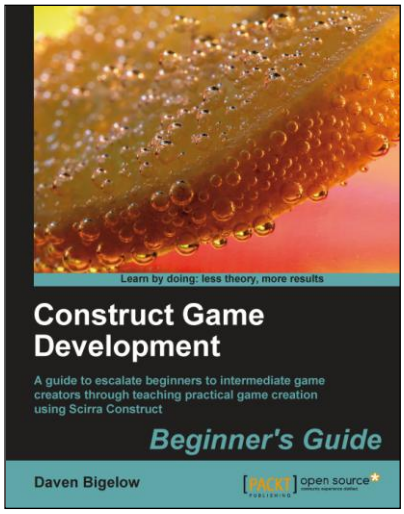
[PACKT] open source
community experience distilled
PUBLISHING

Construct Game Development Beginner's Guide

Daven Bigelow



Chapter No. 2

"Hello World! Construct Style"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.2 "Hello World! Construct Style"

A synopsis of the book's content

Information on where to buy this book

About the Author

Daven Bigelow is a hobby game developer and a software programmer. He has been creating 2D games for over eight years, across different game creation tools and programming languages. However, most of his experience lies in Construct Classic, which has been his tool of choice over the last three years.

He can often be found on the Scirra forums under the name Jayjay, where he provides advice and examples for new users seeking help.

I would like to thank all my friends and family who encouraged me along the way. I also send thanks to the publisher, Packt Publishing, and all of its employees for their efforts.

Lastly, I thank you, the reader, for reading this book. I hope that it meets all of your expectations.

Construct Game Development Beginner's Guide

Welcome to *Construct Game Development Beginner's Guide*. In this book, you will be learning to use the free and open source software Construct Classic to make your own video games from scratch.

Construct Classic is a DirectX 9-based game creation environment for Windows, designed for making 2D games. Construct Classic uses a graphical event-based system for defining how the game behaves, in a visual, human-readable way—you do not need to program or script anything at all. It's intuitive for beginners, but powerful enough for advanced users to work without hindrance.

So, if you have ever wanted to make video games, and haven't tried before, this book will help you get started!

What This Book Covers

Chapter 1, Our First Look at Construct, covers the basics of the Construct Classic editor.

Chapter 2, Hello World! Construct Style, covers the making our first game, a classic plaformer.

Chapter 3, Adding the Challenge, covers creating enemies and a goal for our platform game.

Chapter 4, Making Noise, covers playing music and sound files in Construct Classic.

Chapter 5, Practical Physics, covers making our second game with the built-in physics engine.

Chapter 6, Custom Levels, covers making a level editor to save and load external level files.

Chapter 7, Platformer Revisited, a 2D Shooter, covers learning to make a platform shooter.

Chapter 8, I'm Throwing a Grenade, involves learning to use pixel shader effects in our games.

Chapter 9, Our Final Moments, covers a summary of what we've learned and some extra tips.

2
Hello World! Construct Style

Now that we know how to navigate around the interface, we can move on to making games. The first, we'll be making is a platformer, such as the classic Mario and Sonic, the Hedgehog games.

In this chapter, we are going to:

- Learn more about sprites, as well as how they can be animated
- Learn how to use the tiled background object to make levels and backgrounds
- Learn how to set the attributes of objects
- Learn what behaviors are and how they affect objects
- Learn what private and global variables are and how to define them
- Use textboxes to display information to the player
- Use events to control the game

So let's get started!

Sprites revisited

We've already covered the creation of a sprite object in Chapter 1, *Our First Look at Construct*. We'll now learn to create animations for them and get them ready for a real game. These skills are needed for making player and enemy sprites in nearly any game made in Construct.

Time for action – creating a player sprite

We are now going to make our player object and set the animations for it.

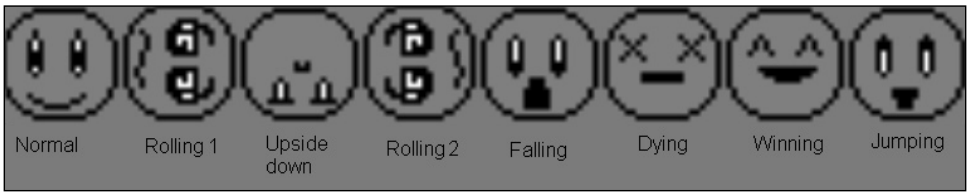
1. Start a blank game project (similar to the one we created in Chapter 1, *Our First Look at Construct*), and give it the name `MyPlatformer`. Then enter your author name in the **Creator** box.

Downloading the example code

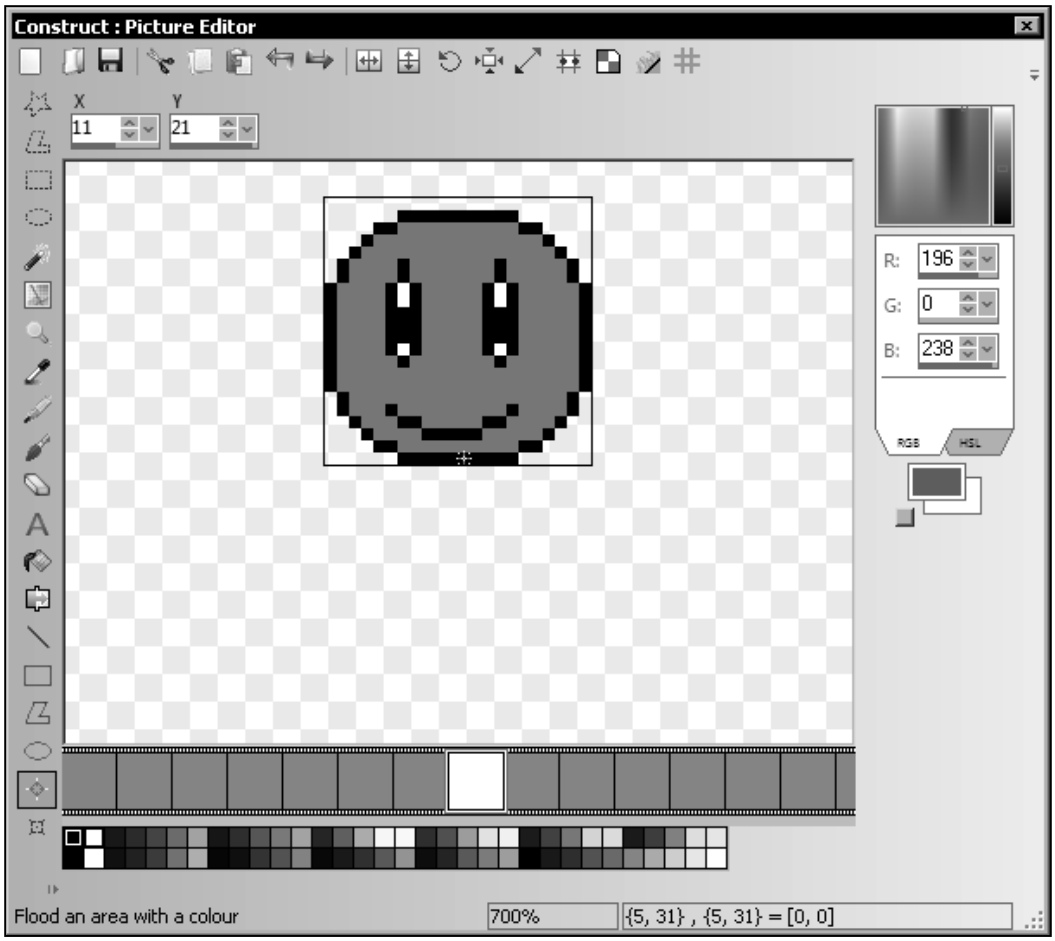
You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

2. Next, add a sprite into the layout.

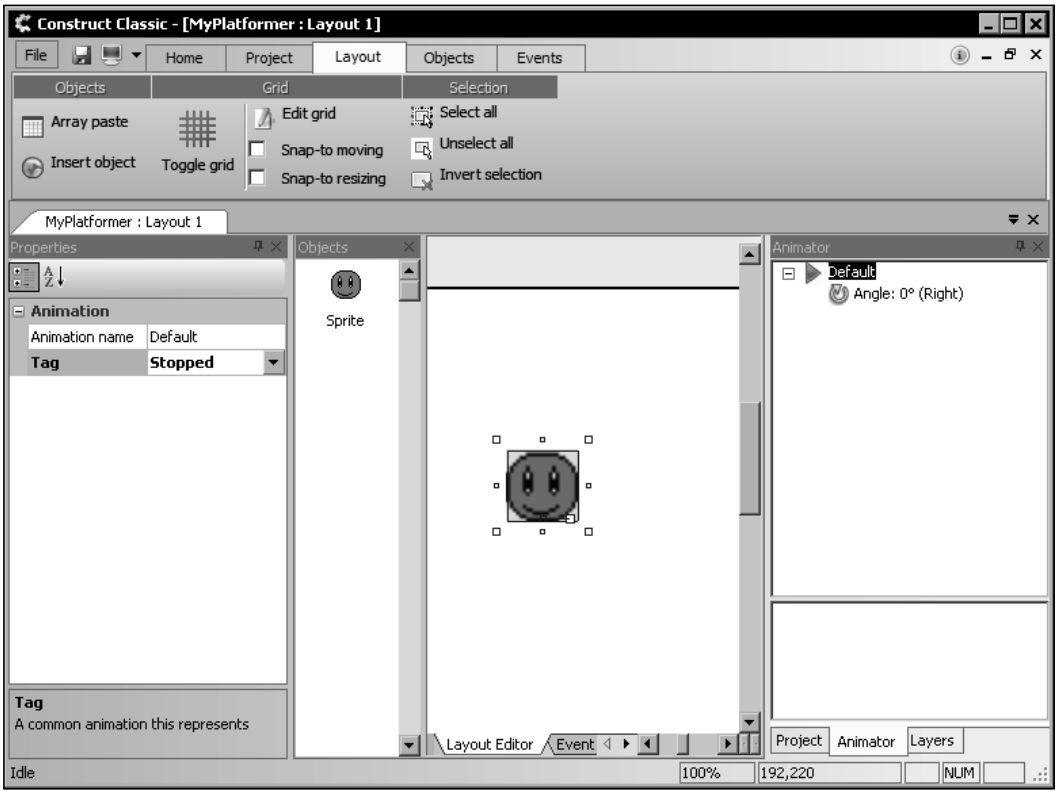
3. We're now going to need some player graphics. You can use Construct's Picture Editor to make these or use a painting program of your choice. For this game, the player is ball-shaped and rolls around the map. It has animation frames for falling, jumping, dying, and winning a level. See the following reference image (exact image size does not matter, as the player is resized in the layout editor):



