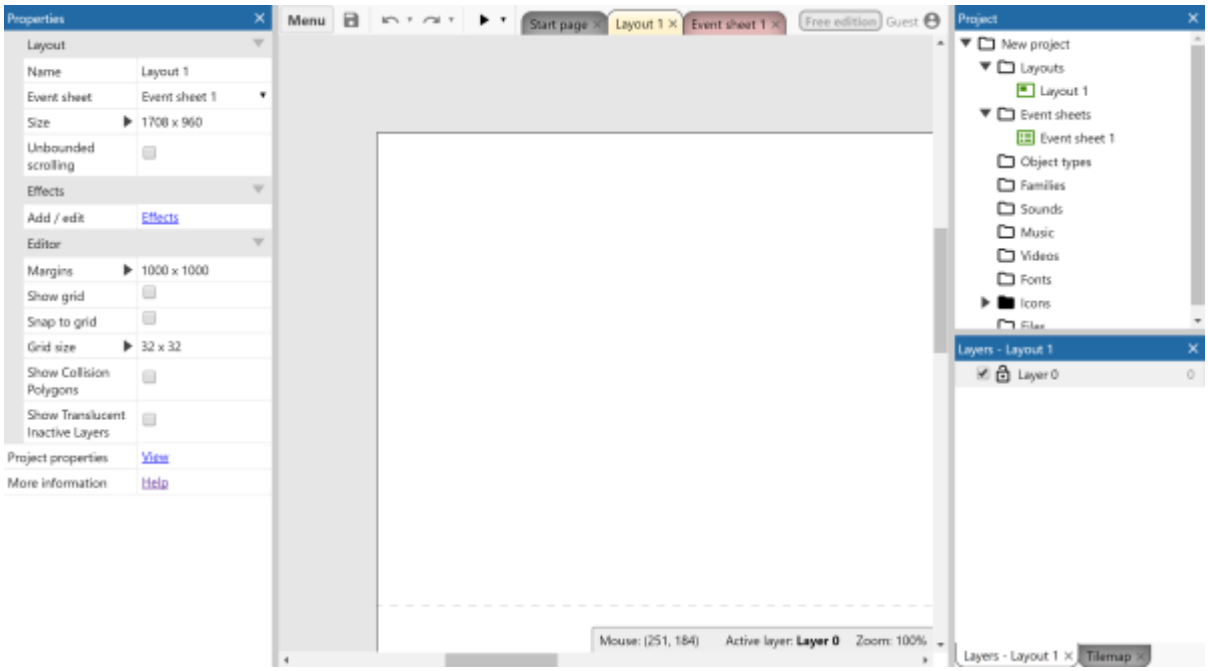
Setting up Construct 3

Construct only works in Google Chrome - if you are using Internet Explorer you will need to change browser.

Construct 3 runs right in your browser! There is nothing to install or set up. If you see an error, check the system requirements page, you might need to update your browser or system.

Create a new project

Click the New project button. A dialog will appear asking for some details. You don't have to change anything, but you can type in a name for your project if you like (how about *My super awesome game*?). Click Create and you should see a new empty project something like this.



A new empty project in Construct 3

Note about screenshots: we're using the default theme in Construct 3 for images. If you change the theme, or Construct 3 looks a little different, don't worry - it should still be straightforward to follow along.

The main view in the middle of the screen is the *layout view*. This is 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*

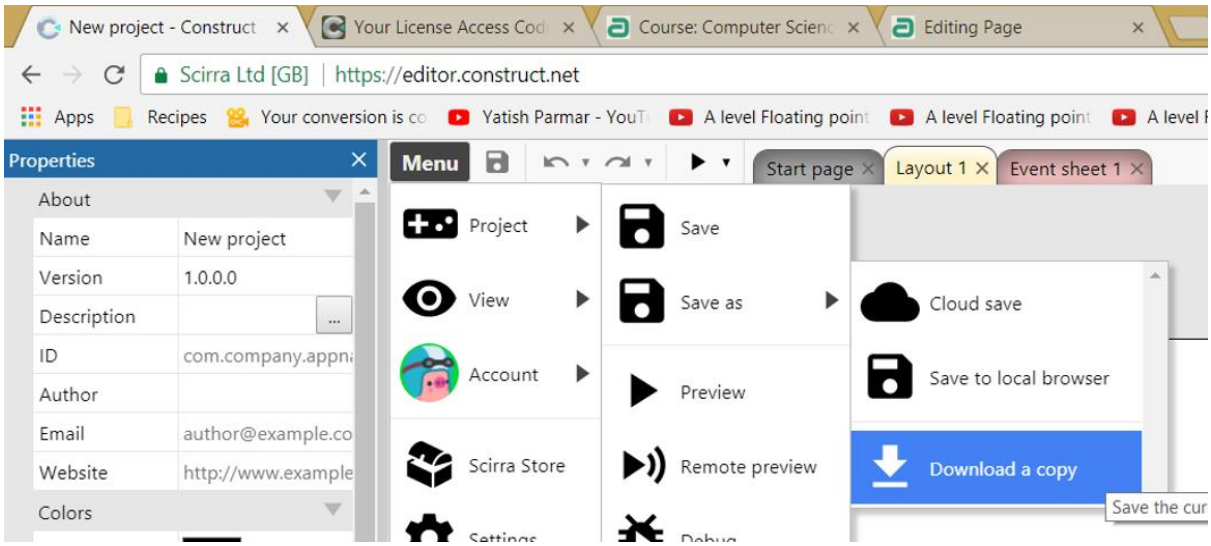
How to save your work

NOTE: Download a Copy will automatically save to your Downloads folder - you must move this into your computer science folder at the end of each lesson otherwise you will lose your work. I have explained how to do this below.

Probably the most important thing to remember when using Construct 3 is to save your work regularly!

Two ways to save – download a Copy:

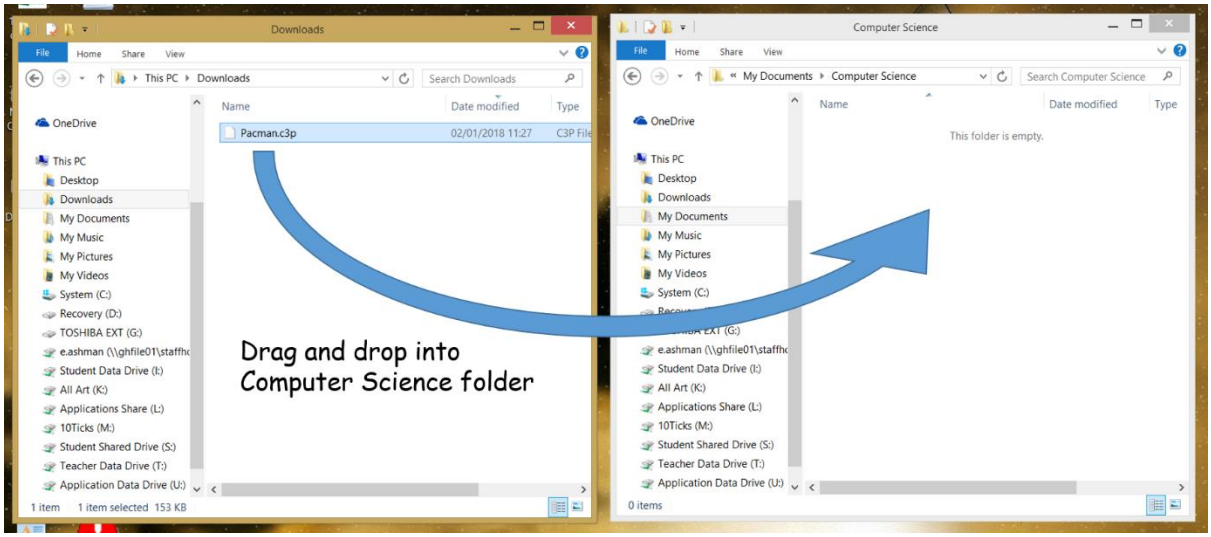
It is easy to save - click on Menu > Project > Save As > Download a Copy



Download a copy will save into your Download folder - this is cleared out each evening - to save permanently you must move your work into your Computer Science Folder.

To do this minimise your browser window by pressing the minus (-) sign on the top right of the page.

Open My Documents and go to your Computer Science folder. Open another My Documents window and go to your Downloads folder. Drag and drop your game file into your Computer Science folder:



Make sure you remember to move your saves to your own folder at the end of each lesson!

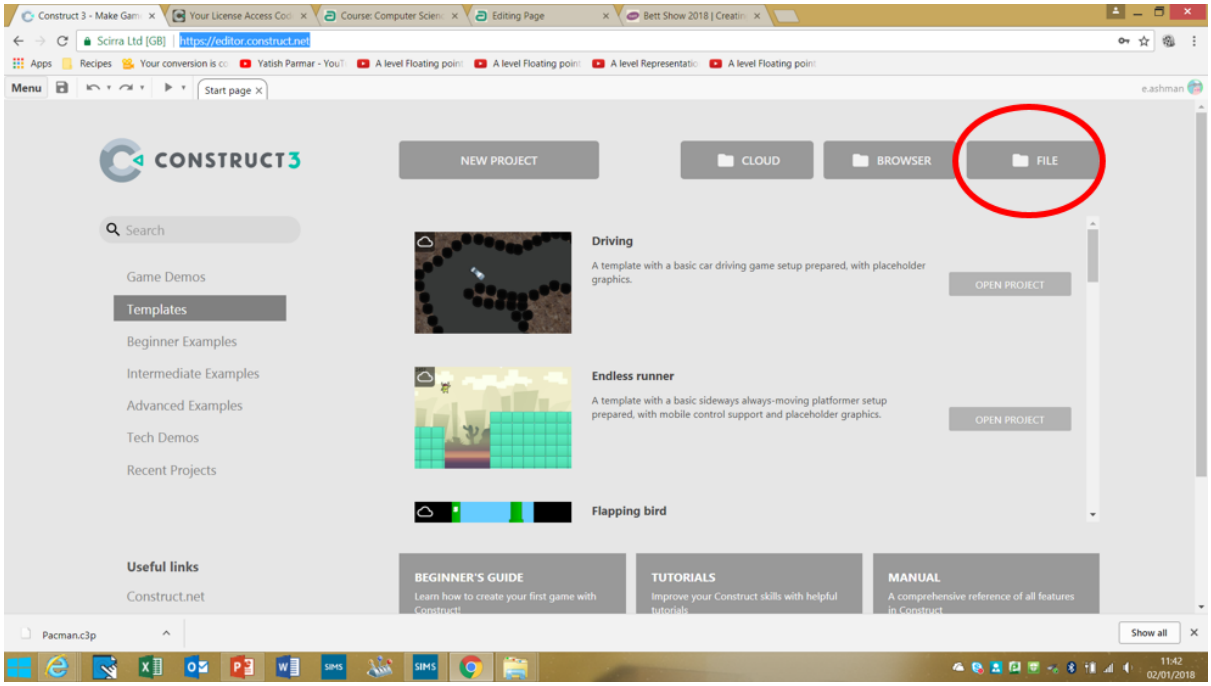
Or click on Cloud save and sign into your OneDrive or DropBox account to save to these locations.

How to load your work from a previous lesson

Launch Construct3 go to:

https://editor.construct.net/

Click on File:



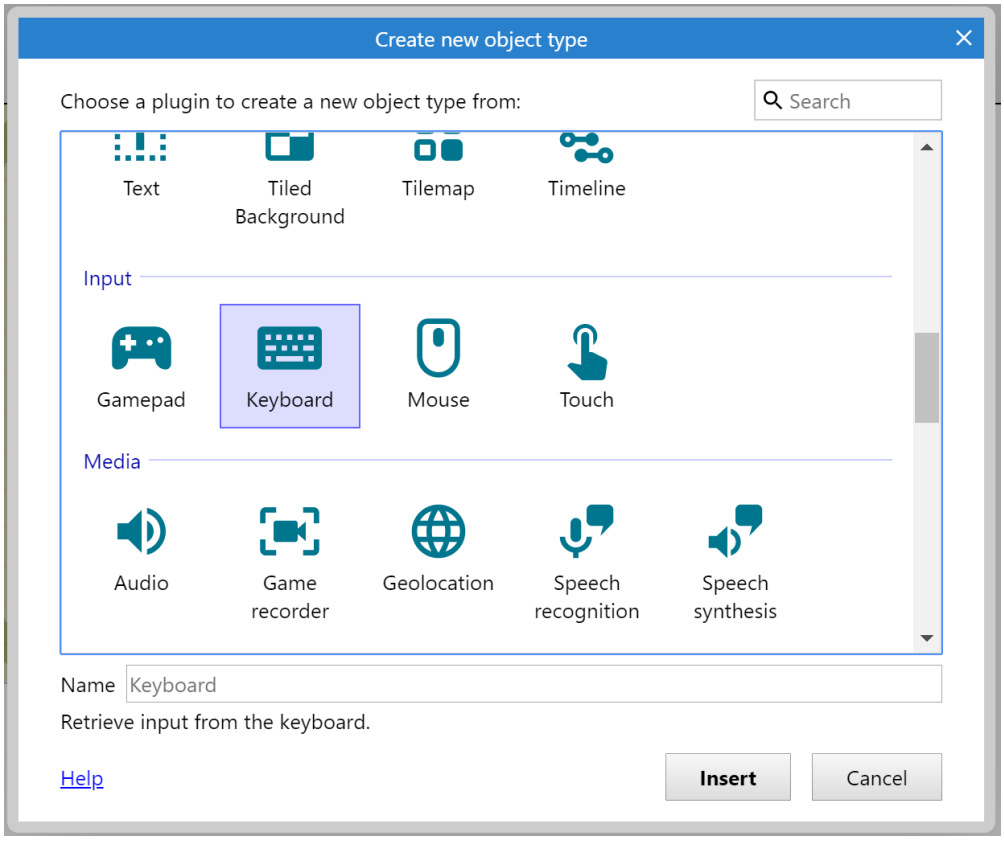
A browse to file window will pop up - go to your Computer Science folder and open the most recent version of your game file (look at the date modified).

If you have used Cloud save click on cloud, sign in and browse to the location you saved your game into.

That's it - continue with your game. Remember to save your work regularly and don't forget to move your work from downloads into your Computer Science folder at the end of the lesson.

Add the input objects

Double-click in a space (this can be anywhere since the background is locked) to add a new object. Select the Keyboard object, since we'll need keyboard input:

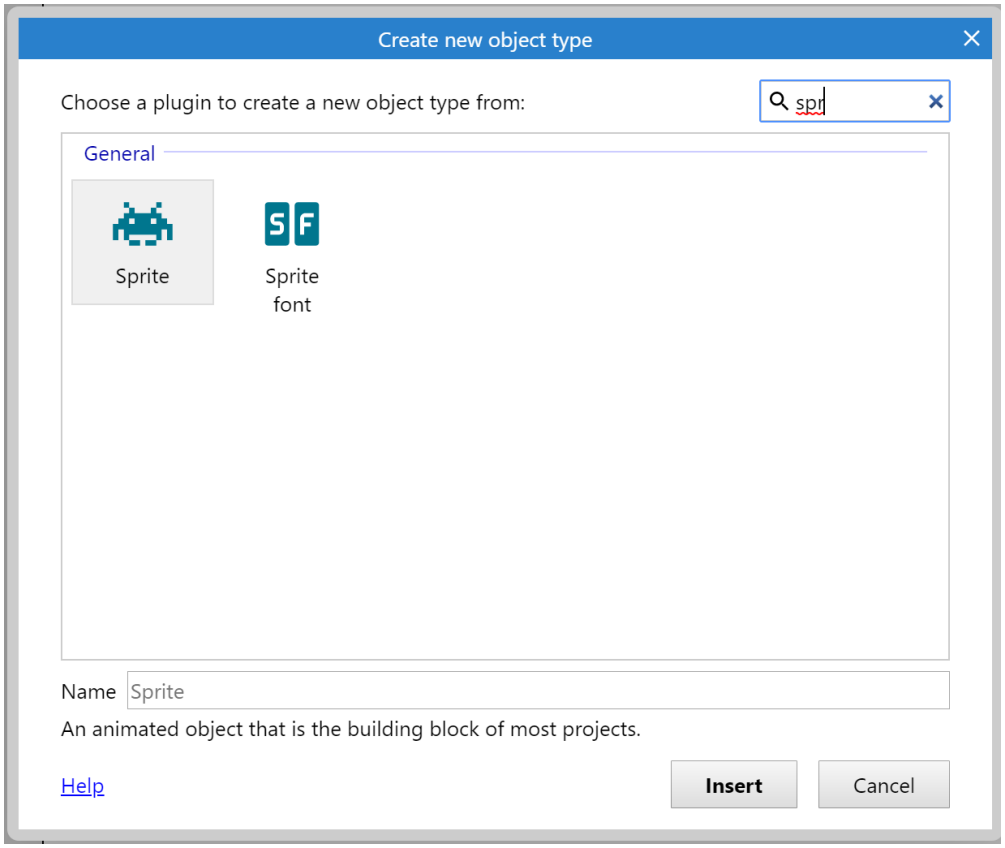


Note that this object doesn't need placing in the layout. It is hidden, and automatically works across the entire project. Now all layouts in our project can accept keyboard input

The game objects

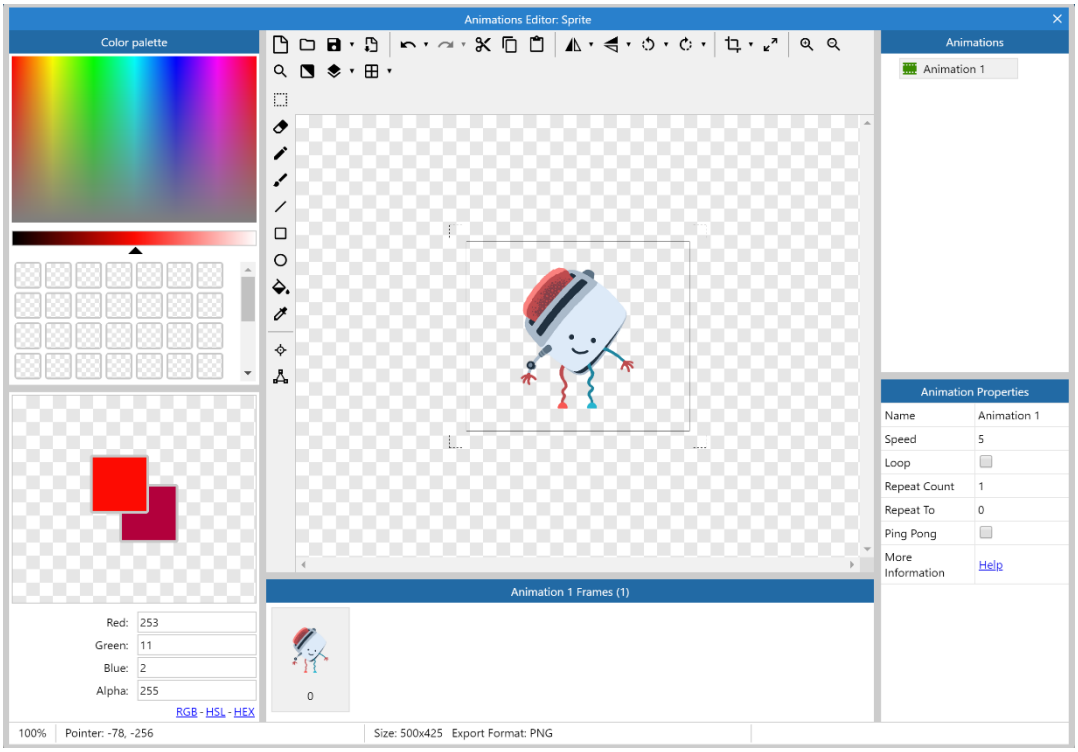
It's time to add our game objects! First we are going to add the idle animation for our Bernie Crumbs sprite.

To do this we will add a Sprite object.

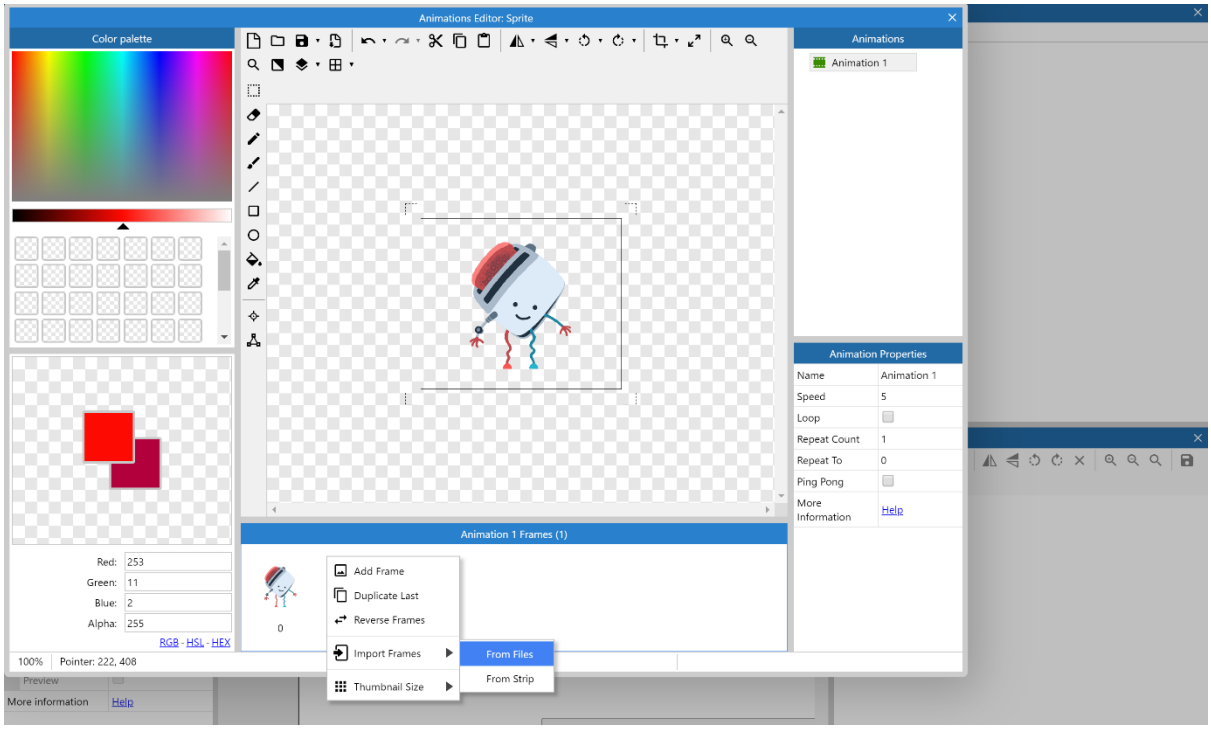


Sprites simply display an image, which you can move about, rotate, resize and optionally animate. Games are generally composed mostly out of sprite objects.

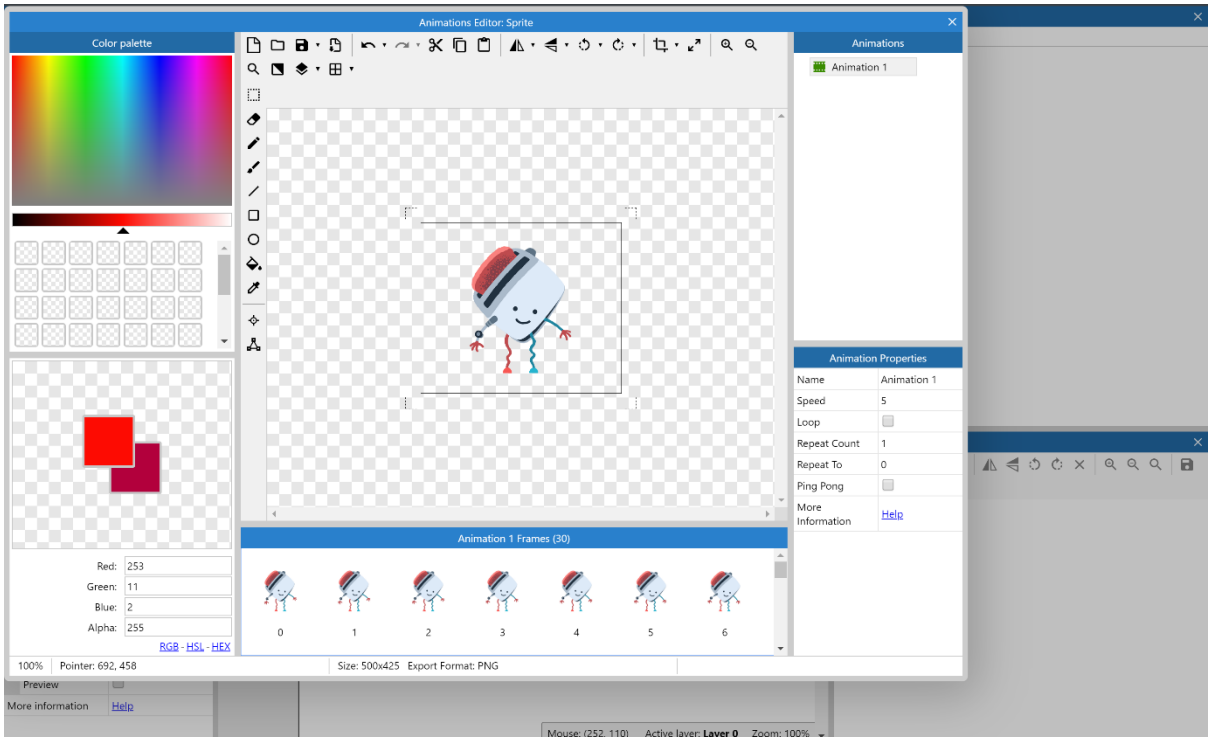
When the mouse turns to a crosshair, click somewhere in the layout to place Bernie Crumbs. The image editor pops up. Click the *Load image* button (folder icon), and select the first of the Toaster idle images:



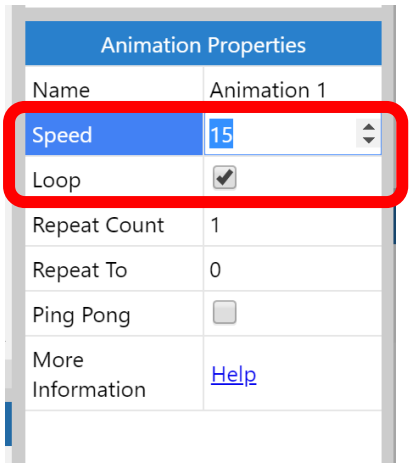
Next we are going to animate the sprite, to do this we need to add more frames. Right click next to frame 0 in the Animation pane at the bottom of the editor and click on Import Frames > From Files



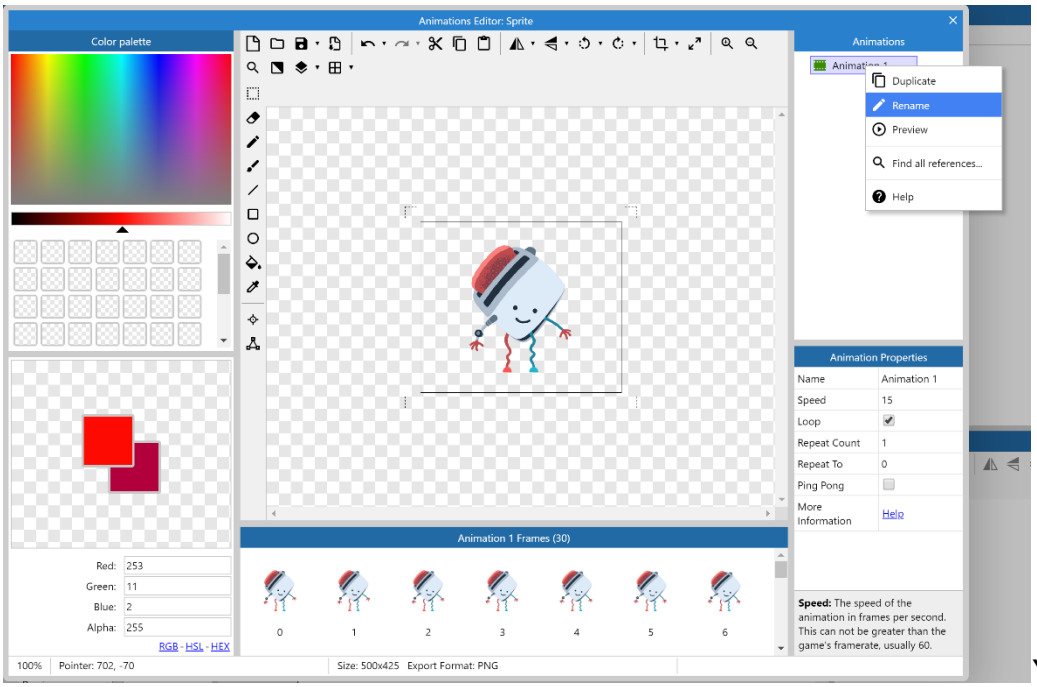
Select the rest of the idle animation frames from the Bernie Crumbs assets pack you will find on your USB stick, you can select a series of images by selecting the first image and then selecting the last image while holding the shift key:



Finally, we will change some of the animation properties as shown:

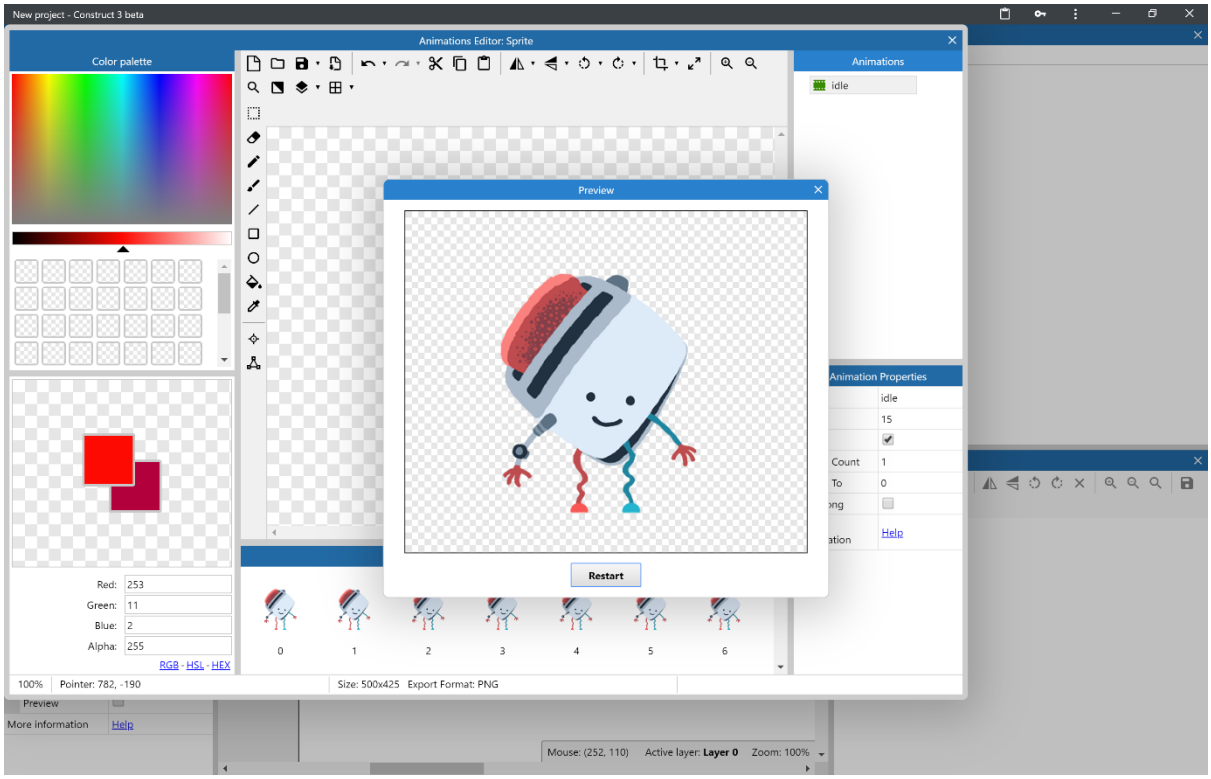


Right click on the name of the animation and rename it to idle:

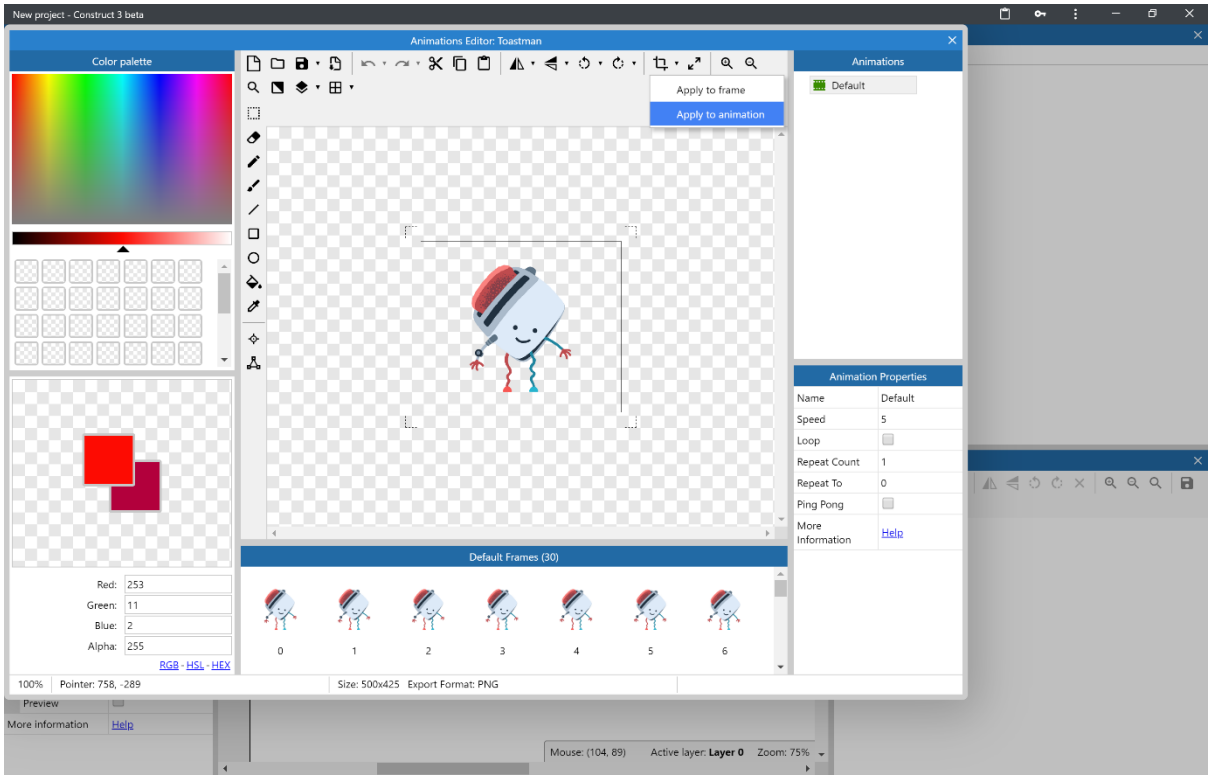


You can

Then right click again and click on preview to check that the animation is playing at the correct speed (frame rate):

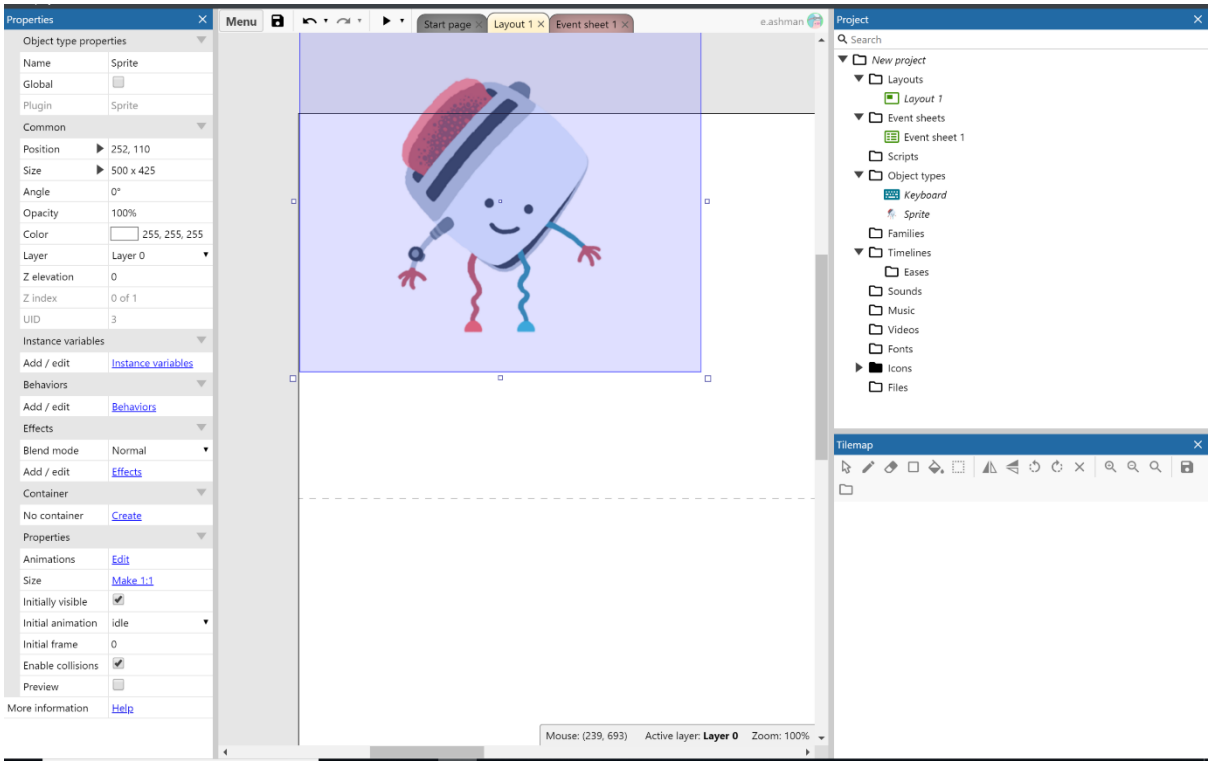


It's a good idea to remove any excess space around your sprite as this will help with collisions later, to do this click on the arrow next to the crop tool and then select apply to animation:



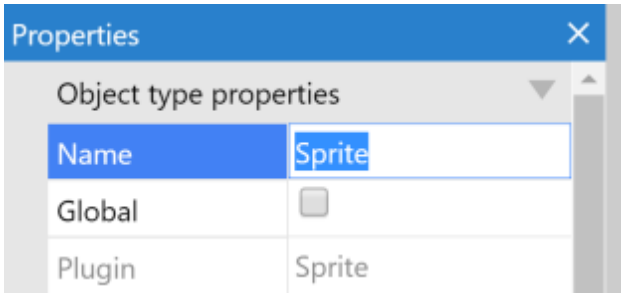
Close the image editor by clicking the X in the top right-hand corner. You should now see the object in the layout!