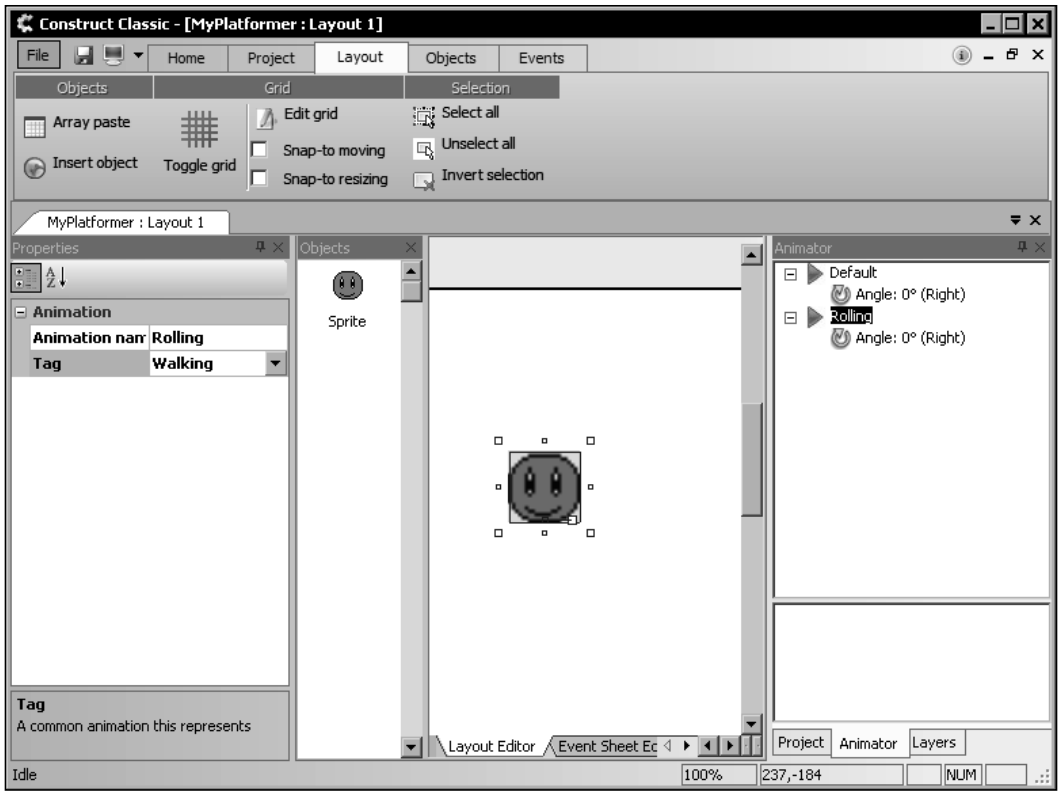
4. Now that we have these graphics ready, put/draw the first frame *Normal* into the image, and click on the **Hotspot** button (the red target icon) to set the hotspot at the bottom-center of the image, as shown in the following image. When finished with your player graphic, close the image editor, and click on **Yes** when it prompts you to save.



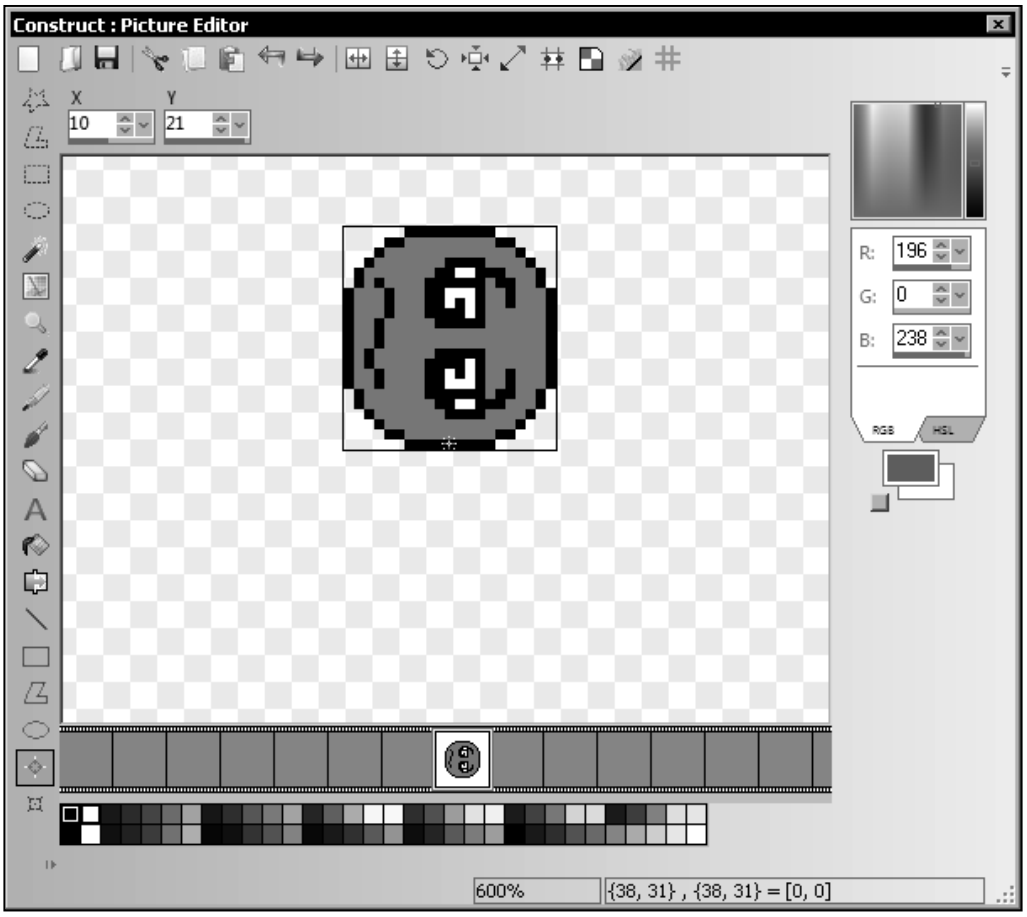
5. Now open the right-side menu to the **Animator** tab, and return to the left-side **Properties** menu. By selecting the Animation **Default** on the right menu, we can now apply a Tag to the animation using the left menu. Choose **Stopped**, as shown in the following screenshot:



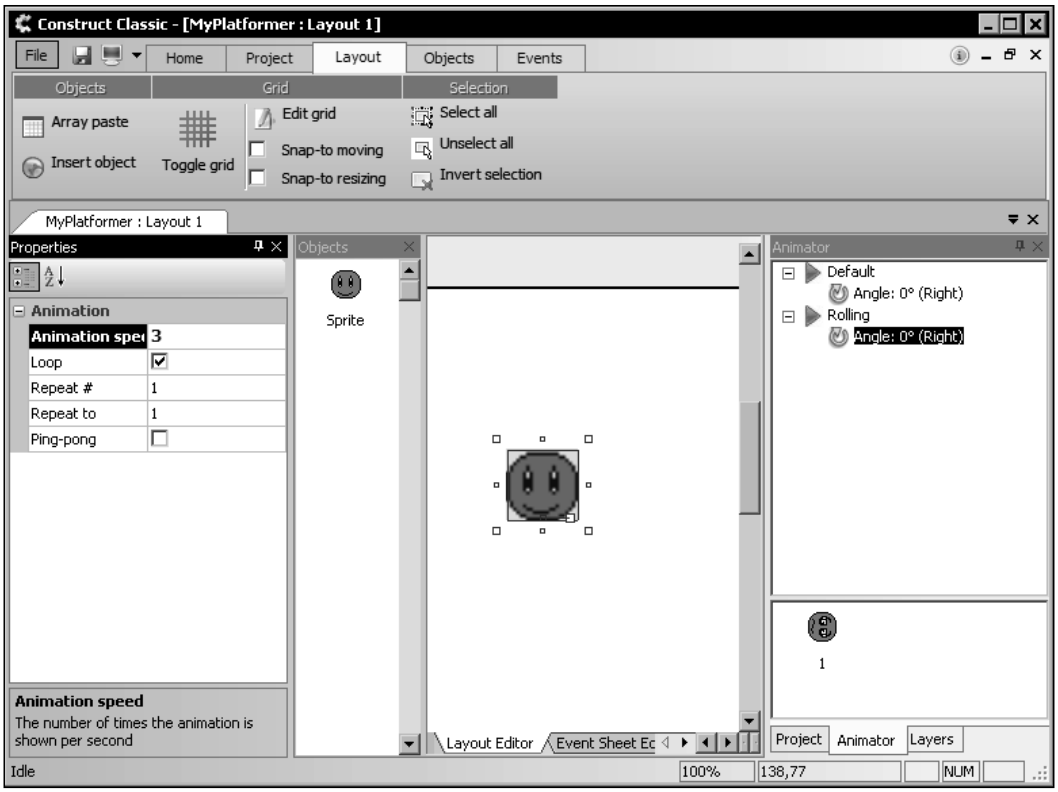
6. Now right-click on a blank area of the right menu; click and choose the option **Add new animation**. On the left side of the screen, we can now name this animation to **Rolling**. Give this animation the tag **Walking**, as shown in the following screenshot:



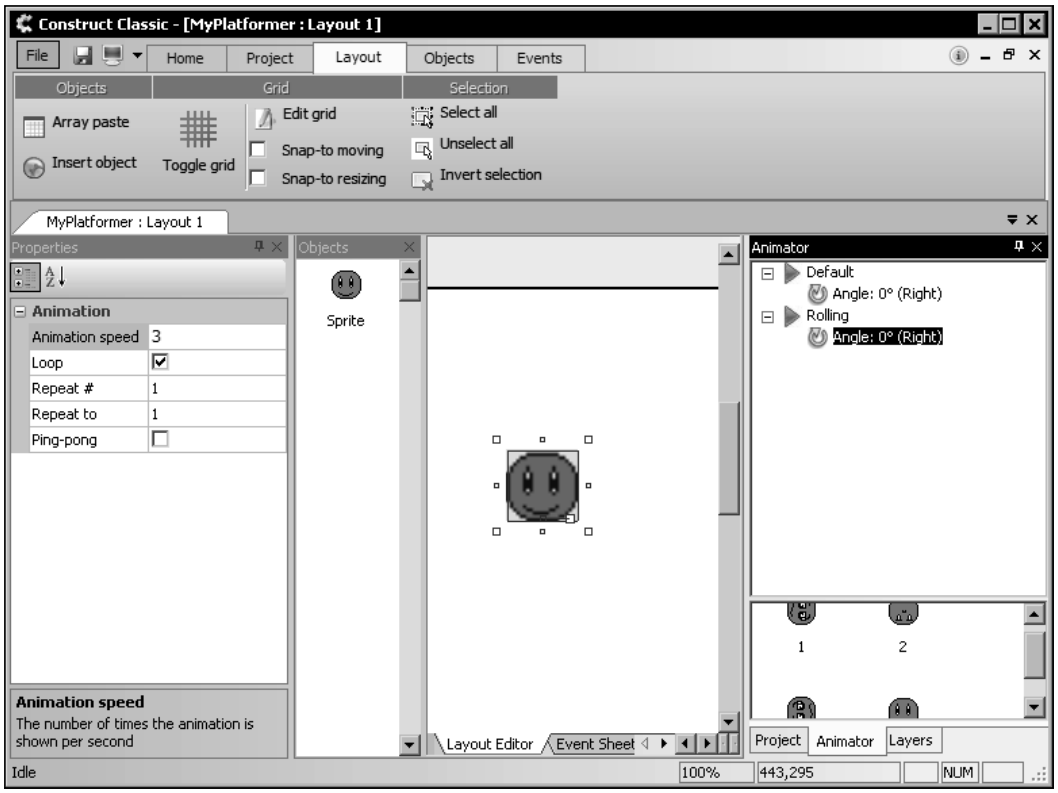
7. Now click on **Angle** underneath **Rolling**, and double-click the blank frame `1`, as shown in the following screenshot, to bring up the image editor. This will be the first image in our rolling sequence.



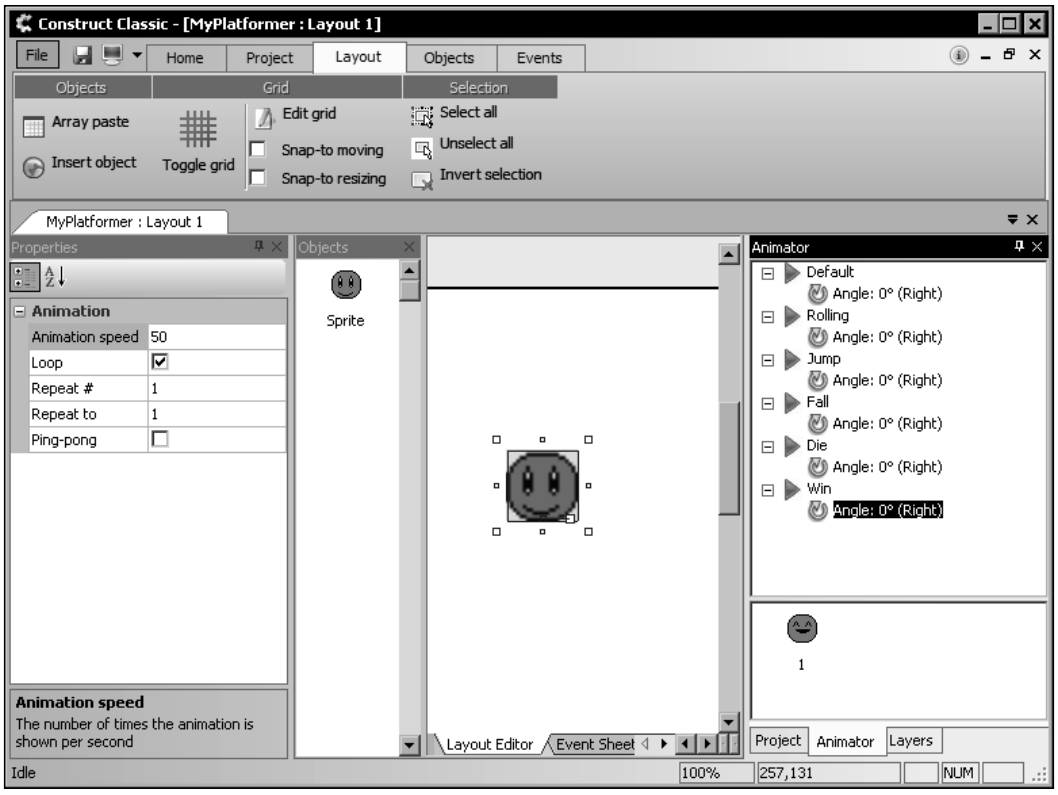
8. Now select the **Angle** again to edit the properties for that angle. We'll want the whole rolling animation to play around three times per second while the player is moving. Change the number in **Animation speed** from `50` to `3`.



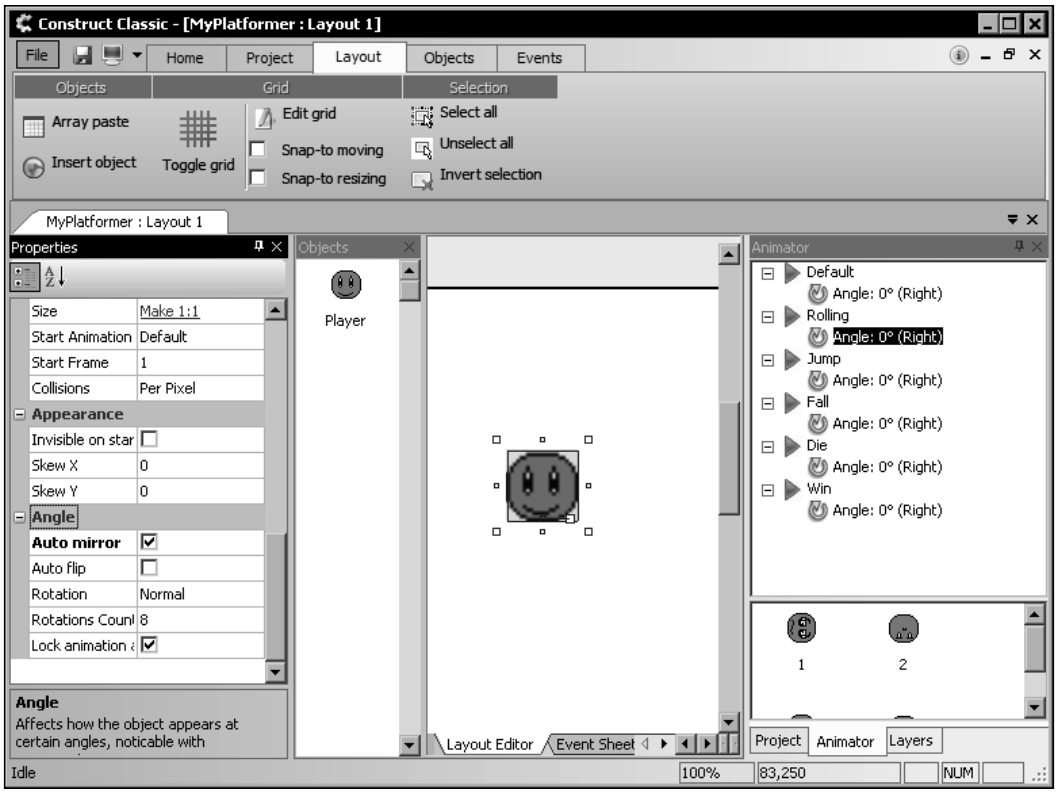
9. We can now continue adding the rest of our animation frames for **Rolling**. Right-click on the animation frames box, and click on **Add frame** from the context menu. Add the next four stages of the rolling sequence (rolling1, upside down, rolling2, and finally, normal again) into the animation so that the four animation frames are as seen in the following screenshot:



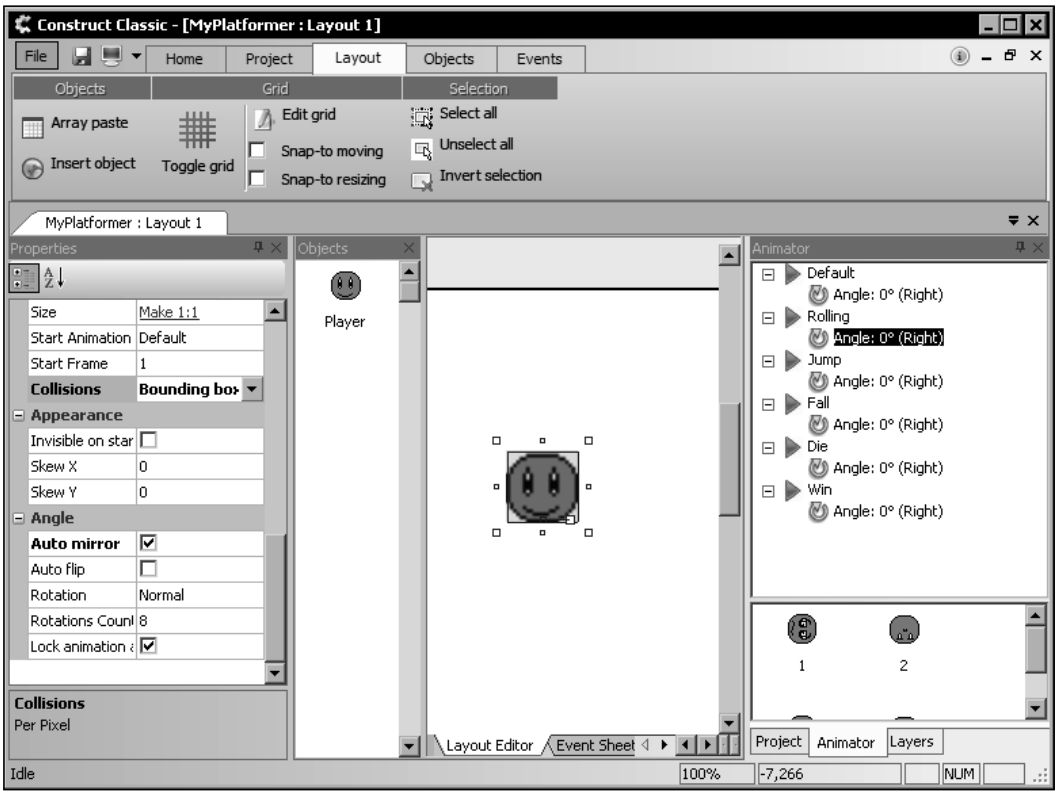
10. Now add the new animations **Jump**, **Fall**, **Die**, and **Win**, as in the following screenshot. For the **Jump** and **Fall** animations, give them the tags `Jumping` and `Falling` respectively.



11. Next, click on your sprite, and in the top field of the **Properties** box, enter a name such as `Player`. Then scroll to the bottom of the **Properties** window and tick the box **Auto mirror** in the group **Angle**, as in the following screenshot:



12. Finally, set the **Collisions** property of the sprite (in the **Properties** group) to **Bounding box**, as shown in the following screenshot:



What just happened?

We now have our player sprite drawn and animated. Let's take a closer look at what we did.

Creating new animations

As seen earlier, we created a new sprite and drew a picture for it. However, this time, we went on to create multiple animations and pictures.

Each animation can have multiple angles added to it; these angles are in increments of 45 degrees and will override the default rotation method of Construct. The default angle included with every new animation is 0 degrees (the direction right).

The hotspot of an image is its centre point, where the image is rotated and positioned around. We placed it at the same point in each image to ensure that the player object doesn't *jitter* as they roll around. A quick way to put the hotspot in corners, edges, or the centre of an image is to use the number pad keys.

Animation tags

Animation tags are used in Construct's built-in movements to define which animation plays when the player is moving in a certain way. They do not affect any other animations an object has, but can override them when the player begins moving.

However, the benefit of animation tags is that they will play the correct animation when an object is moved using the pre-made movement behaviors of Construct Classic, as we will see with this first game.

Choosing the Collisions mode

The Collisions selection is used to decide when a sprite is colliding with another.

The following collision modes are available:

- **None**: In this mode, no collisions will be reported for the object
- **Point**: In this mode, there are collisions only at the hotspot point of the object
- **Bounding box**: In this mode, the object collides as if it is in a box, which stretches to its size
- **Per Pixel**: In this mode, any non-transparent pixels of the object will be checked for collisions

Although each collision mode is more complex than the previous one and will require more processing, no noticeable effect will be produced in simple games besides how the objects move around each other.