The image is a little too big! Use the bounding box that appears when the object is selected to resize your image:



Another quick way to create sprite objects is to drag and drop an image file into the layout view. Construct will create a Sprite with that image for you. If you drag multiple files in at once, Construct will make a single sprite with the rest of the files as animation frames.

Your Bernie Crumbs sprite will be called *Sprite* as default. That's not very useful - things will quickly get confusing like this. Rename your sprite to Bernie Crumbs. You can do it by selecting the object, then changing the Name property in the properties bar:

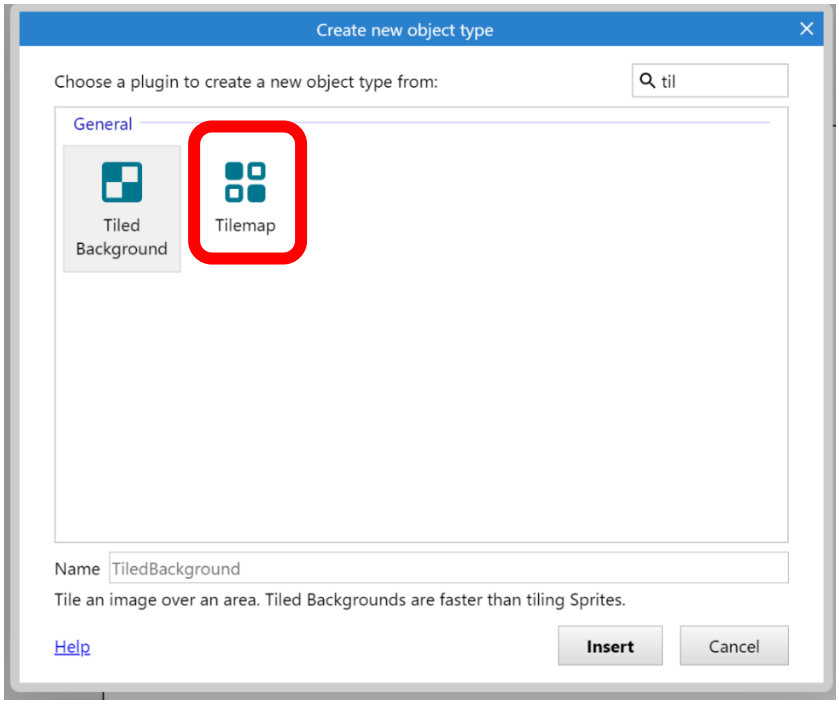


(If you're impatient like me, click the Previewbutton in the main toolbar - the preview window should pop up showing your animated Bernie Crumbs idle animation! Woo!)

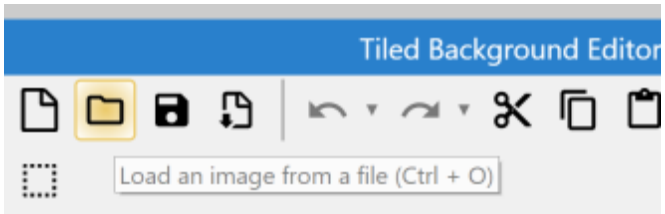
Tilemap

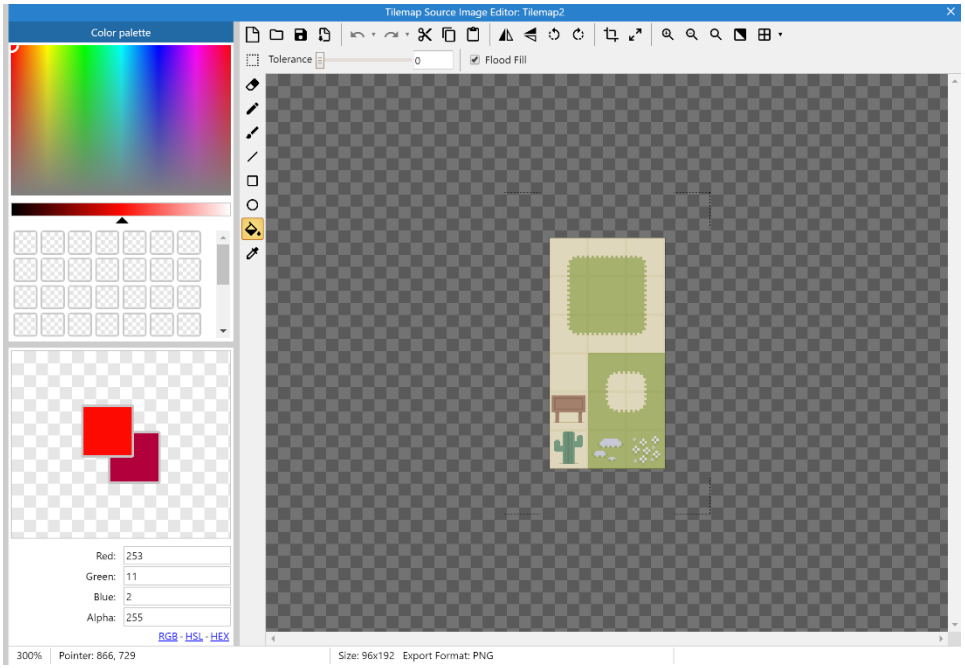
For this game we are going to use a tilemap for the background.

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

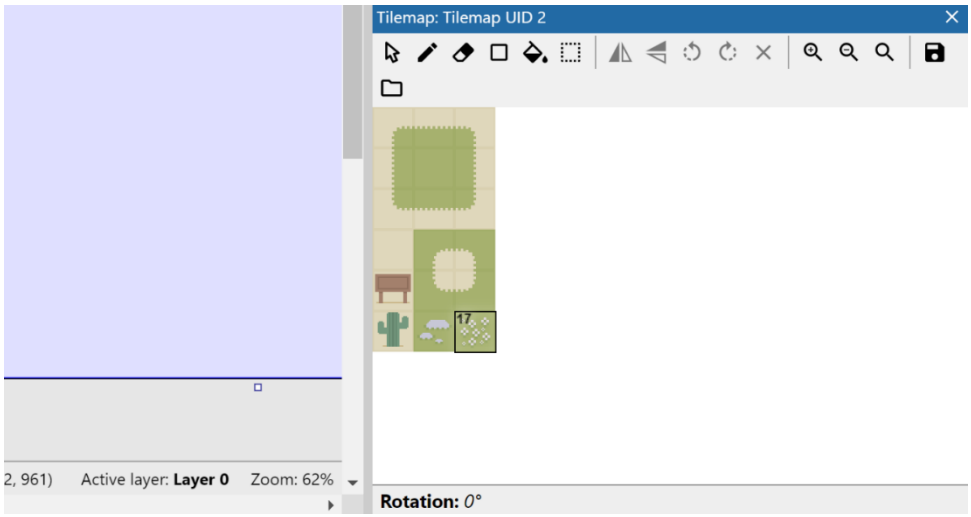


The mouse will turn in to a crosshair for you to indicate where to place the object. Click somewhere near the middle of the layout. The image editor now opens for you to import the Tilemap you want to use. We are going to use Construct3's default Tilemap that is included as standard (if you wanted to import a new Tilemap click on the Folder icon to import a new Tilemap as shown below).

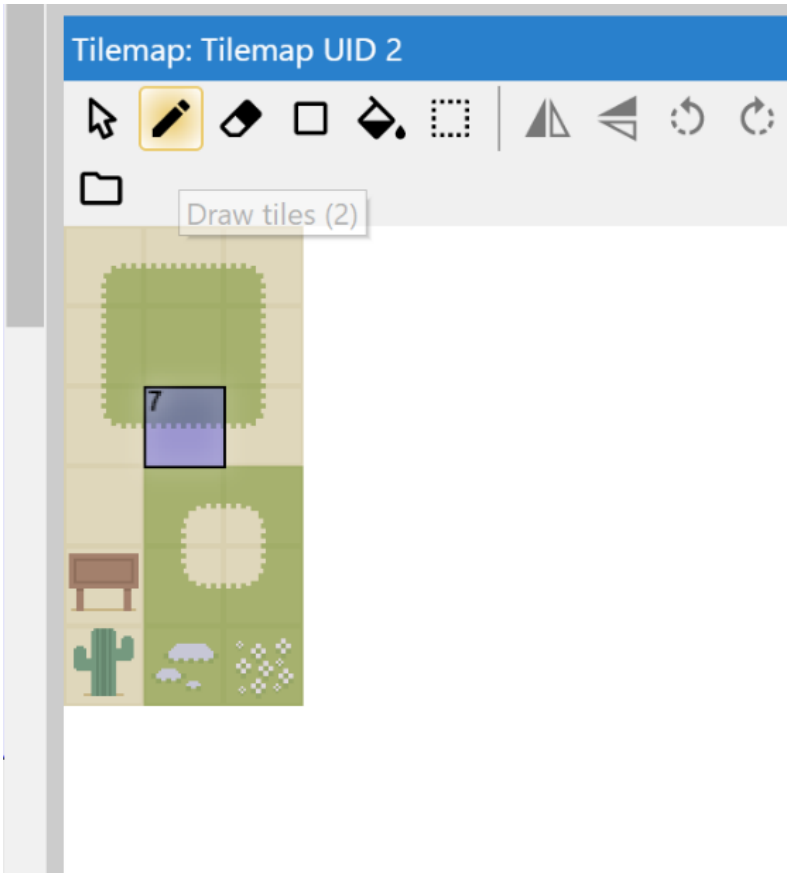




Once you have added your tilemap, close the image editor by clicking the X in the top-right. Now you should see your Tilemap on the bottom right side of the screen:



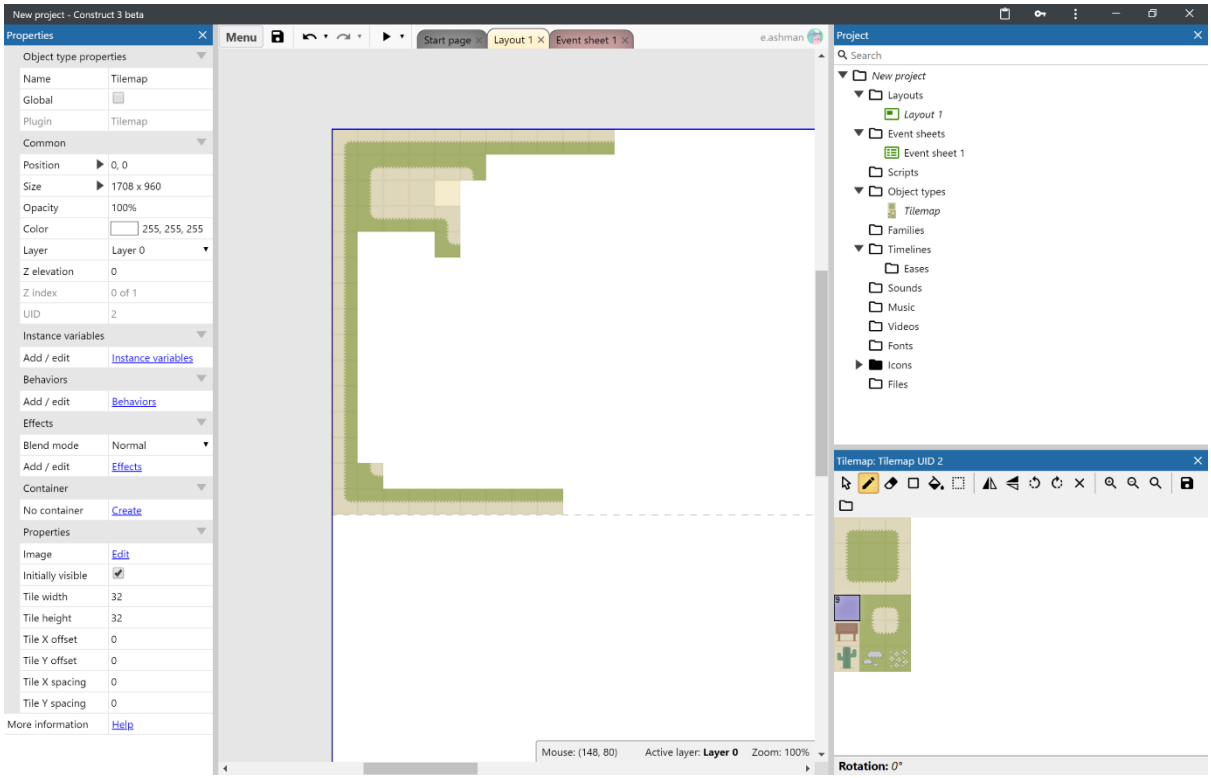
To draw tiles, select the tile you want to use and then select the draw tool:



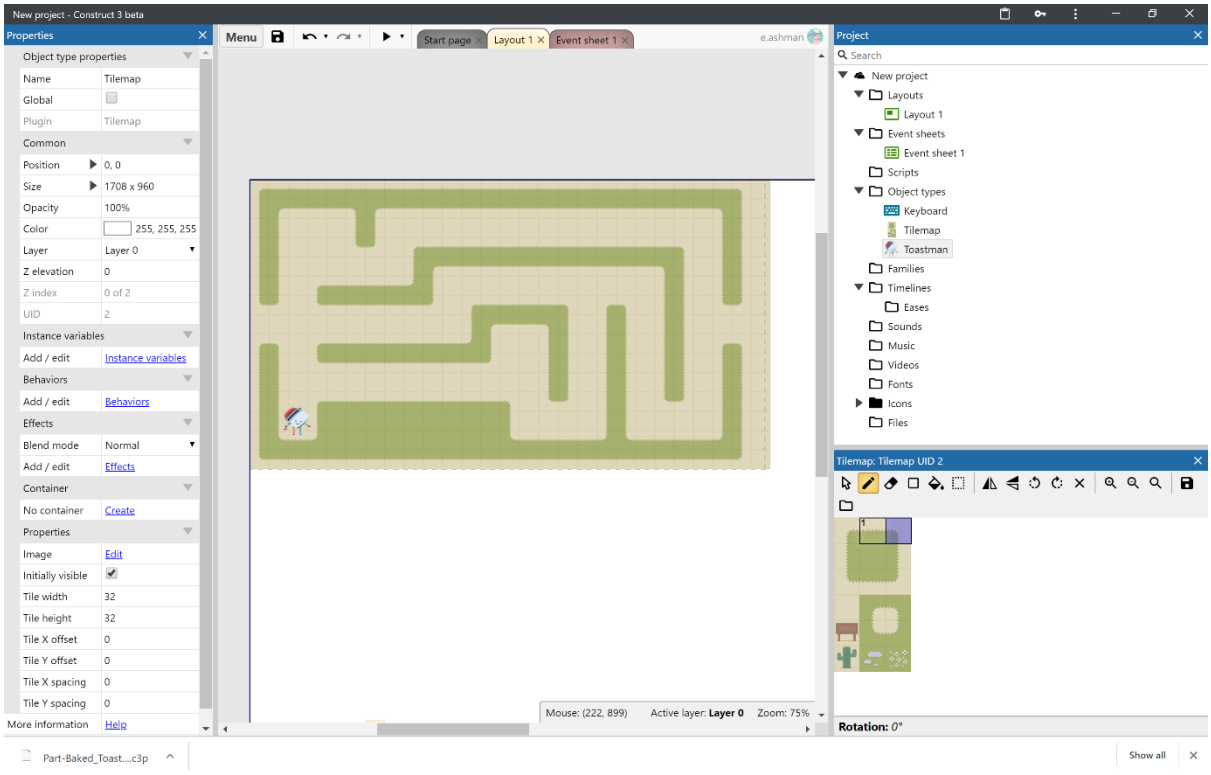
Let's zoom out. Hold Control and scroll the mouse wheel down to zoom out. Alternatively, right-click and select View▶Zoom out a couple of times. You can also hold Space bar, or the middle mouse button, to pan around.

The dotted line you can see is called the viewport, this is the area that will be visible when you play the game.

Click in the viewport to place the tiles where you want them. To change to a different tile, simply select a new tile from the tilemap in the bottom right hand corner.