Tiled backgrounds: defining the world

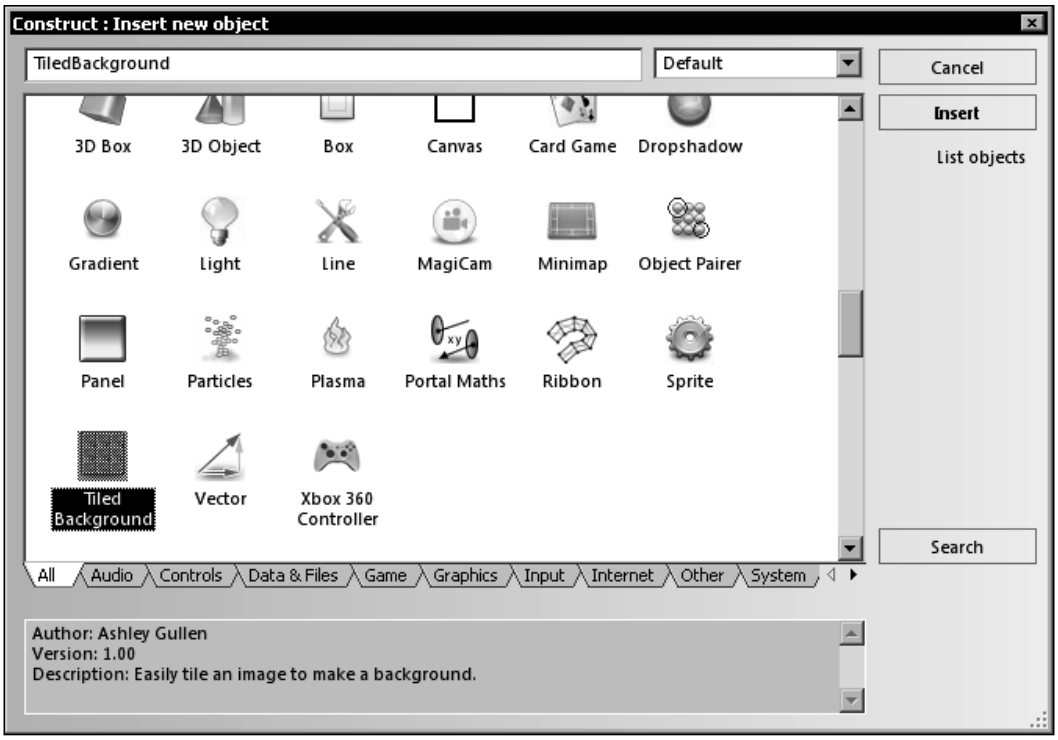
If you tried making a large image out of individual tile sprites, you'd quickly tire of trying to position each one. It is equally troublesome to store a large level in a single image, as they would take up too much video memory.

This is where the tiled background comes into picture. It repeats the same graphic, for as long and wide as you stretch it. These are great for making levels and worlds!

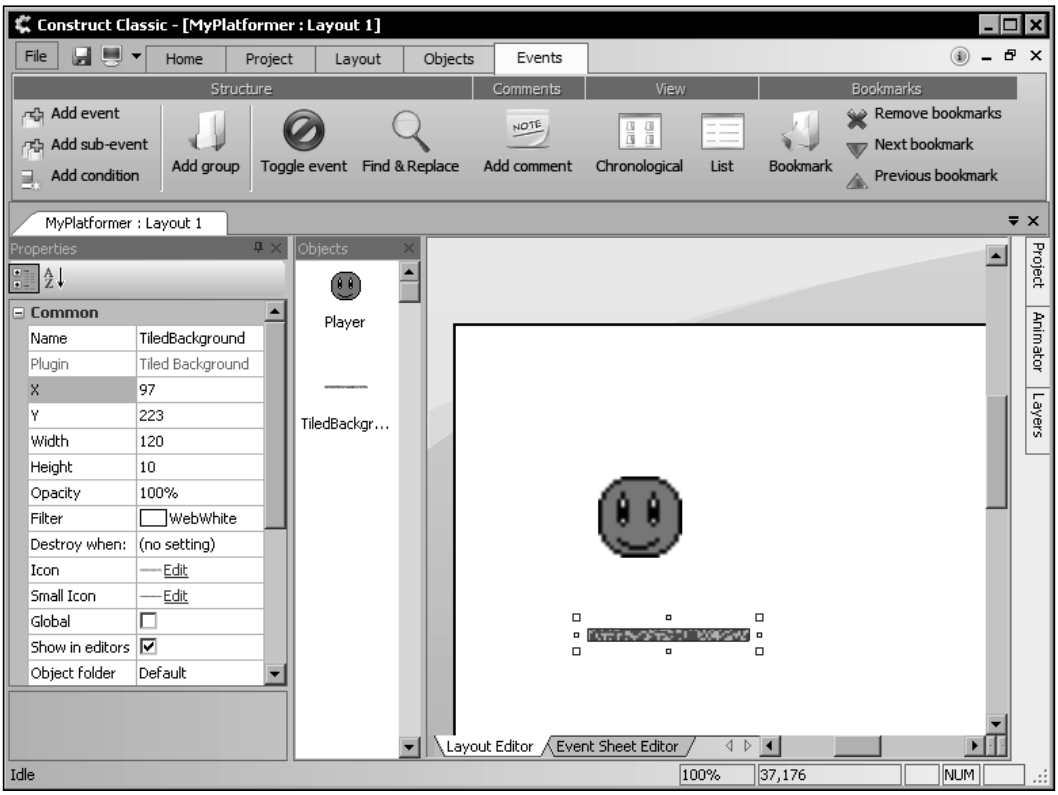
Time for action – make some tiled backgrounds

We are now going to make two tiled backgrounds: one for the grass that our player walks along, and the other for the dirt underneath.

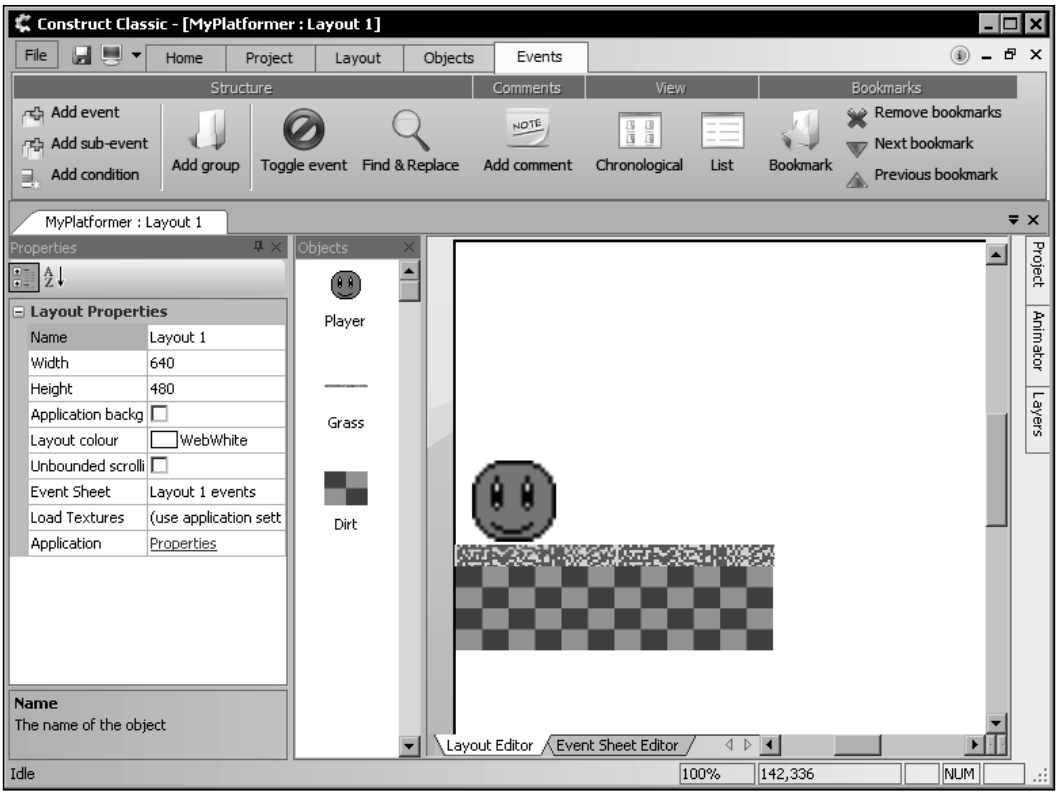
1. First, we'll create the grass background object. Once again, open the **Insert object** box. Instead of a sprite, insert a **Tiled Background** object from the group **Game**.



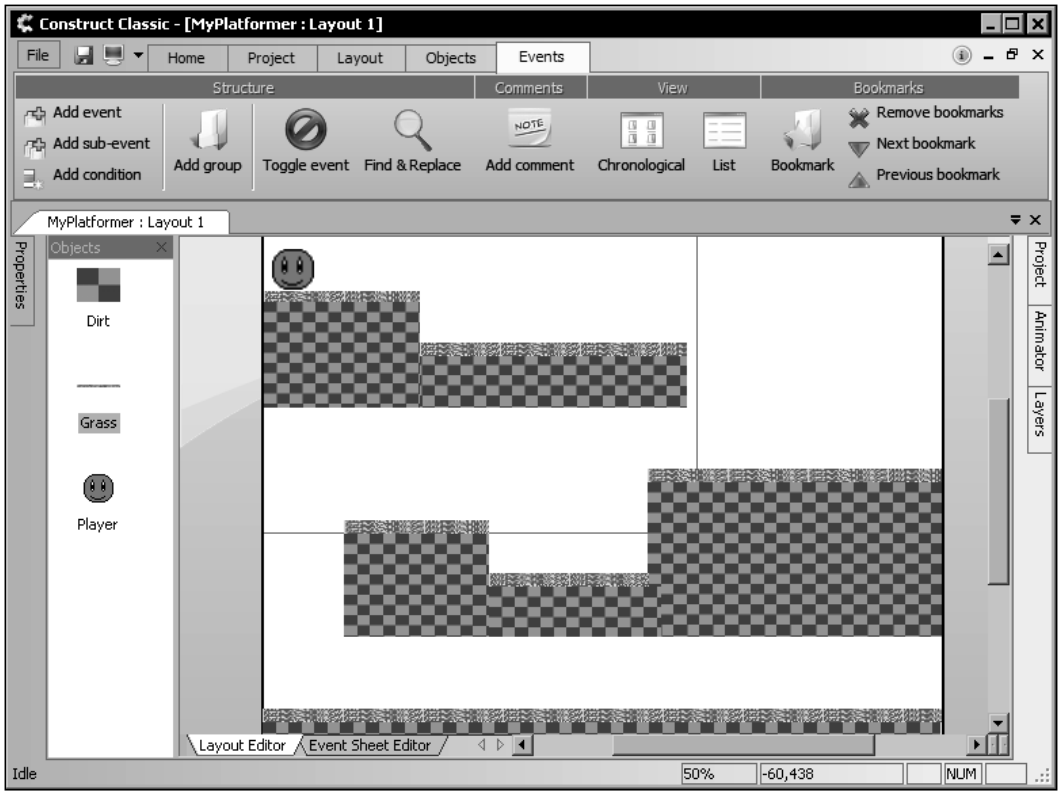
2. Now draw the image for your grass. It does not need a hotspot, as the hotspot of a tiled background is always in the top-left corner.