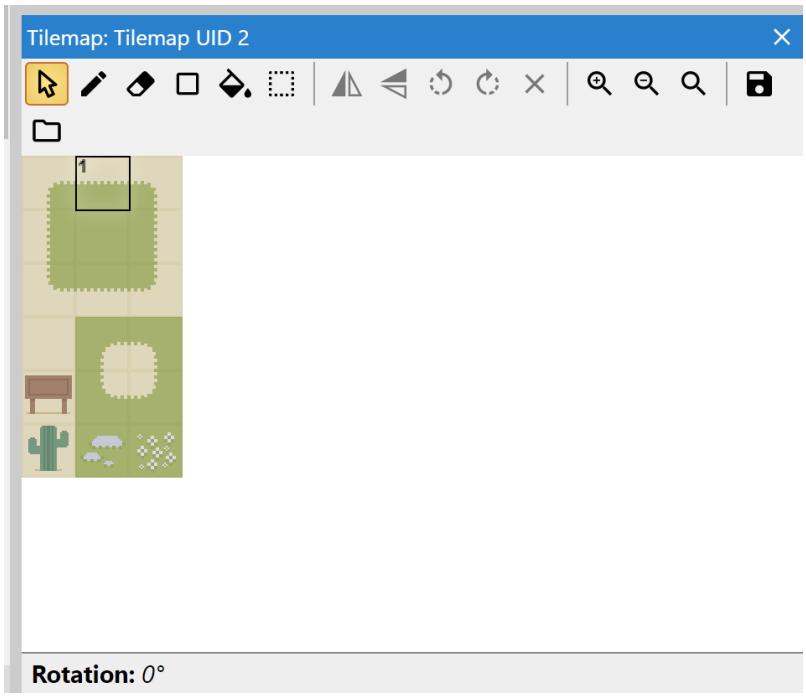


Zoomed out view of completed maze in layout



Hit Ctrl + 0 or right-click and select View▶Reset zoom to return to 1:1 view.

To stop editing the tilemap, click on the arrow to return to normal layout view, this will allow you to add and edit more objects:



Before we continue, the tilemap background should be locked. As we create and move around objects on top of it, it's easy to accidentally select or modify the background. Since we don't need to change the background anymore, locking it makes it unselectable, so it won't get in the way. To lock it, right-click on the tiled background and select Lock▶Lock selection. (If you do want to change it later, simply right-click and select Lock▶Unlock all.)

Behaviours

Behaviors are quick ways to make an object act a certain way. 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 game. You can usually do the same in events, but behaviors are much quicker! Construct has a wide range of behaviors, but here are a few that we'll use in this game.

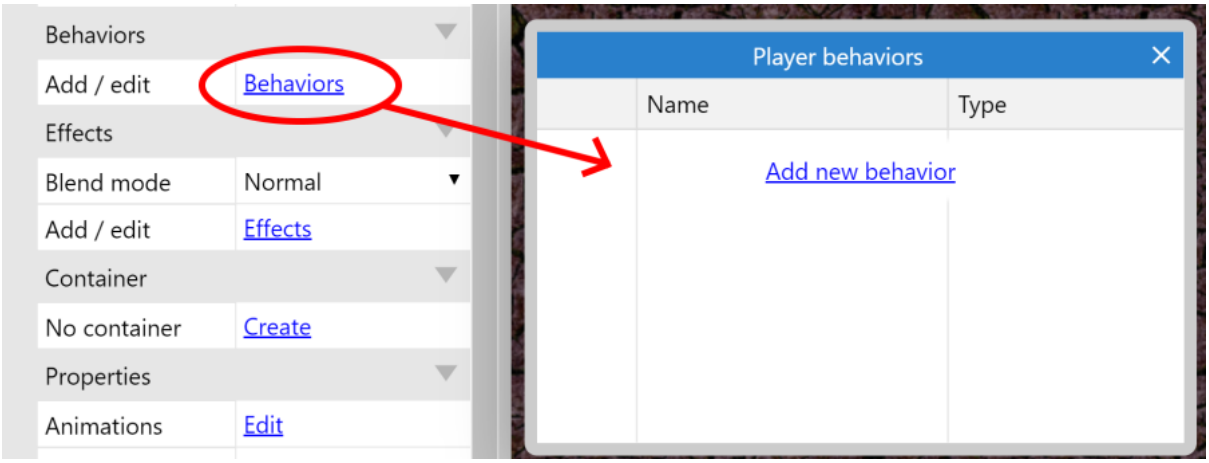
- 8 Direction movement: this lets you move an object around with the arrow keys. It will do nicely for Bernie Crumbs's movement.
- Bullet movement: this simply moves an object forwards at its current angle. Despite the name, it will work nicely to move the enemies around - since all the movement does is move objects forwards at some speed.
- Wrap: this makes an object appear on the other side of the screen when they leave the on opposite side. This will be useful to make Bernie Crumbs able to 'run' through the centre of the screen and appear at the otherside.

- Fade: this makes an object fade out, which we will use on toast that Bernie Crumbs eats.

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

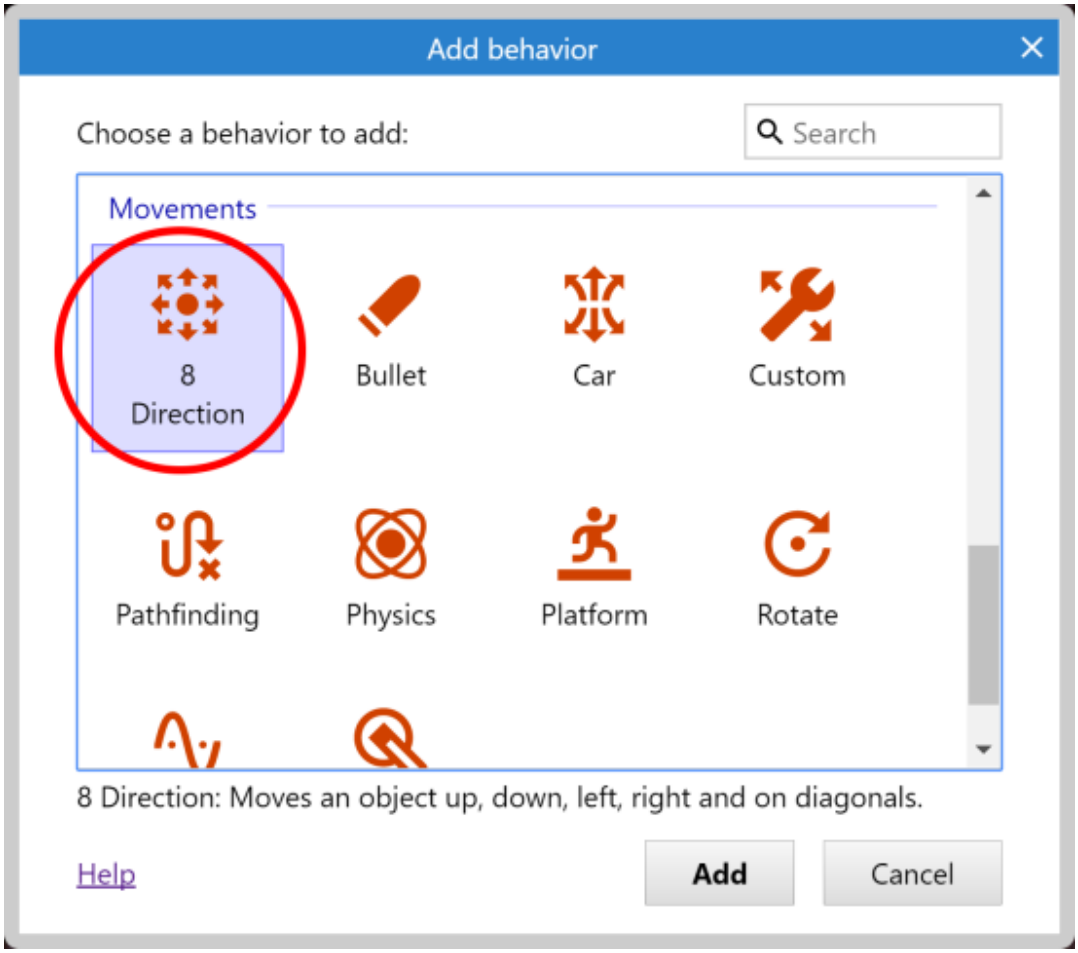
How to add a behaviour

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

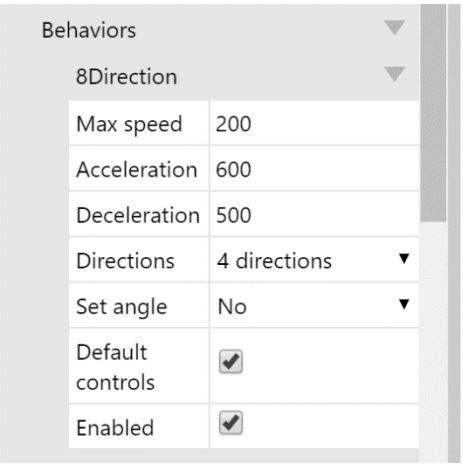


The Behaviors dialog

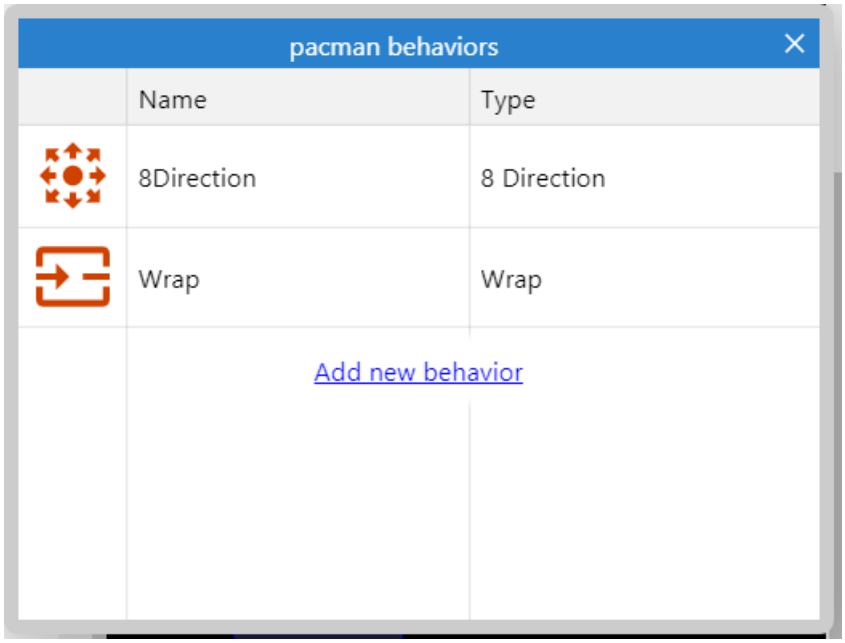
Click Add new behavior in the behaviors dialog. Double-click the 8 direction movement to add it.



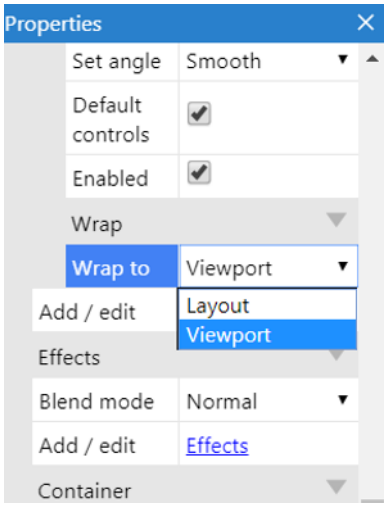
Close the dialogue box, select the Bernie Crumbs object and find the 8-direction movement properties in the Properties bar. Change the properties of Bernie Crumbs's 8-direction movement as follows:



Now add the Wrap behavior, to make Bernie Crumbs appear at the opposite side of the screen from where he leaves it. The behaviors dialog should now look like this:



Close the behaviors dialog. There is one last thing we need to do to make the wrap work correctly. Select the Bernie Crumbs object and find the Wrap properties in the Properties bar. Change it to Wrap to Viewport.

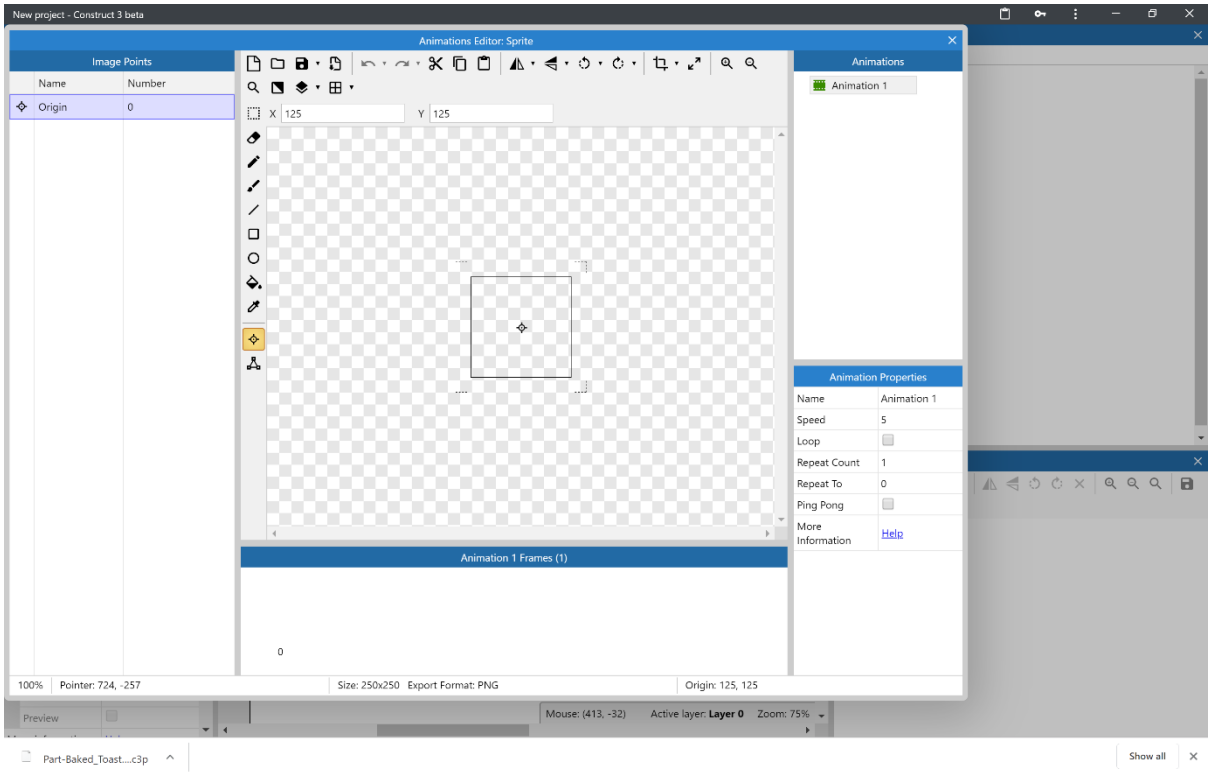


Now try pressing Preview to run the game so far! Once the preview starts, notice you can already move around with the arrow keys. You also appear on the otherside of the screen if you walk outside the layout area, thanks to the Wrap behavior. This is what behaviors are good for - quickly adding common features. We'll be using the event system soon to add custom features.

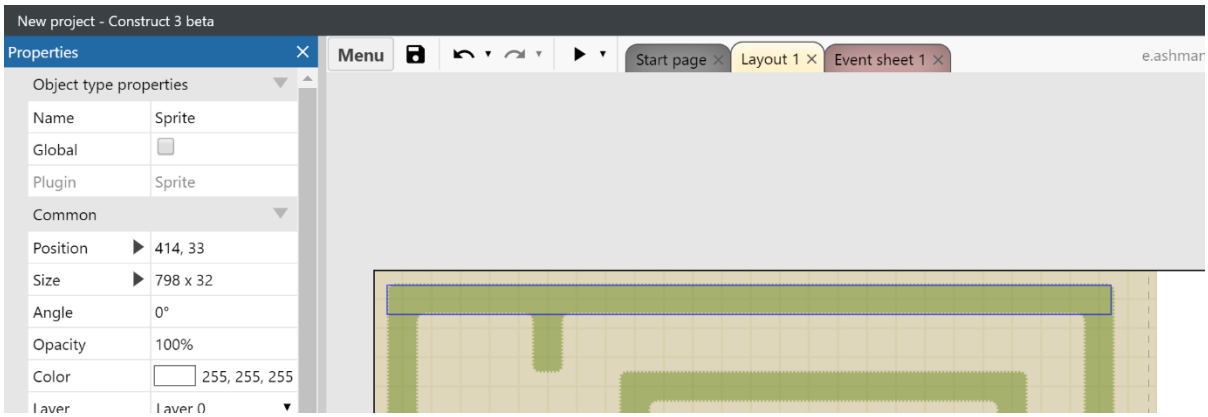
Creating a maze for Bernie Crumbs to navigate

We already have a maze in the tilemap background but currently Bernie Crumbs can walk through it. In this section we will make invisible wall sprites that Bernie Crumbs can't walk through.

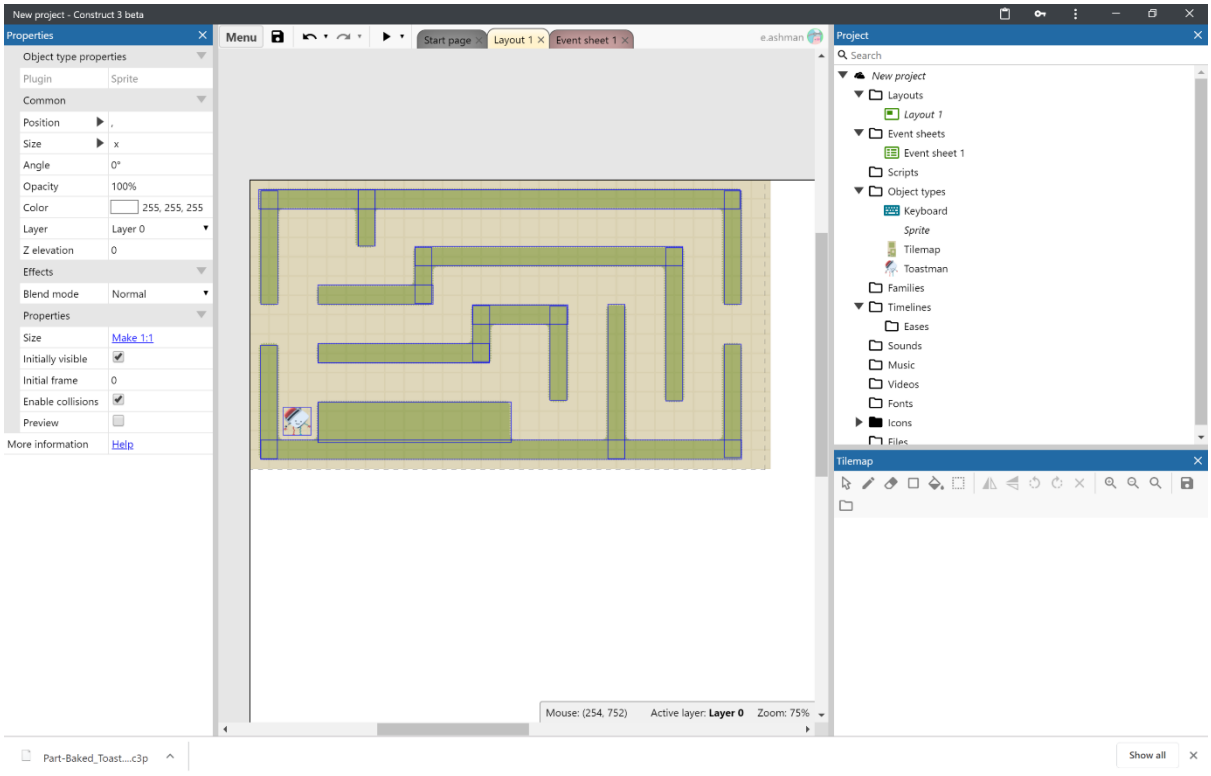
Right click and create a new sprite as you did before, when the animations editor opens leave the frame blank and close the editor:



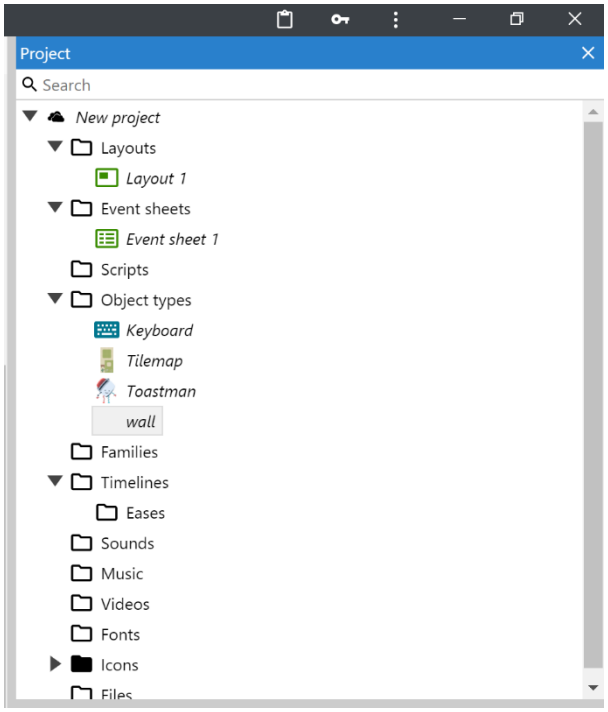
This will create an invisible sprite, resize the sprite to fit over part of the maze:



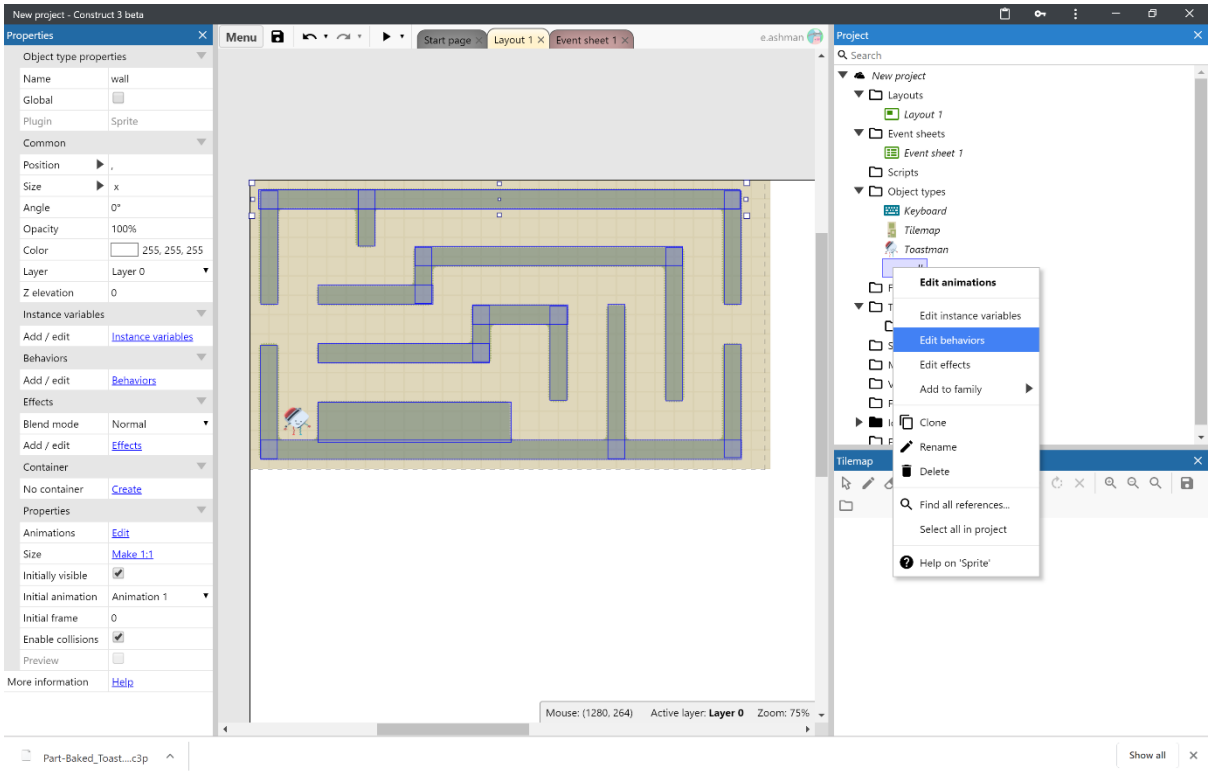
You can copy and paste the sprite in the layout to create a second instance of it (a cross hair will appear to allow you to place it), use these instances of your invisible sprite to cover all the walls of your maze:



You might like to rename the original sprite ‘wall’ in the project toolbar on the right:



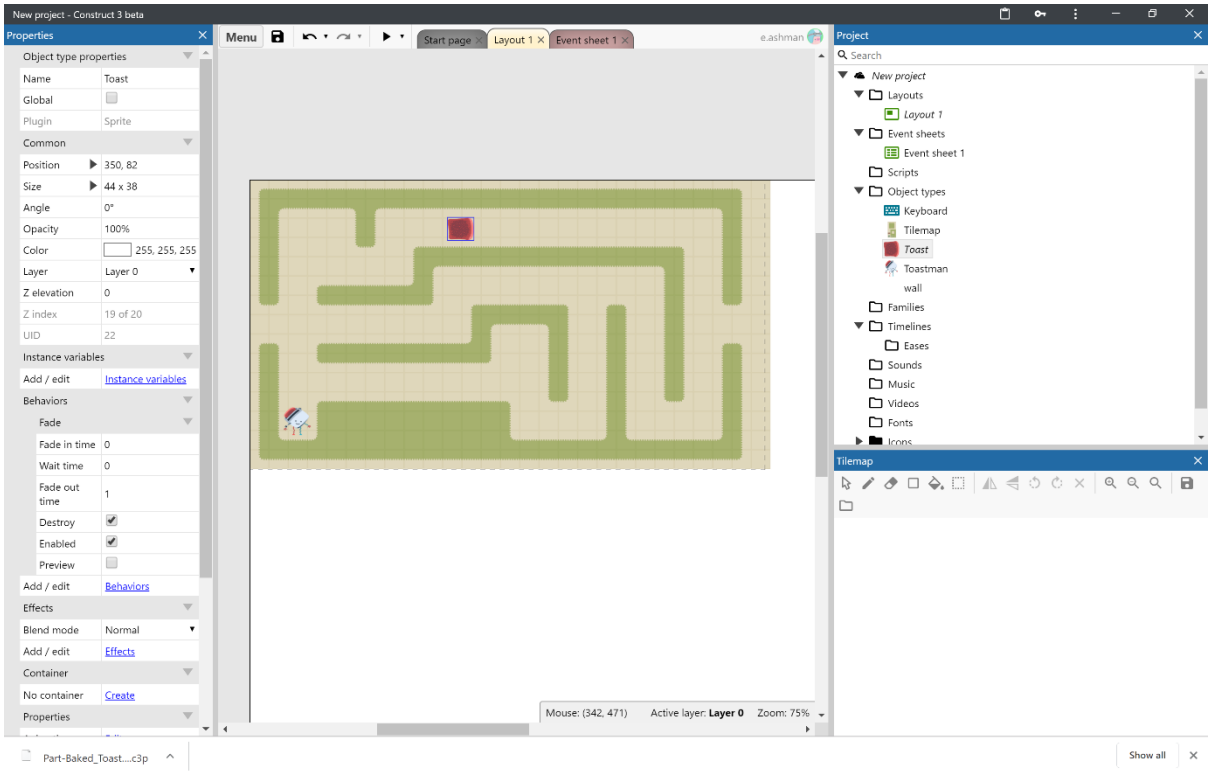
Add the Solid behaviour to each of the instances of your invisible wall by adding it to the original wall object by right clicking it in the project toolbar:



And that's it! Create yourself a maze for your Bernie Crumbs to work his way through!

Adding toast

Next add the toast sprite from the game assets folder using the same steps as you did for adding Bernie Crumbs:

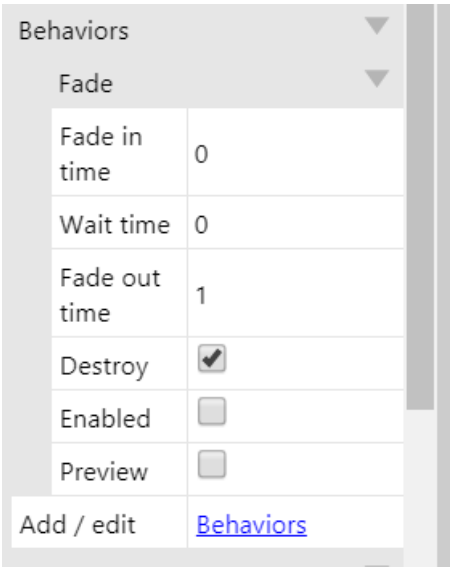


Adding the other behaviours

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

1. Add the Fade behavior to the Toast object (so it gradually disappears if eaten by Bernie Crumbs). By default the Fade behavior also destroys the object after it has faded out, which also saves us having to worry about invisible Toast objects clogging up the game.

If you run the game, you might notice that any Toast suddenly disappears. To sort that out, select your toast. Find the Fade properties in the properties bar and uncheck Enabled:

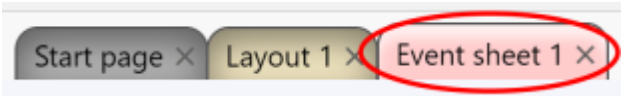


The Enabled property controls whether the fade starts straight away when the game is loaded or as a result of something else happening.

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.



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:

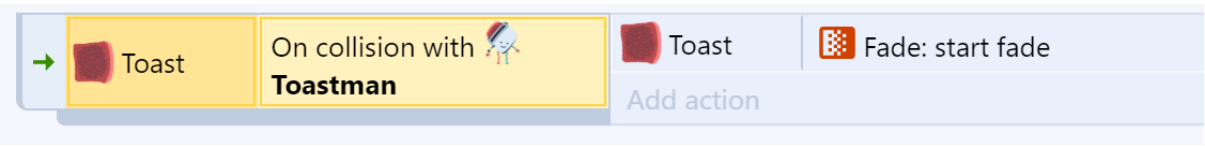
Are all conditions met?

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

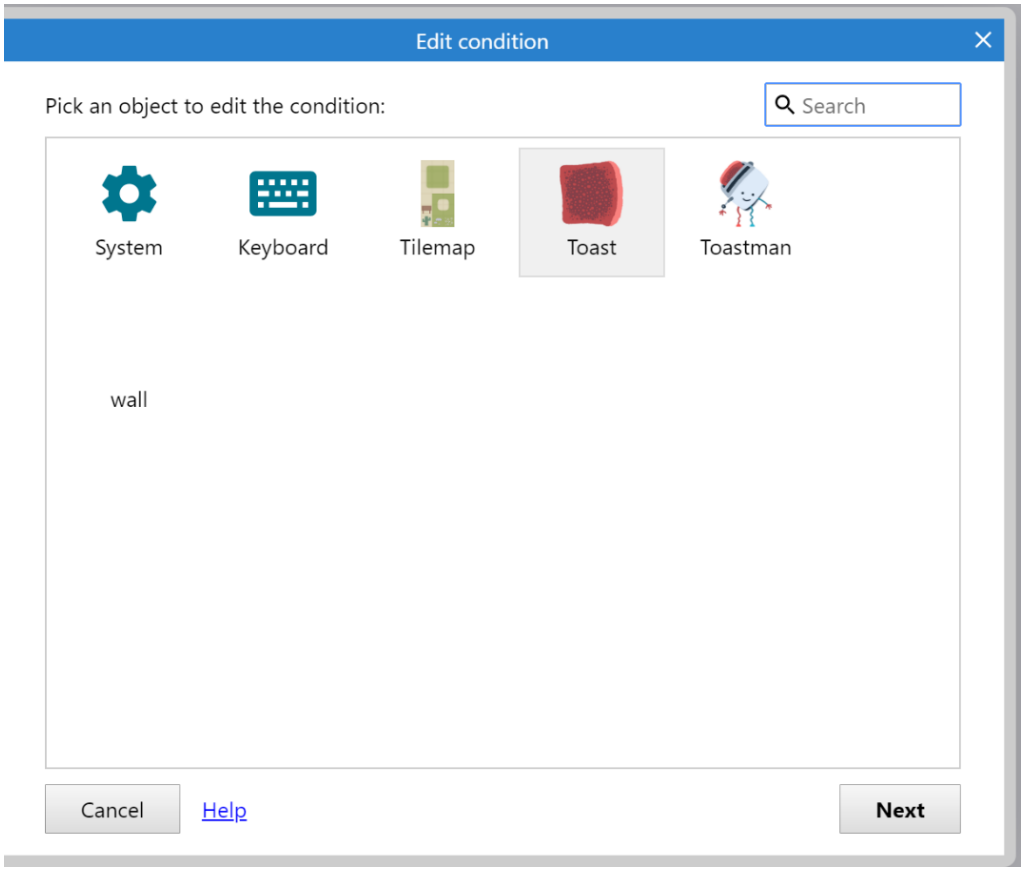
That's a bit of an oversimplification. Construct provides a lot of event features to cover lots of different things you might need to do. However, for now, that's a good way to think about it.

Your first event

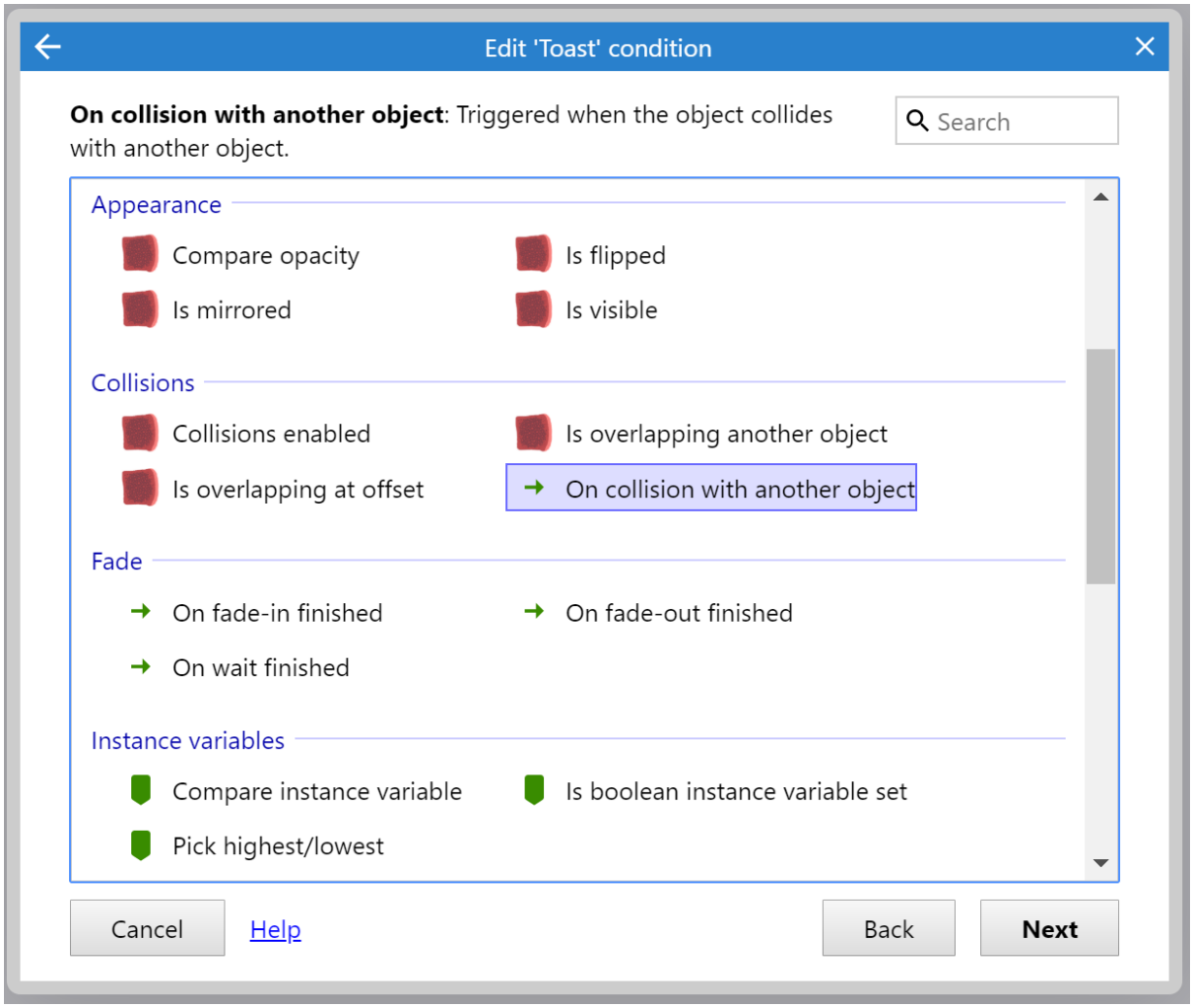
We want to make the Toast objects disappear if they are touched by a Bernie Crumbs object. To do this we need to make an event, it will look like this when we are done:



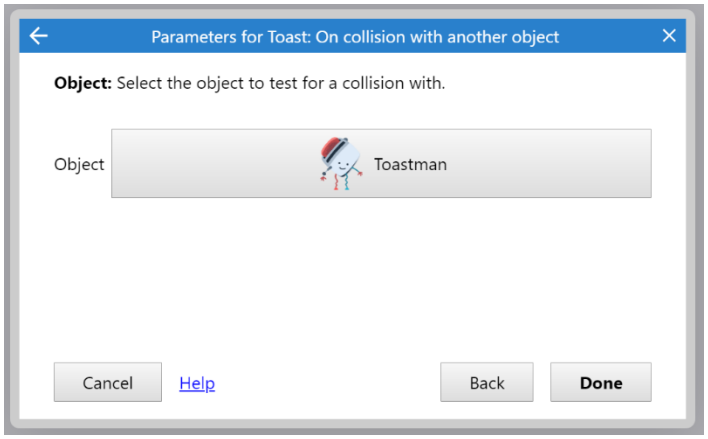
Let's start making this event. Double-click a space in the event sheet. This will prompt us to add a condition for the new event. Different objects have different conditions and actions depending on what they can do. There's also the System object, which represents Construct's built-in features. Double-click the Toast object as shown.