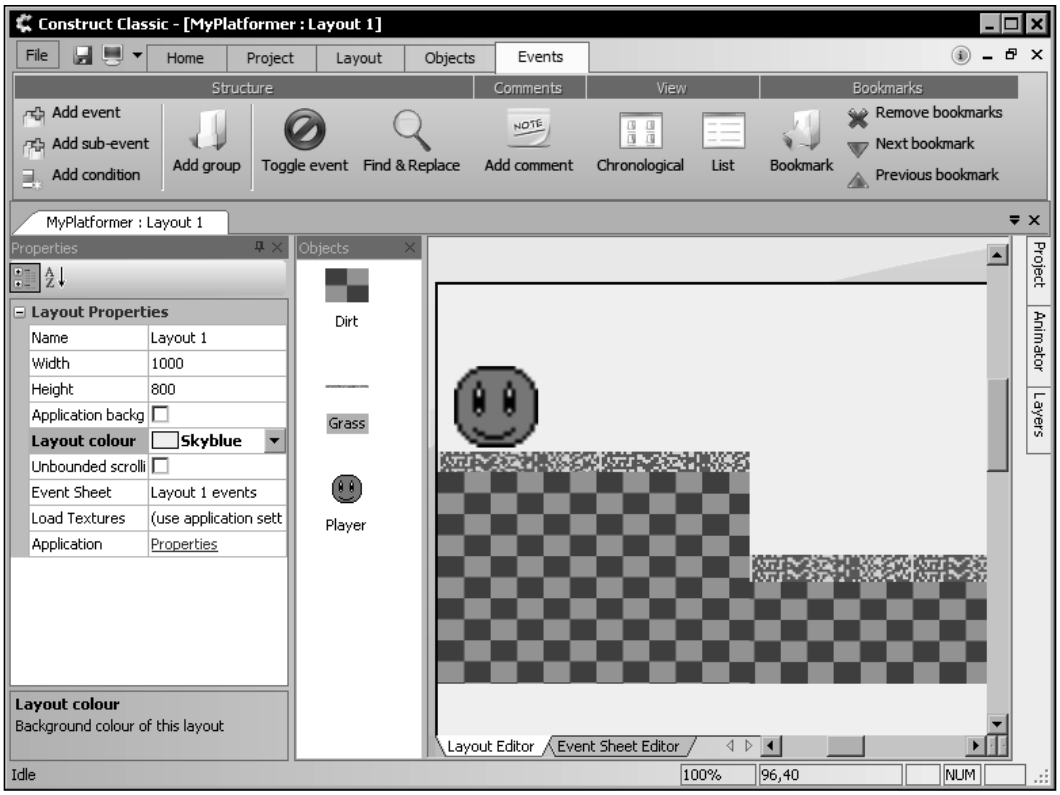
3. Now name it `Grass`, and move on to making another one called `Dirt`. Position them to form an initial platform underneath the player, as in the following screenshot:



4. Change the **Width** of the layout to `1000` and the **Height** of the layout to `800`. Then, make the entire level by copy-and-pasting (or dragging from the **Objects** bar) the grass and dirt objects around the layout, as shown in the following screenshot. You can use *Ctrl* and the mouse scroll wheel to zoom in and out of the layout.



5. Finally, set the layout background color to a shade of blue for the sky. Do this by clicking on a blank space of the layout, using the **Layout colour** box in the left properties menu, as shown in the following screenshot:



What just happened?

We have now learned how to place tiled backgrounds and make maps from them; notice how they all count as the same object. We have also learned how to change the background color of a layout.

Have a go hero – another tiled background

Using your newfound knowledge, try adding some more tiled background pieces to the map such as blades of grass or clouds in the sky. You can also go on to add background scenery. However, for non-repeating scenery such as rocks, it would be better to just make sprites.

Attributes: telling Construct more about our objects

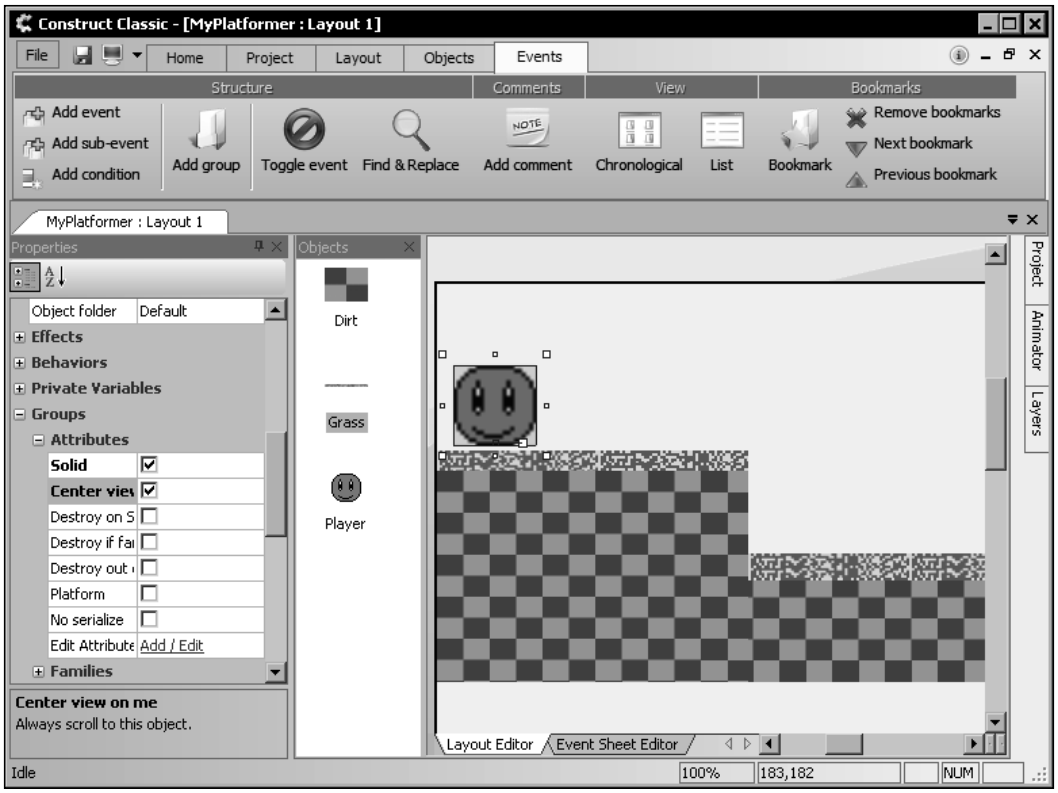
We are now going to learn about attributes. These can be thought of as a list of features about objects. An example of this is how apples, bananas, and oranges are all fruits.

In Construct, attributes can be custom-made, but there are also some built-in ones with special uses we'll be looking at.

Time for action – adding attributes to our objects

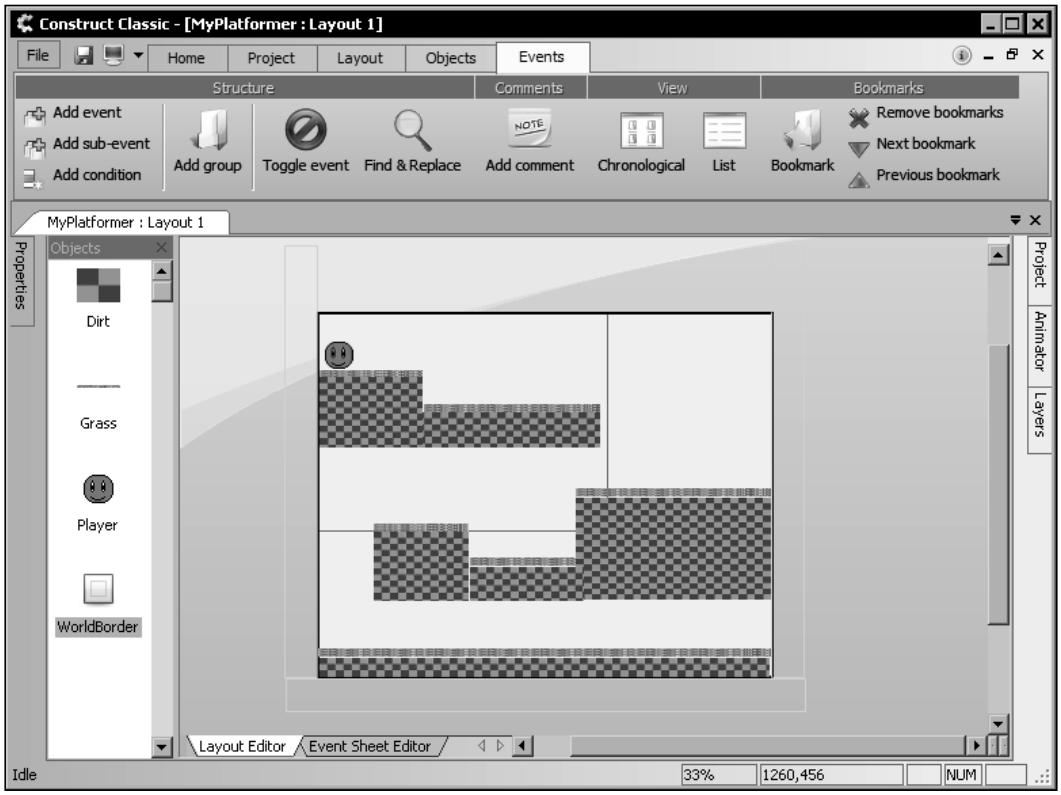
We are now going to give each of our objects the **Solid** attribute so that they can collide with each other. We will also give the player the **Center view on me** attribute so that the screen scrolls with the player.

1. Click on the player sprite and expand the **Groups** menu. Then expand the **Attributes** submenu. We can now tick the two boxes **Solid** and **Centre view on me**. The end result is shown in the following screenshot:

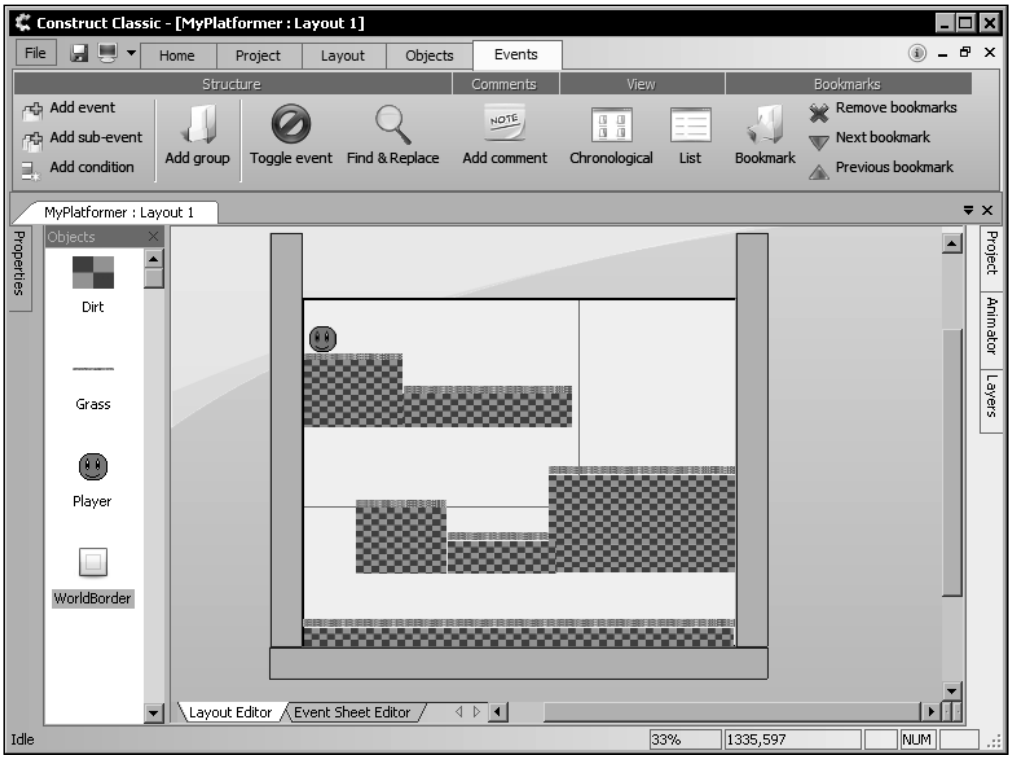


2. Now move on to give the **Solid** attribute to the grass and dirt tiled backgrounds.

3. To stop the player leaving the level, create a new tiled background called `WorldBorder` and place three of them: one for the bottom and two for the sides of the layout, as in the following screenshot. Tick **Solid** for these as well.