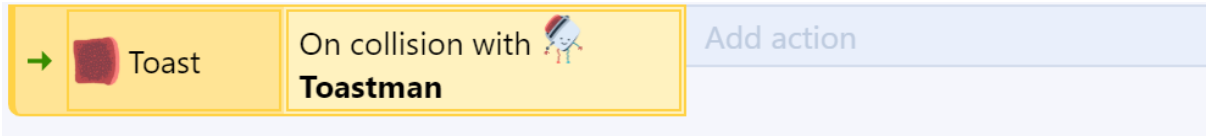
The dialog will then list all the Toast object's conditions. Double-click the *On collision with another object* condition to create an event with it:



You will then be asked to select the object to test for a collision with - select Bernie Crumbs. The object to test for a collision *with is called the parameter of the condition. Actions can have parameters too.*

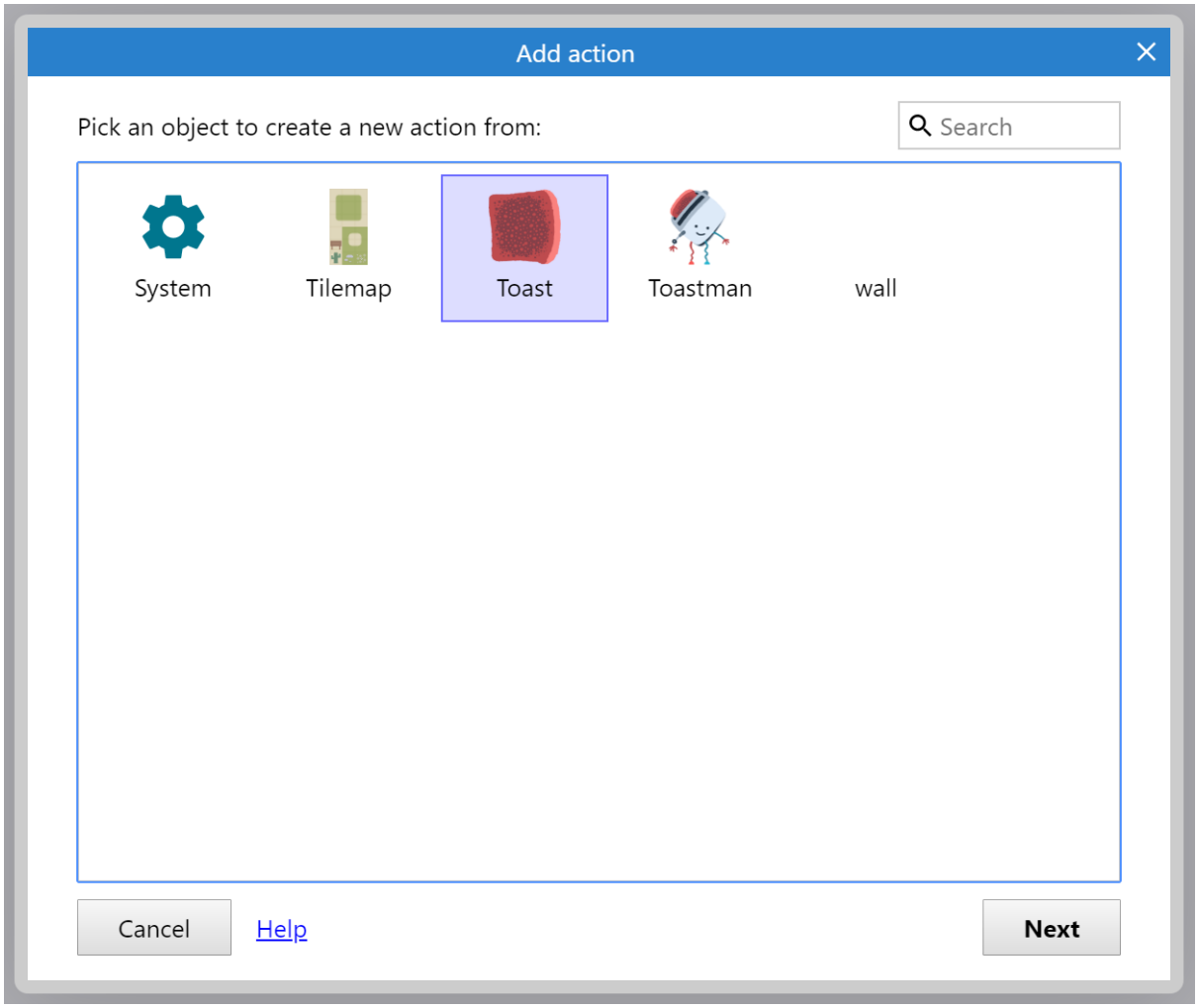


Click Done on the parameters dialogue. The dialog will close and the event is created, with no actions.

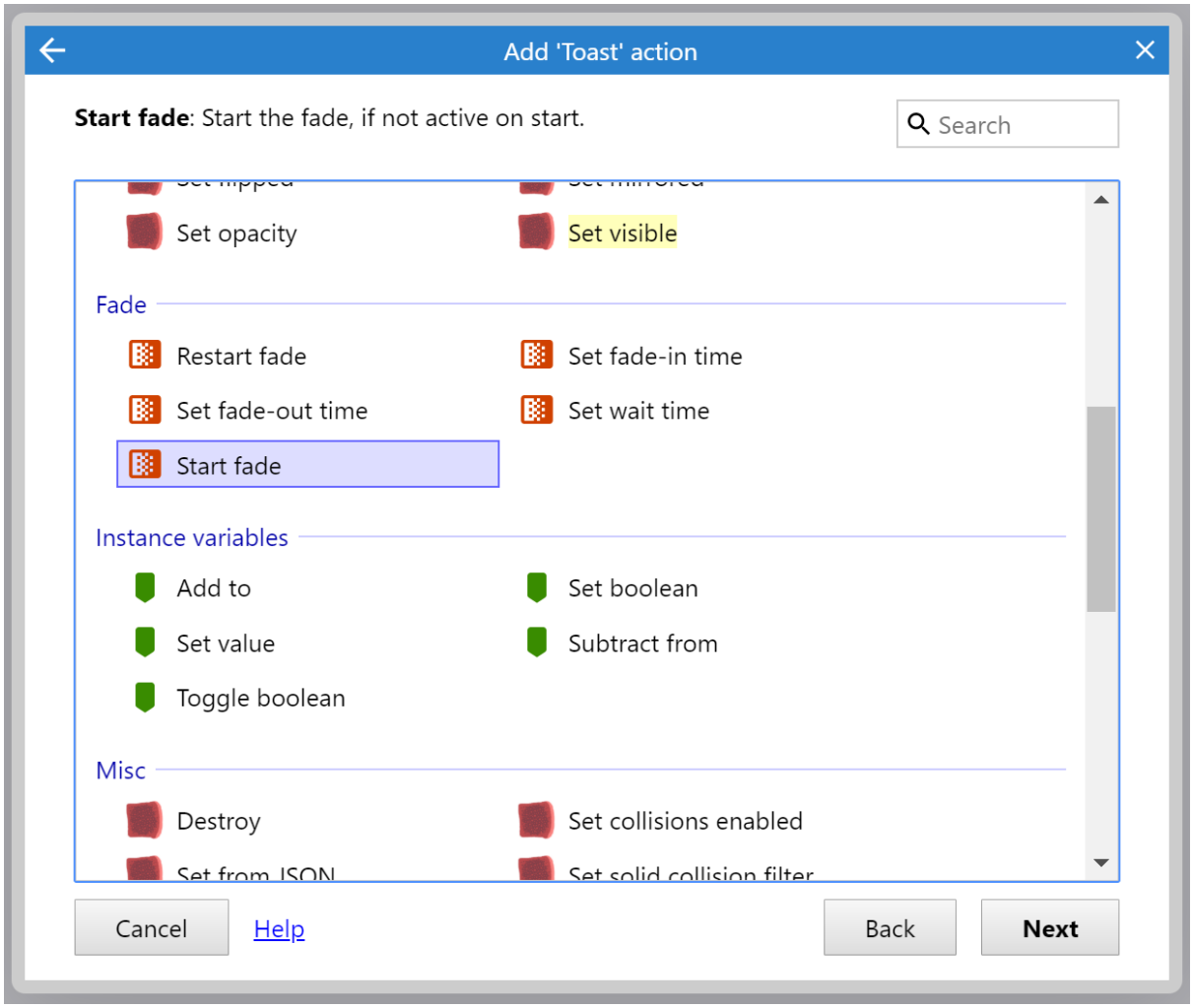


Now we want to add an action to make the Toast object fade if Bernie Crumbs collides with it. Click the Add action link to the right of the event. (Make sure you get the *Add action* link, not the *Add event* link underneath it which will add a whole different event again.) The Add Action dialog will 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!

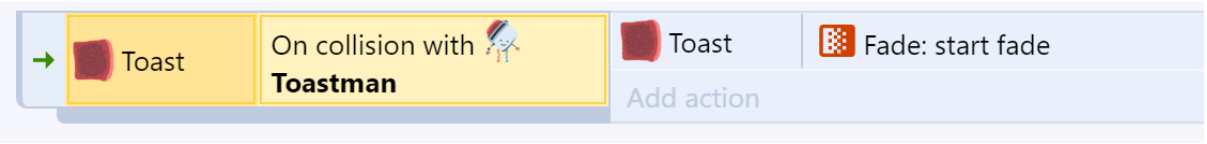
As shown, double-click the *Toast* object, as it is the Toast we want to fade out:



The list of actions available in the Toast object appears. Notice how the toast's fade behavior has its own actions. Select *Start fade*:



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 Bernie Crumbs should now be able to move around as before, but the Toast fades if Bernie Crumbs collides with it. This is our first bit of custom logic.

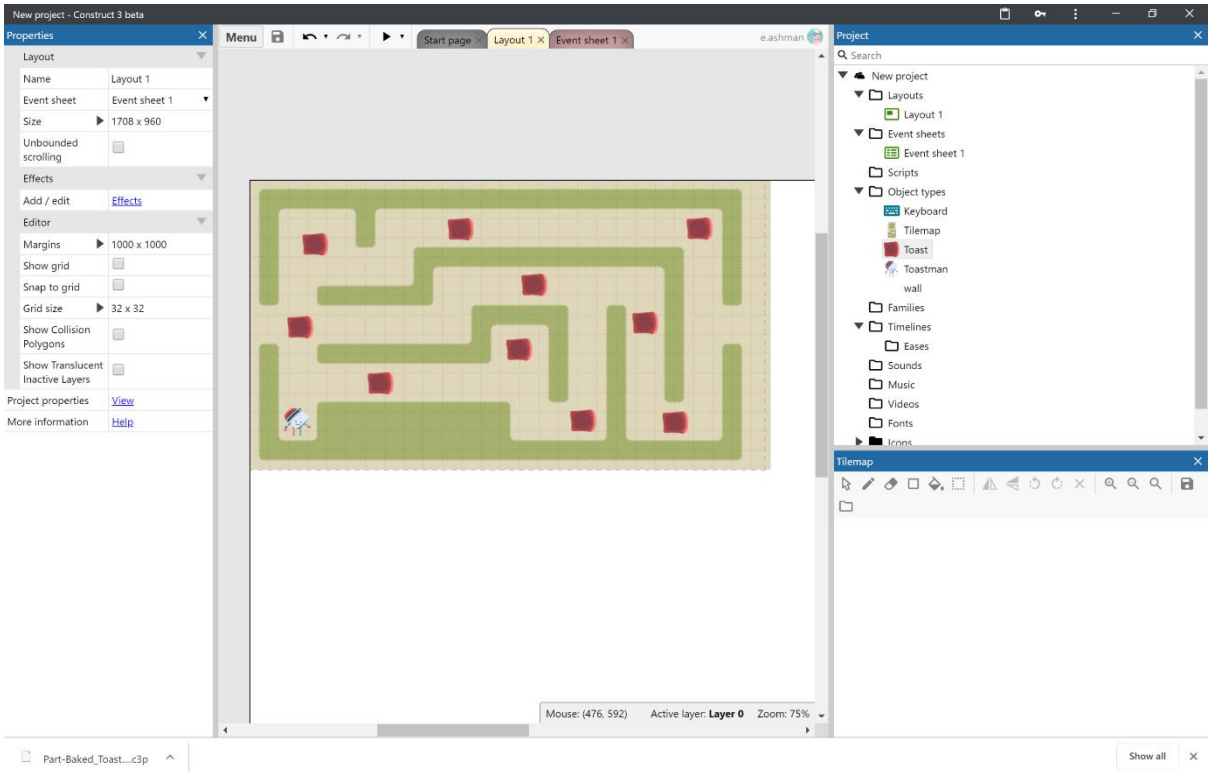
Create some more toast

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

Object types are essentially 'classes' of objects. In the event system, you mainly deal with object types. We've already made an event that says "Toast collides with Bernie Crumbs". This actually means "*Any instance of the* Toast *object type* collides

with *any instance of the* Bernie Crumbs *object type*" - as opposed to having to make a separate event for each and every toast. We'll cover more on object types vs. instances later. For now, a good example to think about is different types of food are different object types, then the actual food items themselves (which there might be several of) are instances of those object types.

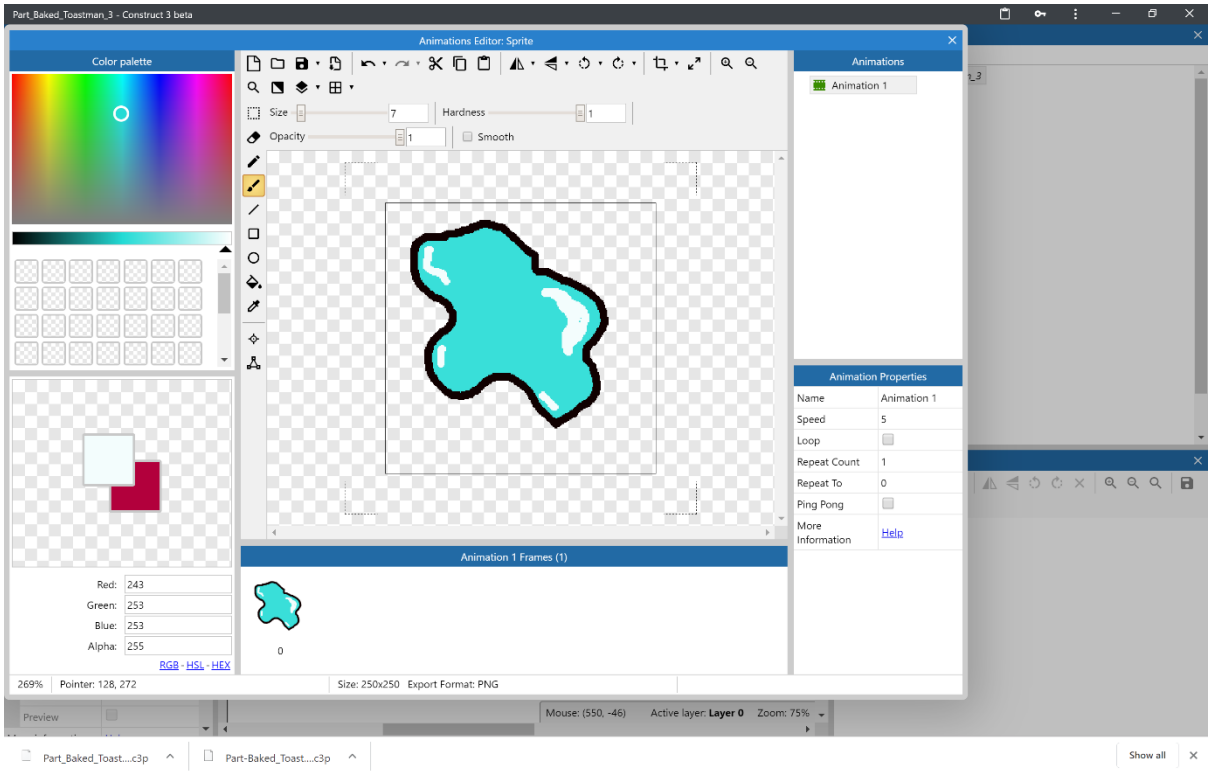
Using Control + drag, create 7 or 8 pieces of toast. Don't place any too close to Bernie Crumbs, or they might disappear straight away! Remember you can zoom out with Control+ mouse wheel down if it helps, and spread them over the whole layout. You should end up with something a bit like this.



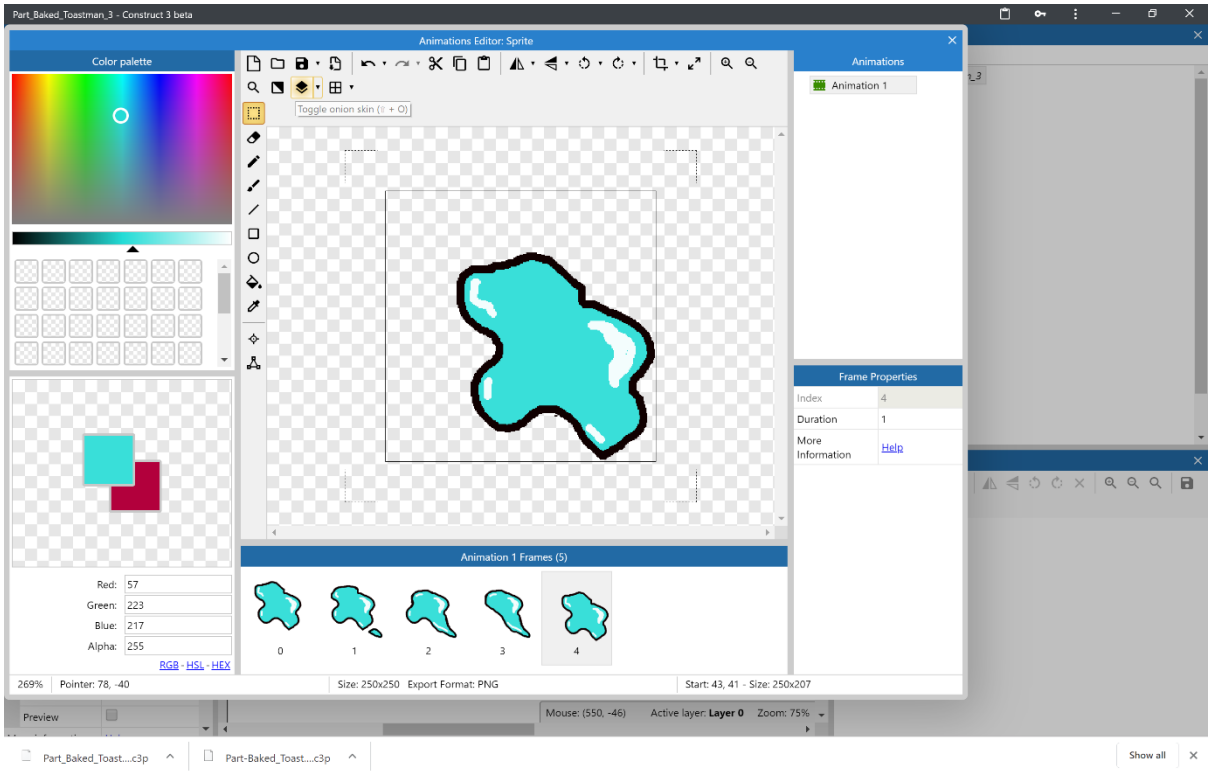
Preview your game again - Bernie Crumbs should be able to eat all those pieces of toast!