4. Finally, scroll down to the **Properties** menu (outside of **Groups**), and untick **Transparent fill** for the `WorldBorder` object and choose **WebGreen** as the **Fill** color. Notice in the following screenshot, the world border on the sides stretches above the level; this is to prevent the player from jumping over the wall.



What just happened?

We've set up the attributes of our objects. Now, we can take a closer look at the other attributes in the list.

- `Destroy on Startup`: This attribute automatically destroys the object when the game is run. This is useful when the object is created later (for example, bullets from a gun do not *exist* until fired).
- `Destroy if far`: This attribute destroys the object if it is far from the layout.
- `Destroy out of screen`: This attribute destroys the object if it isn't in view.
- `Platform`: This attribute allows the object to be stood on, or jumped onto, from underneath.
- `No serialize`: This attribute prevents the object from being saved and loaded when using Construct's built-in *save* feature.

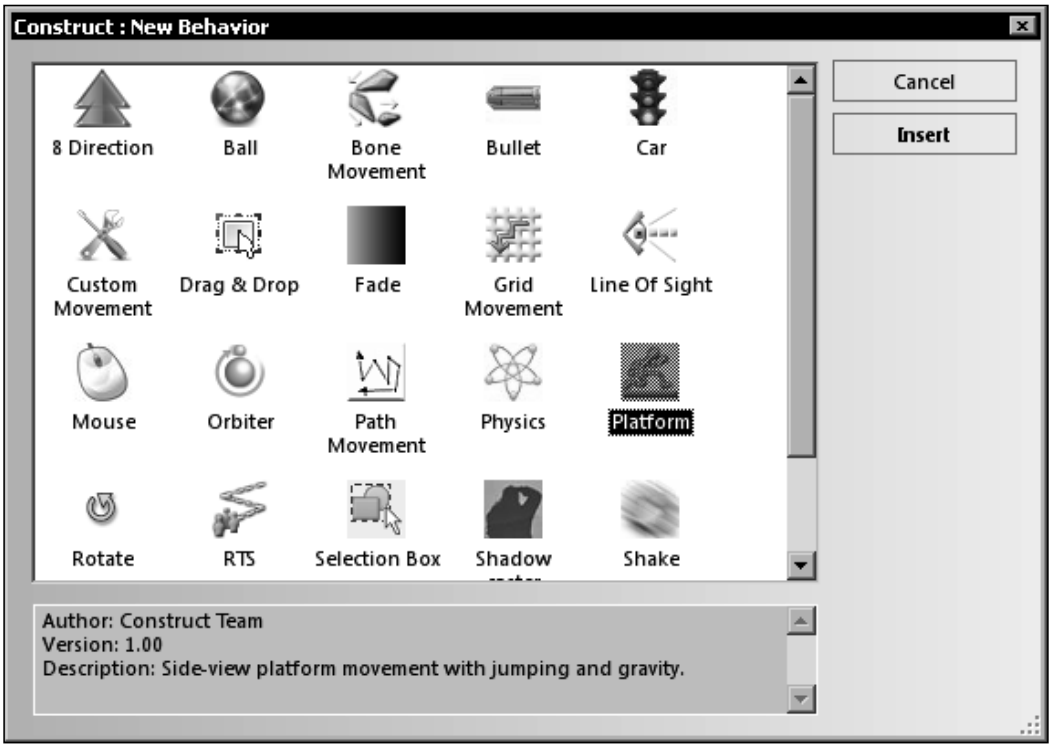
Behaviors: teaching objects how to act

In this segment, we'll be looking at the platform behavior and how to tweak it. Behaviors are pre-made movements and actions that an object performs in certain situations. They all have varying levels of customization, but when used correctly, can speed up game development tenfold.

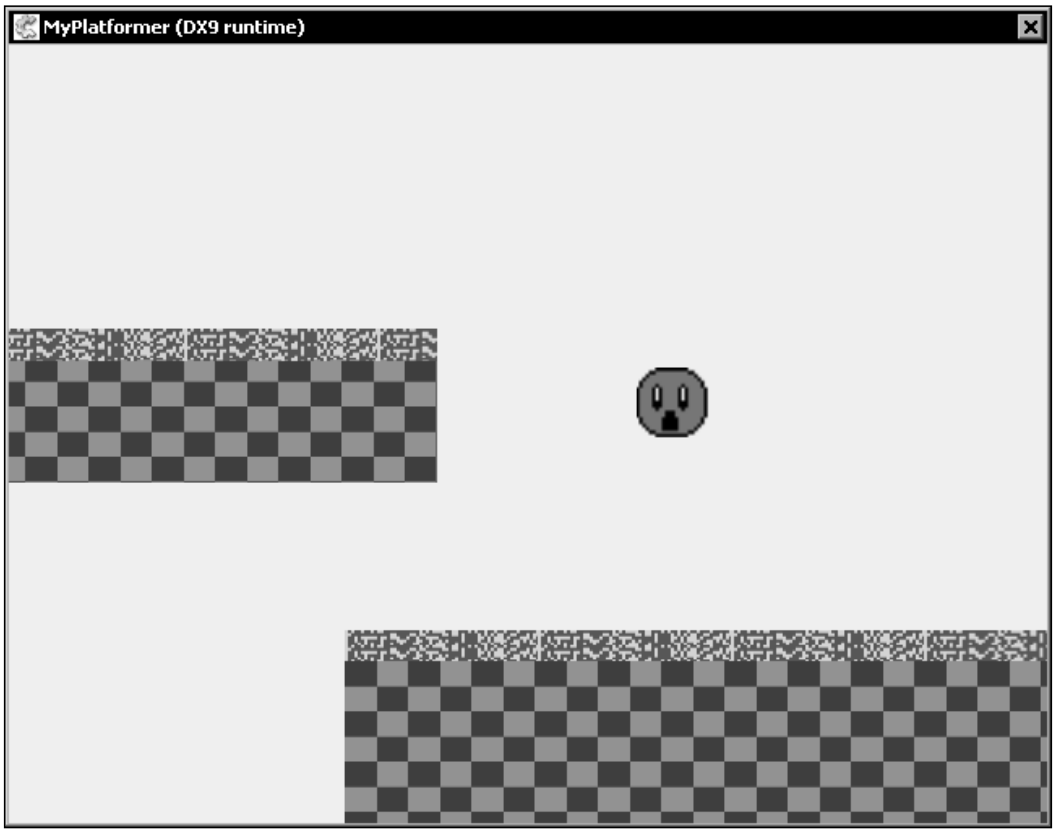
Time for action – getting our player moving

Now we can give our player the platform behavior and try it out when we run the game. This is the first step in making the game interactive.

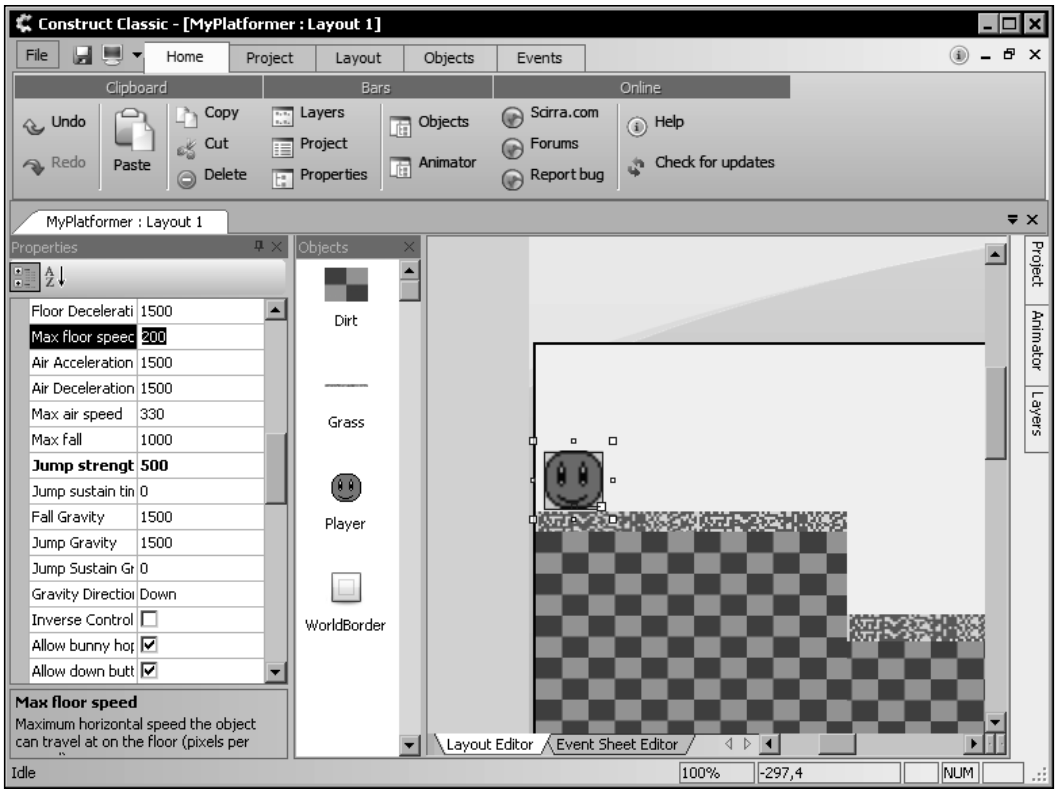
1. Select the player sprite, and scroll down the left-side menu to the **Behaviors** group. Then, click on **Add** in the **New Behavior** box. We now see the following screenshot:



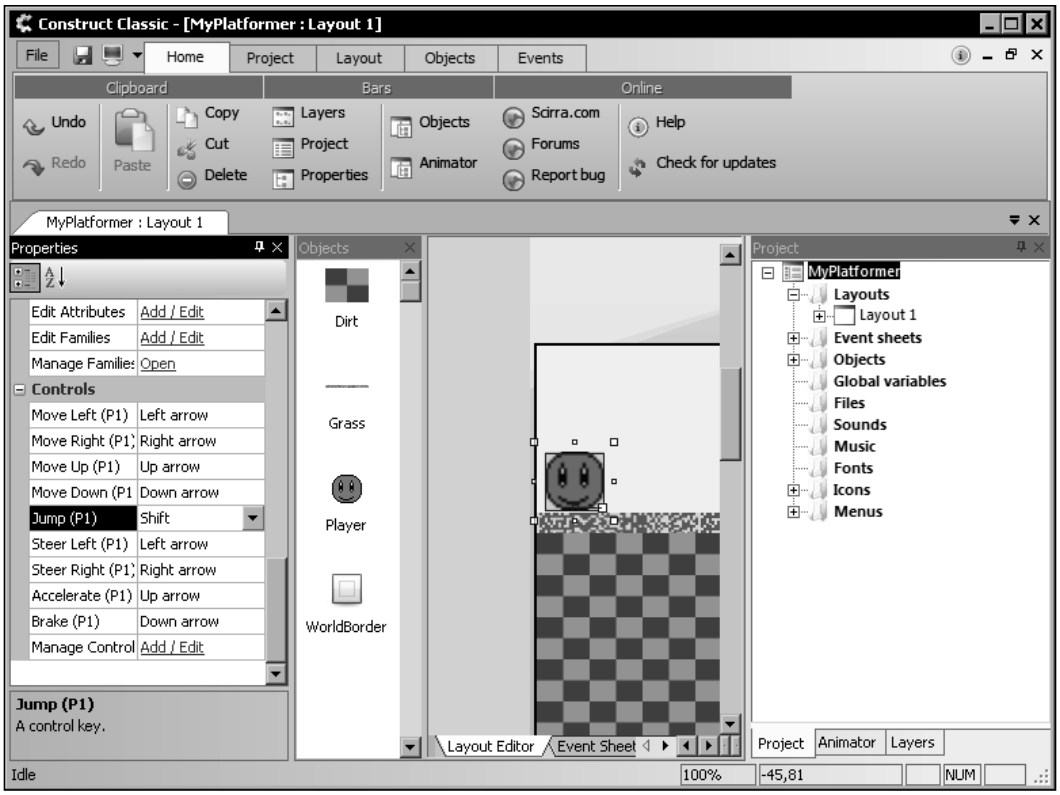
2. Our player object now has the ability to move around using the arrow keys and jump using the *Shift* key. Run the game to see this in action.



3. Now we can look at the settings for the behavior in the left screen. Change the **Jump Strength** and **Max floor speed** values, as in the following screenshot, until they control how you would want them to:



4. Finally, if you'd like to change the controls of the game, bring up the game properties menu and scroll down to the **Controls** menu. Then use the drop-down boxes to choose keys for each movement, as in the following screenshot:



What just happened?

Our character can now move around the level at the player's wish. Let's take a look at some of the other things we caught a glimpse of.

The behaviors

We saw a large number of behaviors in the insertion box. Objects can have any number of behaviors attached to them, which in turn can be deactivated and reactivated at will. Here is a list of notable behaviors that come with Construct; more can be found on the user forums online.

- **8 Direction**: The player can travel both vertically and horizontally, usually used for top-down games.
- **Ball**: A randomly moving ball movement that can be used to produce bouncing balls, similar to the Pong game.
- **Bullet**: Gives the object (bullet) controls, such as distance and speed of fire (or instant, for realistic bullets).
- **Car**: A top-down car movement.
- **Custom Movement**: Provides the advanced settings needed to create nearly any movement type while still using Construct's built-in collision methods.
- **Grid Movement**: Forces the object to move in a grid-like fashion.
- **Physics**: Allows the object to behave with realistic motion. Objects with this behavior only collide with other objects that have it.
- **RTS**: Allows the object to be moved around obstacles using a pathfinding algorithm. As the name suggests, it is designed for **Real Time Strategy** (**RTS**) games.

Although these are only a few of them, the explanation of these and other behaviors can also be found in their descriptions.

Setting controls

It is worth noting that custom controls can be added and removed from the previous list. This is useful as controls can later be referenced by name in developing the game, but changed from a single point.

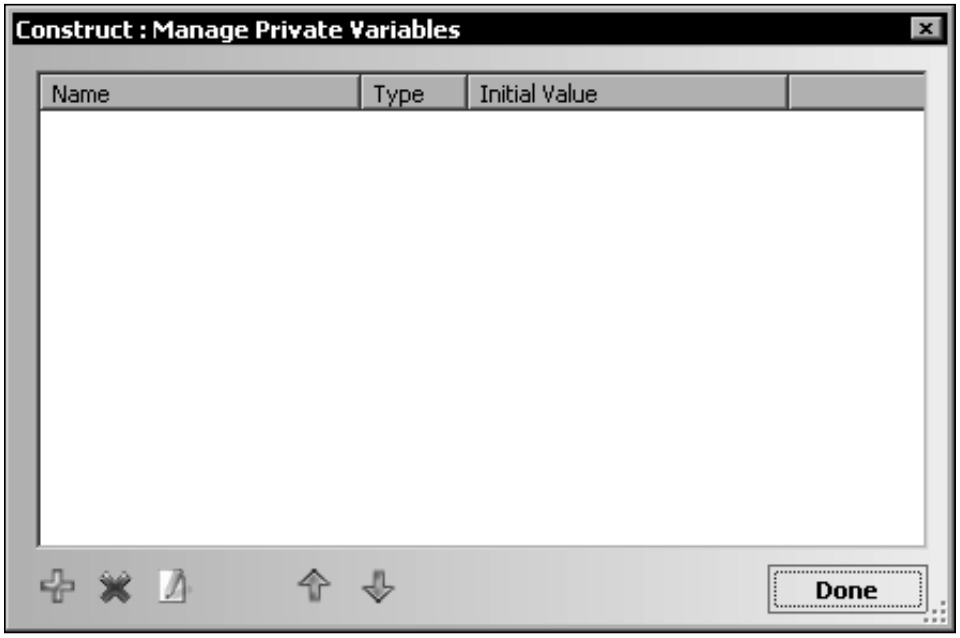
Variables: private and global

A variable is used to store a value that can change. These are useful in every game for keeping score and tracking the player's lives or health. In Construct, a private variable is stored by an object itself and can be different for each copy of that object. Meanwhile, a global variable is stored between layouts throughout the program.

Time for action – giving our player a life

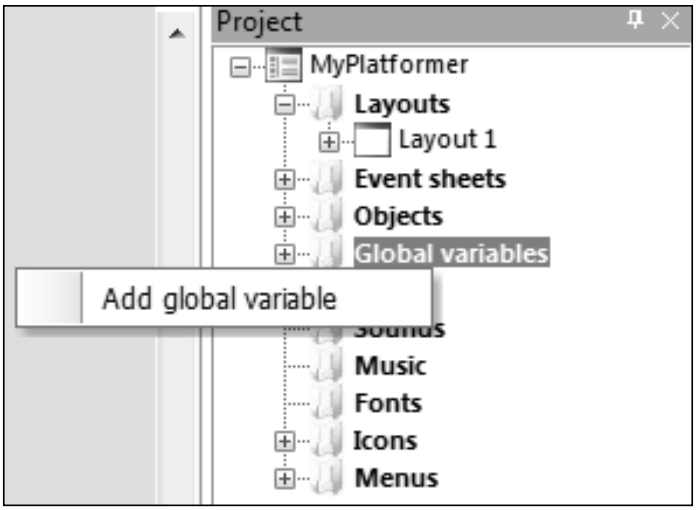
In video games, a player gets a *Game Over* when the character they control runs out of lives. Right now, our player doesn't have that luxury, so let's change that. In our game, the player will have a `Score` private variable to store how many enemies they stomped in that life, while a `Lives` global variable stores how many lives they have left before they lose.

1. Open the left menu for our player again. Scroll down to the **Private Variables** group, and click on **Add/Edit** to get a window similar to the following screenshot:

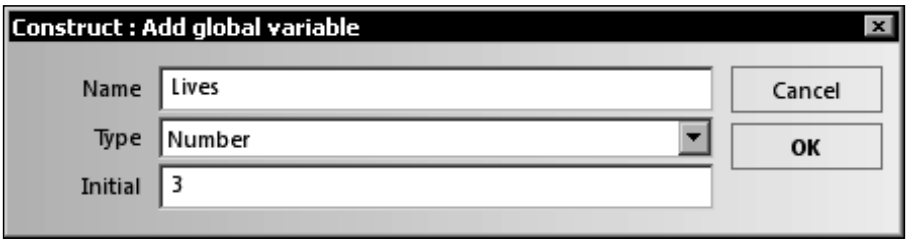


2. Now click on the plus button and create a number named `Score`. Leave the value of it at `0` and click on **OK**. Then click on **Done** in the private variable manager screen.

3. To finish, make a global variable `Lives`. To do so, open the right menu, and in the project tab, right-click on the item **Global variables** and click on the context item **Add global variable**, as in the following screenshot, when it appears:



Set the value to **3** and click on **OK**, as shown in the following screenshot:



What just happened?

We have now created a private variable to store the score of the player's character for its current life and a global variable to store the number of lives they have left. Private variables are usually used to store values such as health and ammo, but can be used in situations like this where the *combo* score is more important than the total score.

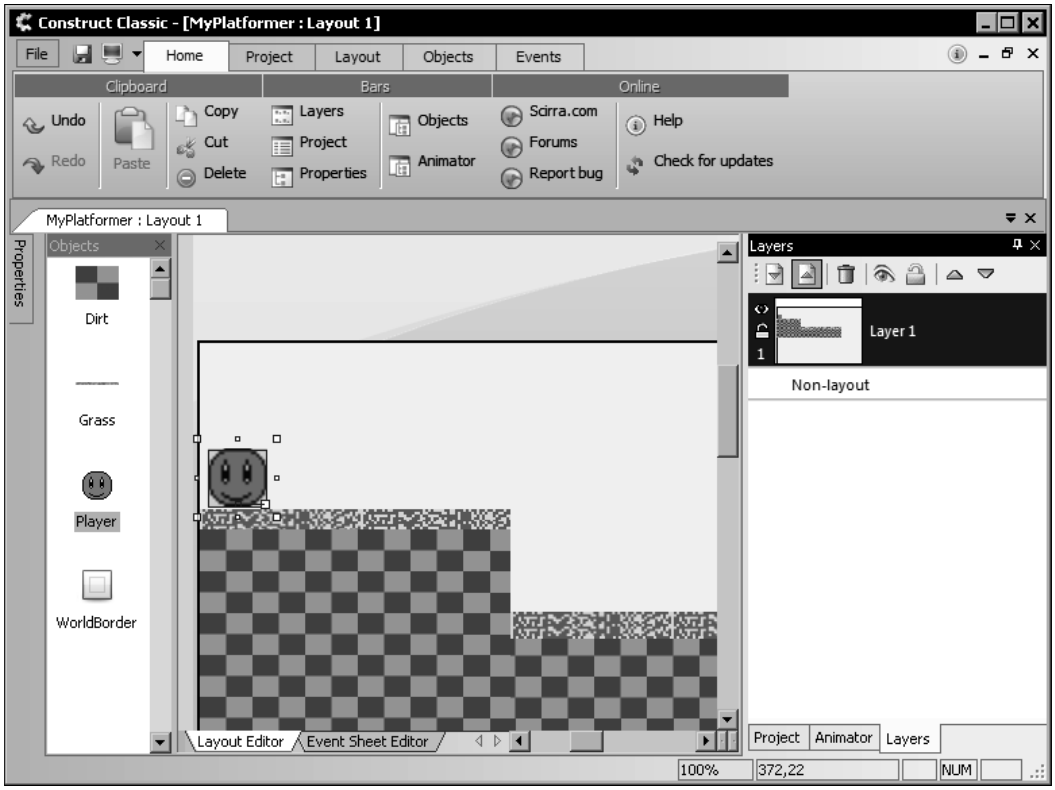
Textboxes: giving the player a heads-up

Textboxes are used to write information to the player. They can tell the game story, or tell the player how they are doing. They are important in most games as they can be used to teach the player how to play, as well as guide them through the levels.