Adding enemys

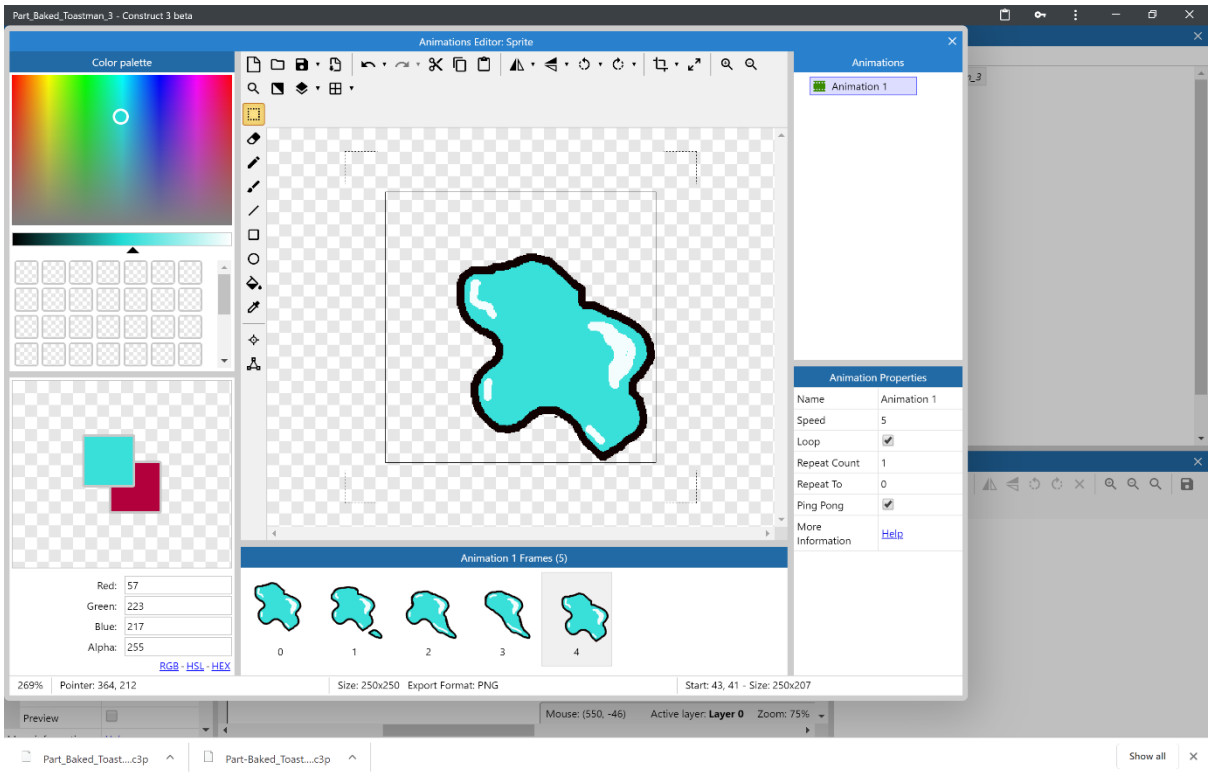
Next we need to add an enemy for Bernie Crumbs, of course the obvious enemy for a toaster is water! Add a new sprite and use the animations editor to design a splash of water, name the new sprite object water:



Use the frame editor at the bottom of the window to create an animation to make your water appear to move, you may like to turn on onion skinning so you can see the previous frames as you draw a new one:



Click on the name of your animation (in this case Animation 1) and experiment with different settings until your animation looks right:

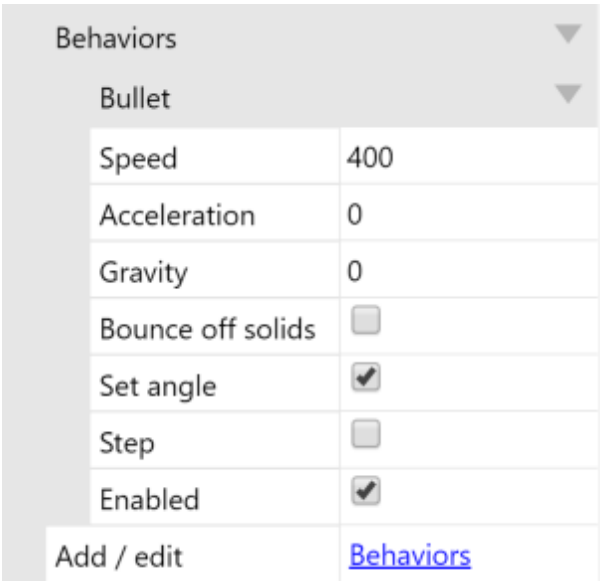


Add enemy behaviours

Add the Bullet movement and Wrap (don't forget to change to Wrap to Viewport) to the Water object (because it just moves forwards at a slowish speed)

If you run the game, you might notice that any Water you can see suddenly shoot off rather quickly!

Let's start by slowing down the Water to a leisurely pace. Select the Water 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).

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 Toast▶On collision with another object, and for Object: Bernie Crumbs

Add action Toast▶Start fade

Destroy Bernie Crumbs if hit by a water

When a Water hits Bernie Crumbs he should be destroyed and the game ends. This can be done with the *Destroy* action in Bernie Crumbs, which destroys the instance of Bernie Crumbs that collided with a Water object.

Make the following event:

Condition: Water▶On collision with another object▶Bernie Crumbs

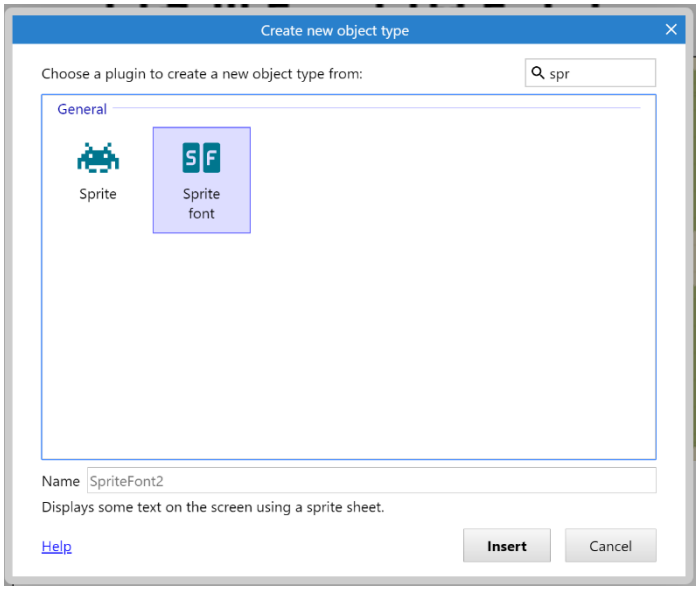
Action: Bernie Crumbs▶Destroy

Your event should now look like this:

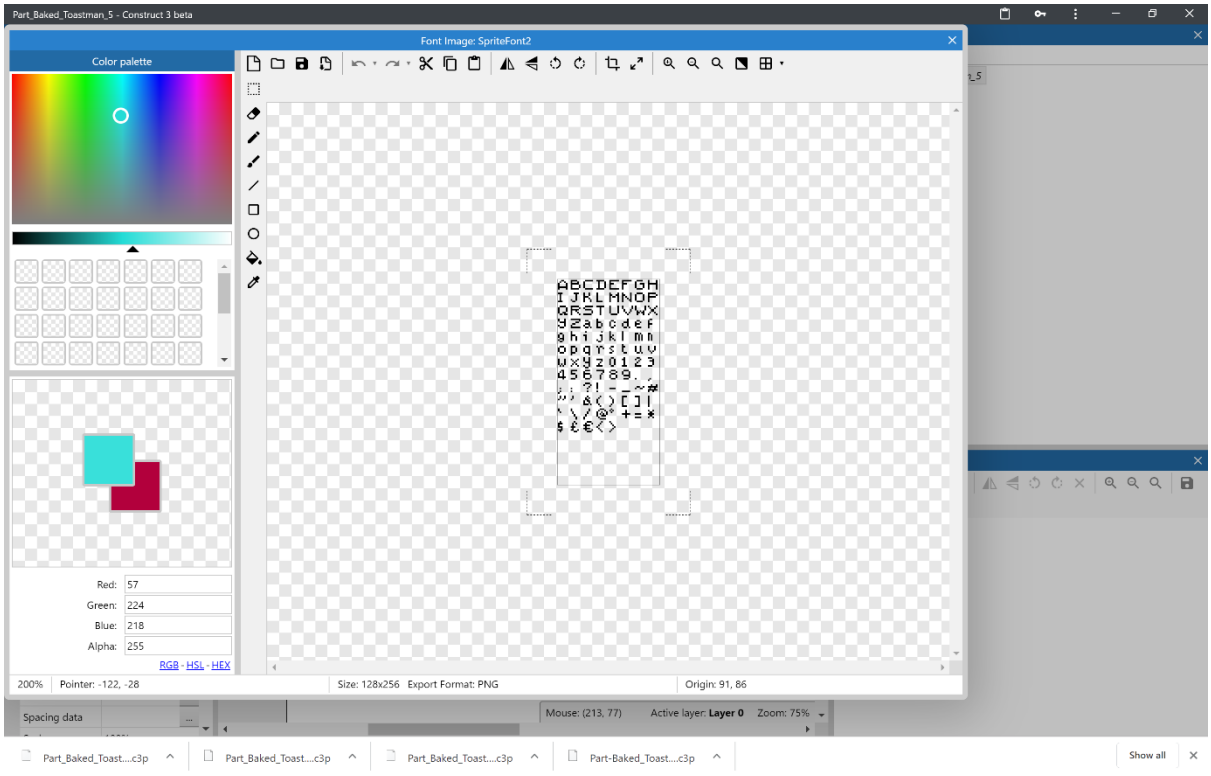
Add game over text

If you run the game Bernie Crumbs is destroyed if he is hit by a Water but there is no GAME OVER to tell you what is going on. Let's fix that by making the GAME OVER text appear if Bernie Crumbs is destroyed.

To do this we are going to add a new object, this time we will add the spritefont object:



Click in the layout roughly where you want the Game Over! text to be, the animation editor will open and look like this:



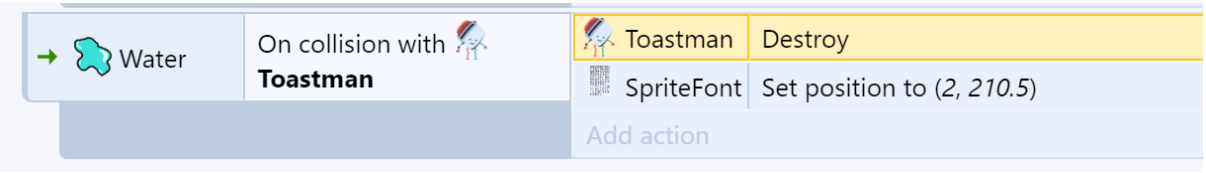
Don’t make any changes, just close the animations editor. Double click where the word text has appeared in the layout, edit the text to say Game Over!:

Move the *Game Over* sprite to somewhere off the edge of the layout - we don't want to see it when the game starts.

Go back to the events tab and click on add action below the destroy Bernie Crumbs action (yes you can add more than one action to a condition), add the following action:

Add action Game Over▶Set position to, and for X: 427, Y: 240 (this will make the Game Over object appear in the centre of the viewport)

Your event should now look like this:



Test your game again - you should now find that when Bernie Crumbs is destroyed, your Game Over object appears in the centre of the screen.

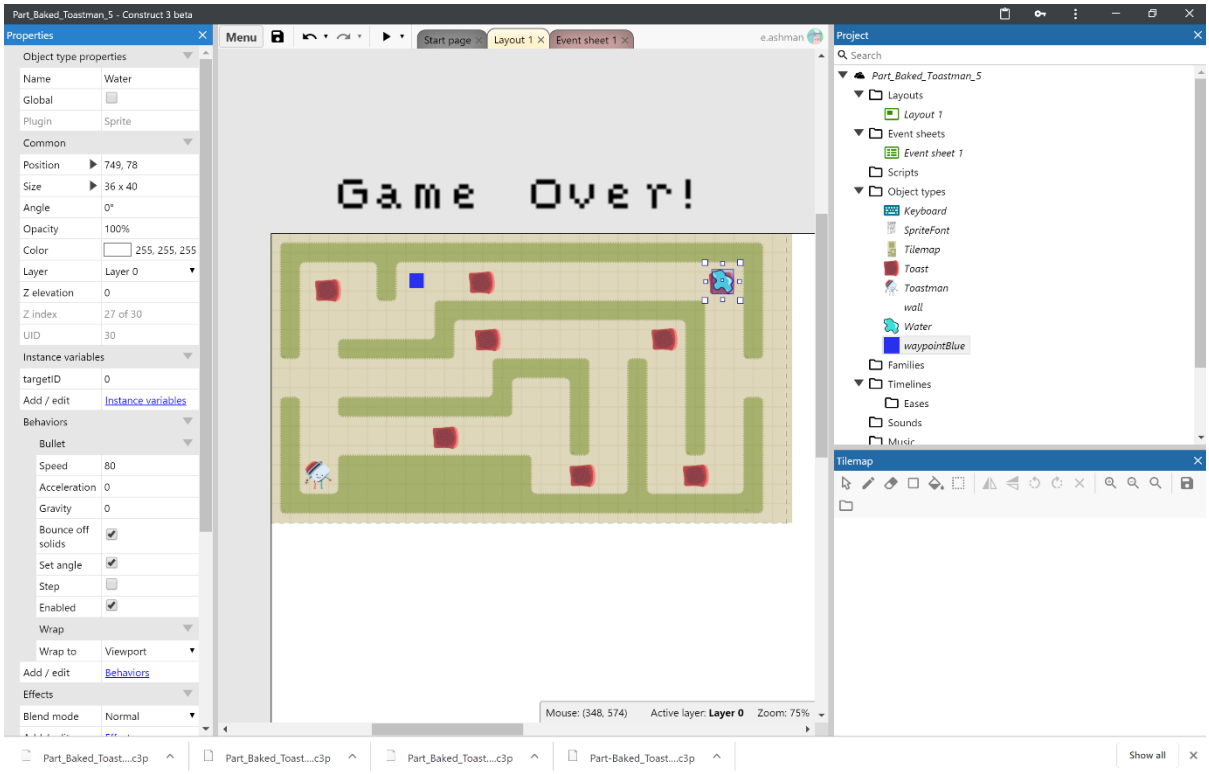
Making your water follow a path around your maze using waypoints

At the moment the water sprite just bounces around and its really just chance if it begins to move around the maze, to make the water appear to chase Bernie Crumbs we need to use some Artificial Intelligence. Artificial Intelligence (or AI for short) sounds really fancy but really it's just making things in your game appear like they have some intelligence and can think for themselves.

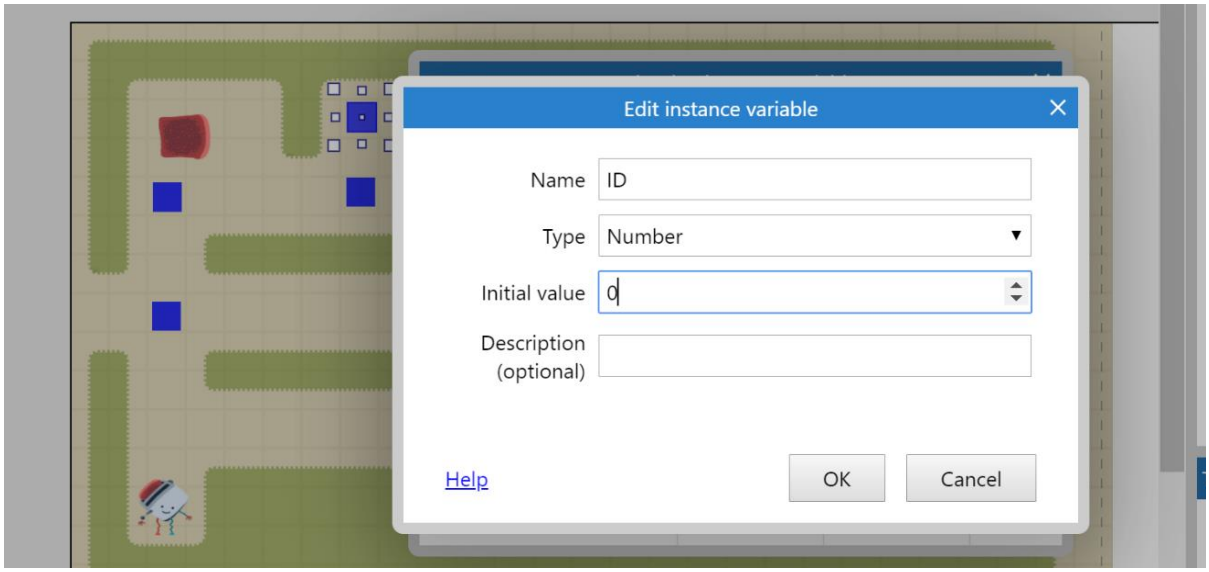
In the original Pacman all of the Ghosts appeared to chase Pacman around the maze - in fact they followed set routes that each repeated over and over again - except when they were turned into blue Ghosts but we can worry about that later. We call follow instructions like this an algorithm.

The way we will be doing this is by using waypoints - in other words using sprites as points that the Water must touch to make it appear to follow a route.

Go back to the layout tab and add a sprite that is a coloured square. You can do this by double clicking on the background and filling the square with the colour required. Name the square 'waypointBlue' (or whatever colour your square is).

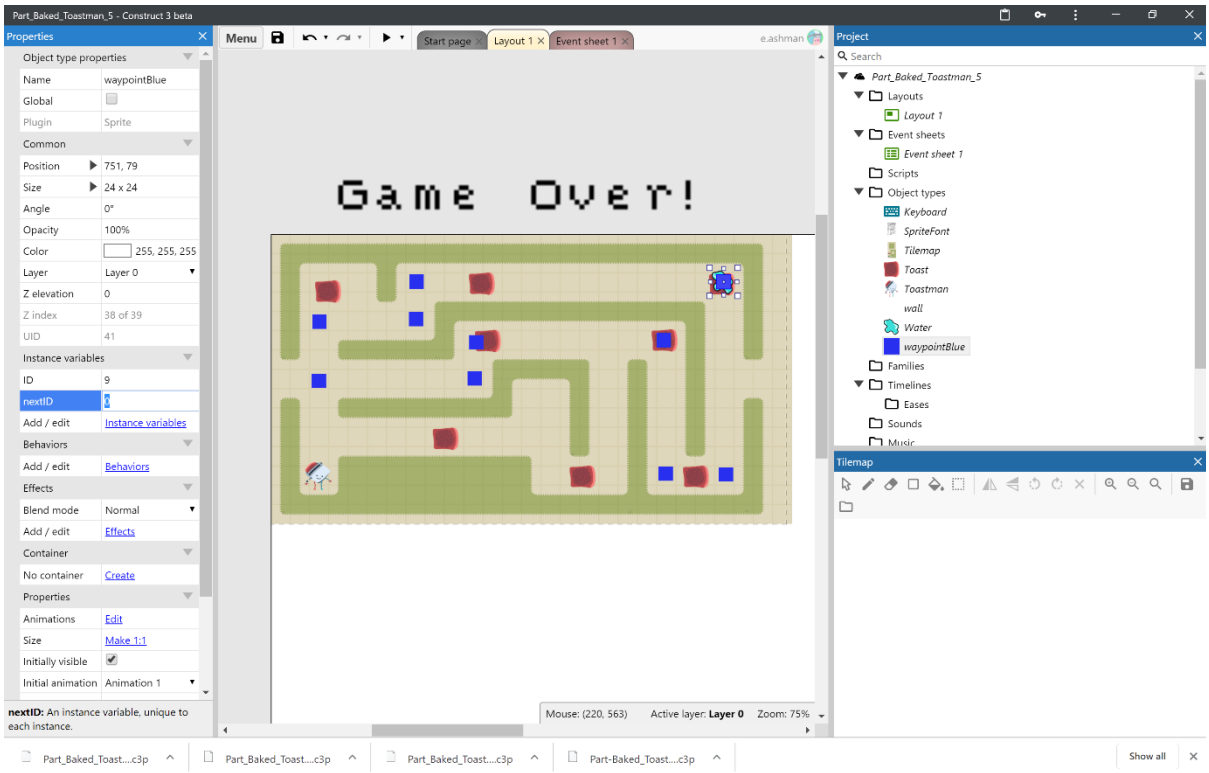


Now you are going to create instance variables which will be used to control the movement of your water. With your square still selected add the instance variable 'ID', leaving its initial value as 0:

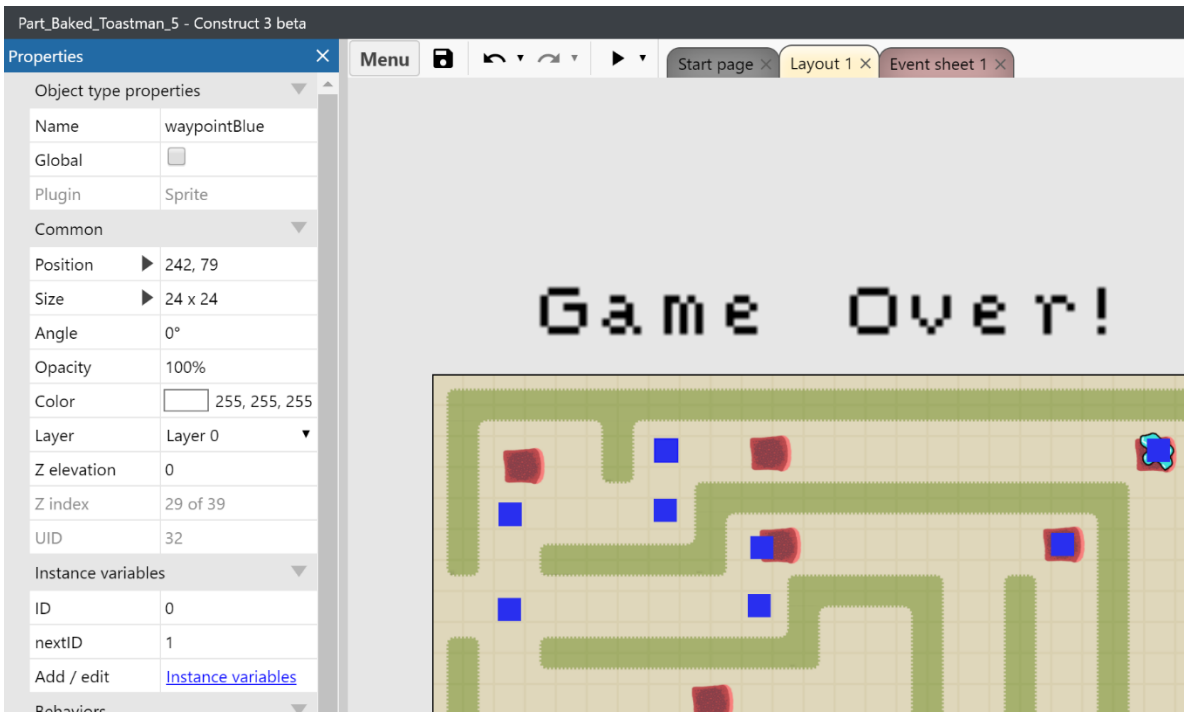


Create a second instance variable called 'NextID' again leave the initial value as 0.

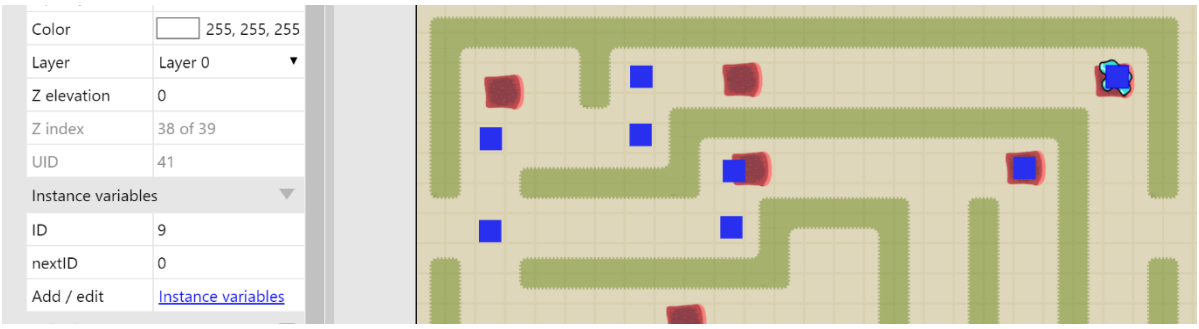
Now use ctrl and click and drag to create multiple copies of your waypointBlue sprite:



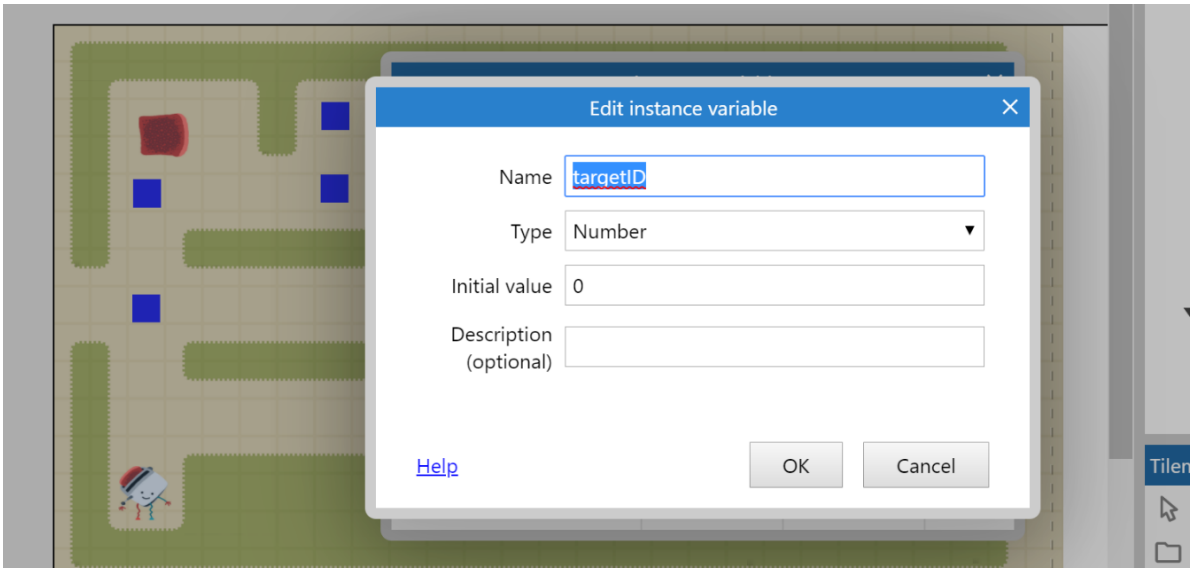
Change the instance variable ID of each of your waypointBlue sprites to the numbers 0,1,2 etc. in the order they should be visited by your water. Then set each of the nextID variables to one more than the ID set for that instance:



Set the nextID of the last of your waypointBlue sprite to 0. This means that when your water follows the path it will loop back to the beginning again:



Select your water sprite and then add an instance variable called targetID, leave the initial value as 0:



Select the event sheet tab.

You will need to add two events.

This is the first:

Add condition waypointBlue▶Compare instance variable, leave Instance variable as ID (number) and Comparison as = Equal to. Set Value to: water.targetID

Add action water▶Set angle of motion, and for Angle: angle (water.X, water.Y, waypointBlue.X, waypointBlue.Y)

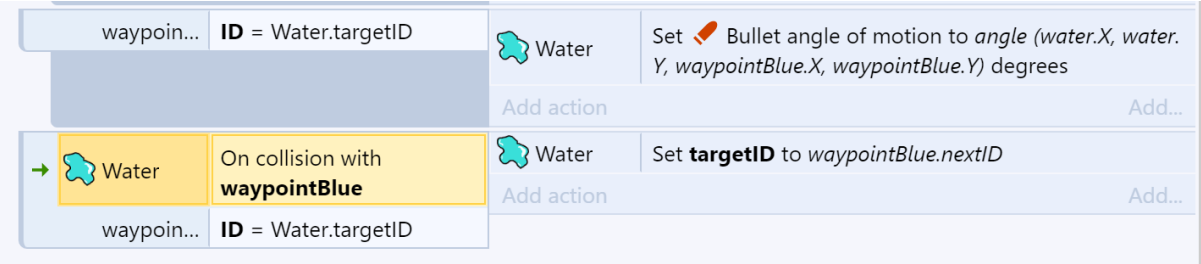
This is the second:

Add first condition water►On collision with another object, and for Object: waypointBlue

Add second condition waypointBlue►Compare instance variable, leave Instance variable as ID (number) and Comparison as = Equal to. Set Value to: water.targetID

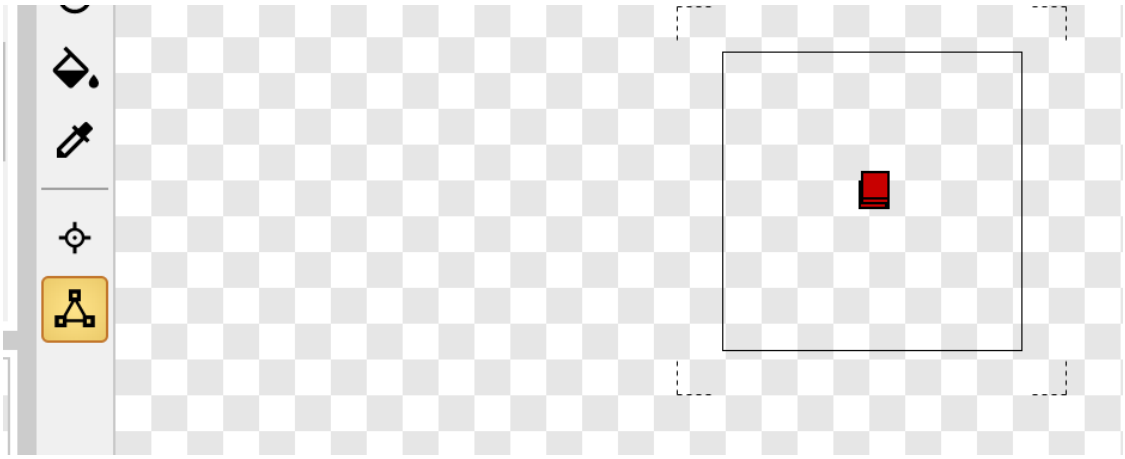
Add action water►Set value, and for Instance variable set targetID (number): waypointBlue.nextID

Your events should look like this:



Test your game and you should find that your water follows your waypoints.

To make your waypoints invisible just double click on one of them and remove your fill by using the eraser tool. You can also experiment with making the wayPoints smaller and editing the collision polygon:



Computing Programmes of Study Links

This tutorial covers the following content from the KS3 National Curriculum Programme of Study for Computing:

Key stage 3

Pupils should be taught to:

- design, use and evaluate computational abstractions that model the state and behaviour of real-world problems and physical systems
- understand several key algorithms that reflect computational thinking [for example, ones for sorting and searching]; use logical reasoning to compare the utility of alternative algorithms for the same problem
- use 2 or more programming languages, at least one of which is textual, to solve a variety of computational problems; make appropriate use of data structures [for example, lists, tables or arrays]; design and develop modular programs that use procedures or functions
- understand simple Boolean logic [for example, AND, OR and NOT] and some of its uses in circuits and programming;
- undertake creative projects that involve selecting, using, and combining multiple applications, preferably across a range of devices, to achieve challenging goals, including collecting and analysing data and meeting the needs of known users
- create, reuse, revise and repurpose digital artefacts for a given audience, with attention to trustworthiness, design and usability
- understand a range of ways to use technology safely, respectfully, responsibly and securely, including protecting their online identity and privacy; recognise inappropriate content, contact and conduct, and know how to report concerns

It also covers the following competencies from Computing at School's Progression Pathways:

	Algorithms	Programming	Data Representation	Hardware and Processing	Communication and Networks	IT
Pink	Understands that computers need precise instructions. **(AL)** Demonstrates care	Knows that users can develop their own programs, and can demonstrate this by creating a	Recognises that digital content can be represented in many forms. **(AB)** **(GE)**	Understands that computers have no intelligence and that computers can do nothing unless a	Obtains content from the world wide web using a web browser **(AL)**	Uses software under the control of the teacher to create, store and edit digital content using

	and precision to avoid errors. **(AL)**	simple program in an environment that does not rely on text e.g. programmable robots etc. **(AL)** Executes, checks and changes programs. **(AL)** Understands that programs execute by following precise instructions. **(AL)**		program is executed. **(AL)** Recognises that all software executed on digital devices is programmed. **(AL) (AB) (GE)**		appropriate file and folder names. **(AB) (GE) (DE)** Understands that people interact with computers Knows common uses of information technology beyond the classroom. **(GE)**
Yellow	Understands that algorithms are implemented on digital devices as programs. **(AL)** Designs simple algorithms using loops, and selection i.e. if statements. **(AL)** Uses logical reasoning to predict outcomes. **(AL)** Detects and	Uses arithmetic operators, if statements, and loops, within programs. **(AL)** Uses logical reasoning to predict the behaviour of programs. **(AL)** Detects and corrects simple semantic errors i.e. debugging, in programs. **(AL)**	Recognises different types of data: text, number. **(AB) (GE)** Appreciates that programs can work with different types of data. **(GE)**	Recognises and can use a range of input and output devices	Navigates the web and can carry out simple web searches to collect digital content. **(AL) (EV)** Demonstrates use of computers safely and responsibly, knowing a range of ways to report unacceptable content and contact when online.	Uses technology with increasing independence to purposefully organise digital content. **(AB)** Shows an awareness for the quality of digital content collected. **(EV)** Uses a variety of software to manipulate and present digital content: data and information. **(AL)**

	corrects errors i.e. debugging, in algorithms. **(AL)**					
Orange		Creates programs that implement algorithms to achieve given goals. **(AL)** Declares and assigns variables. **(AB)** Uses post-tested loop e.g. 'until', and a sequence of selection statements in programs, including an if, then and else statement. **(AL)**		Knows that computers collect data from various input devices, including sensors and application software. **(AB)**	Understands the difference between the internet and internet service e.g. world wide web. **(AB)** Shows an awareness of, and can use a range of internet services e.g. VOIP. Recognises what is acceptable and unacceptable behaviour when using technologies and online services.	Collects, organises and presents data and information in digital content. **(AB)** Creates digital content to achieve a given goal through combining software packages and internet services to communicate with a wider audience e.g. blogging. **(AL)**
Blue					Selects, combines and uses internet services. **(EV)** Demonstrates responsible use of technologies and online services, and knows a range of ways to report concerns.	Makes judgements about digital content when evaluating and repurposing it for a given audience. **(EV)** **(GE)** Recognises the audience when designing and creating digital content. **(EV)**

Purple	Understands that iteration is the repetition of a process such as a loop. **(AL)**	Uses a range of operators and expressions e.g. Boolean, and applies them in the context of program control. **(AL)** Selects the appropriate data types. **(AL) (AB)**		Knows that there is a range of operating systems and application software for the same hardware. **(AB)**		Evaluates the appropriateness of digital devices, internet services and application software to achieve given goals. **(EV)**
Red	Understands a recursive solution to a problem repeatedly applies the same solution to smaller instances of the problem. **(AL) (GE)**				Uses technologies and online services securely, and knows how to identify and report inappropriate conduct. **(AL)**	Justifies the choice of and independently combines and uses multiple digital devices, internet services and application software to achieve given goals. **(EV)** Evaluates the trustworthiness of digital content and considers the usability of visual design features when designing and creating digital artifacts for a known audience. **(EV)**
Black						Considers the properties of media when importing them into digital artefacts. **(AB)**

White		Understands and uses two dimensional data structures. **(AB) (DE)**				