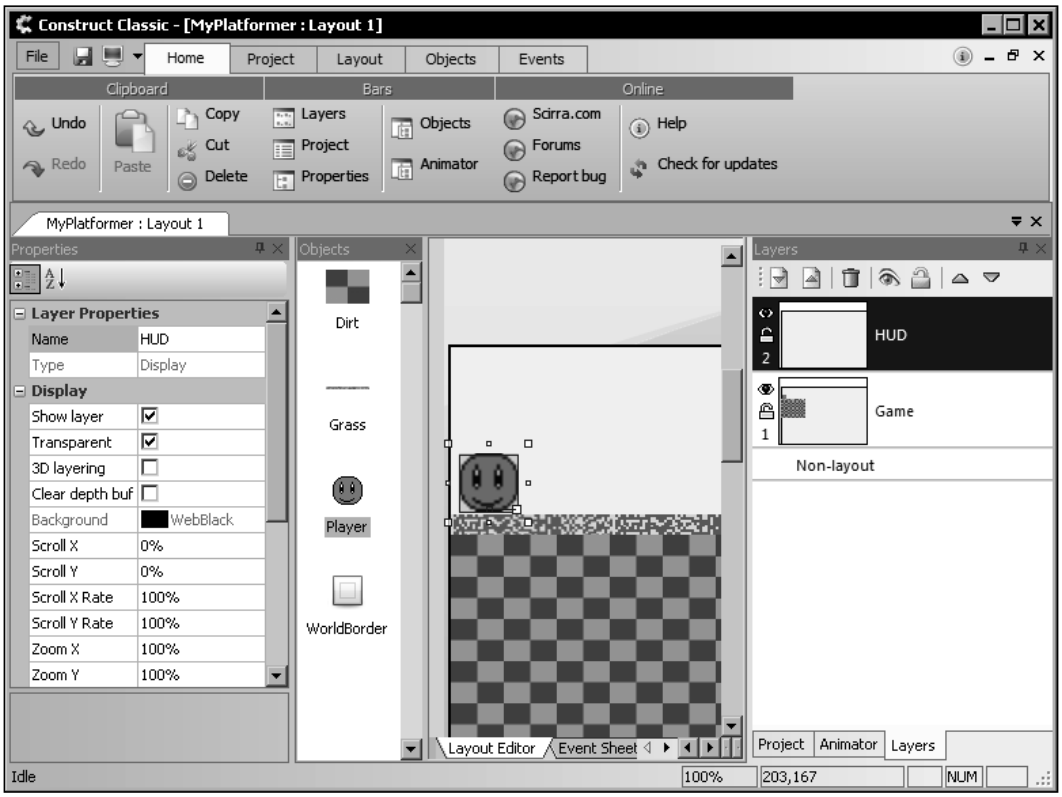
Time for action – showing our player their health and score

We are now going to make two textboxes: one is to tell the player the score, and the other is to tell the player how many lives the character they are controlling has left. We'll also learn how to keep these textboxes on the screen with the player.

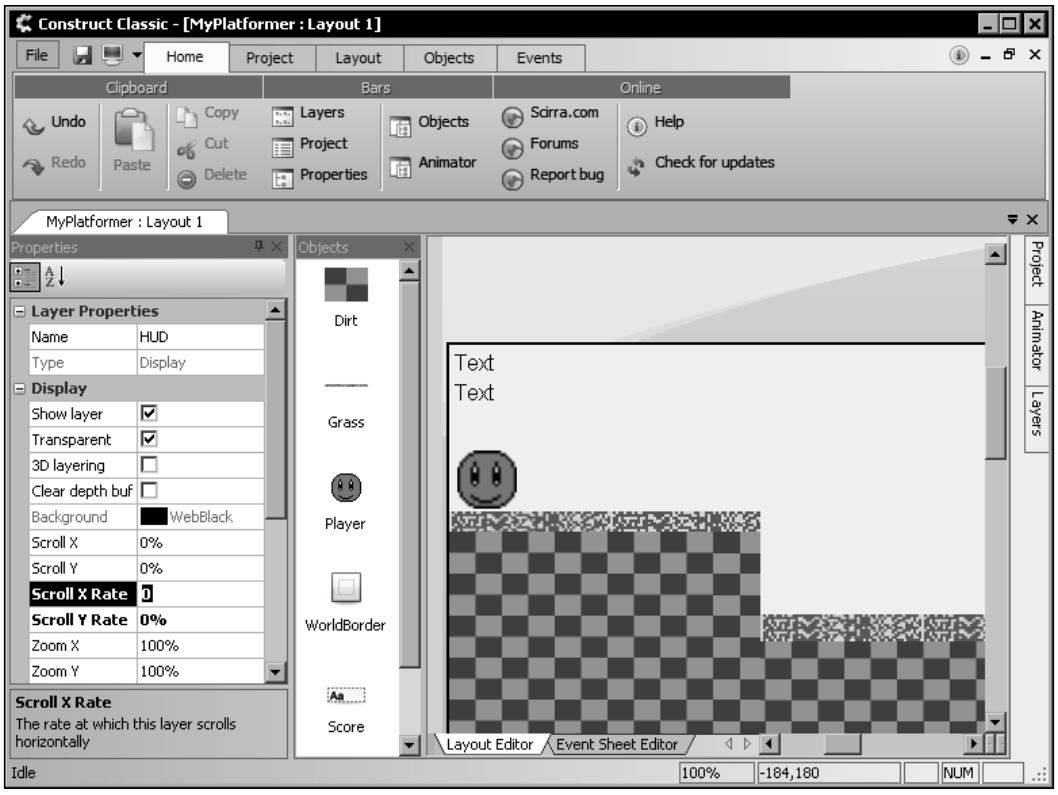
1. First, we will need to add a new layer to the layout. To do this, open the **Layers** tab on the right-hand menu, and click on the upward pointing paper button to make a new top layer.



2. Layers can be modified similar to objects. Open the left menu and rename the text layer to `HUD` (Heads Up Display), and the other layer to `Game`.



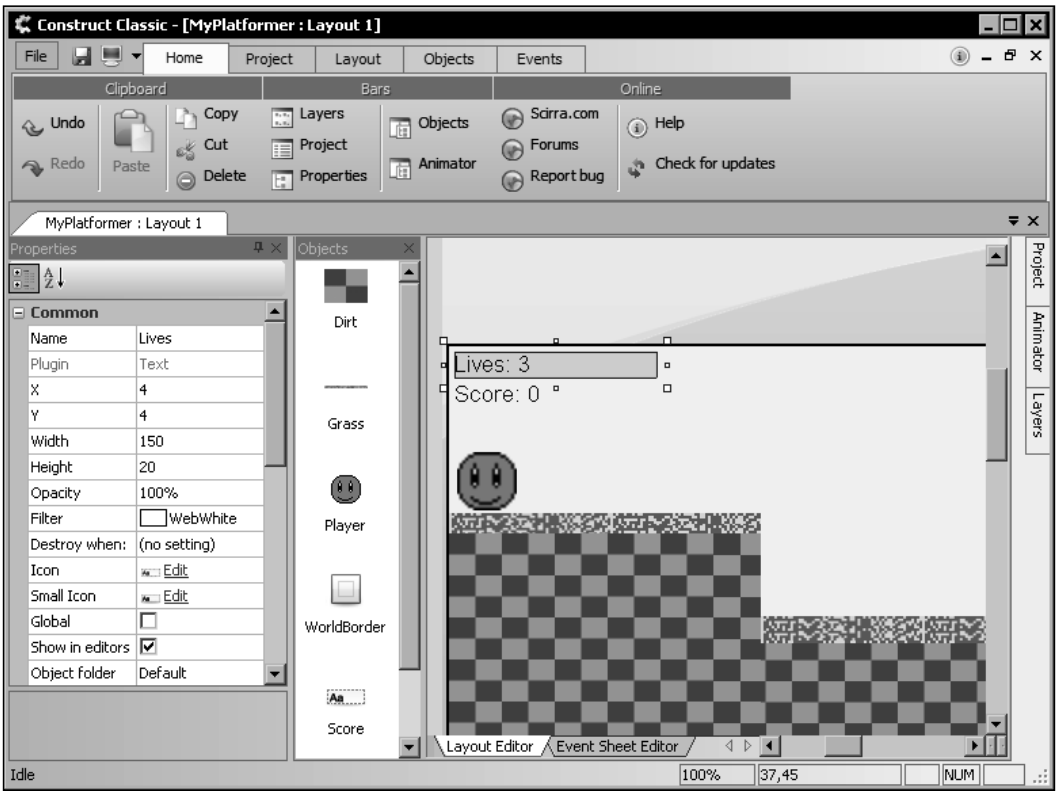
3. In the layer properties for HUD, change the `Scroll X Rate` and `Scroll Y Rate` percentages to `0`. This keeps all objects on the HUD layer in the same position on the screen while the game is running.



4. Now, make sure the HUD layer is highlighted, as shown in step 2. This means each object we make is created on that layer. Insert two new `Text` objects to the layout (found under the `Graphics` group).

5. We can now change the name of the first text object to `"Score"` and the second to `"Lives"`.

6. Finally, scroll down in the properties window for each textbox. Change the text of Score to "Score: 0" and the text of Lives to "Lives: 3".



What just happened?

We now have some textboxes in our game to show the lives and score count for the player's character. We'll be linking them to the variables next through Events.

Events: setting the rules and goals of a game

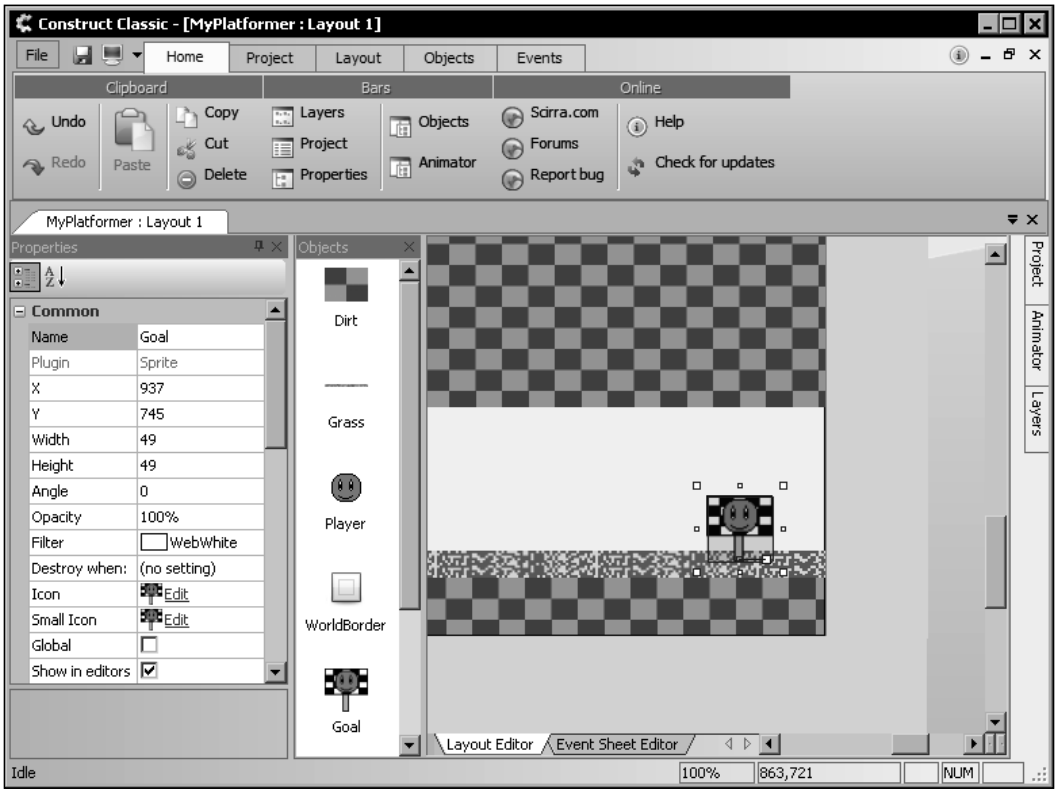
We are now going to learn a little of the biggest part of game-making in Construct. Events are used to define the very logic of a game and play an integral role in complex games.

For now, we'll be learning how to use them in this game to keep their lives and score updated.

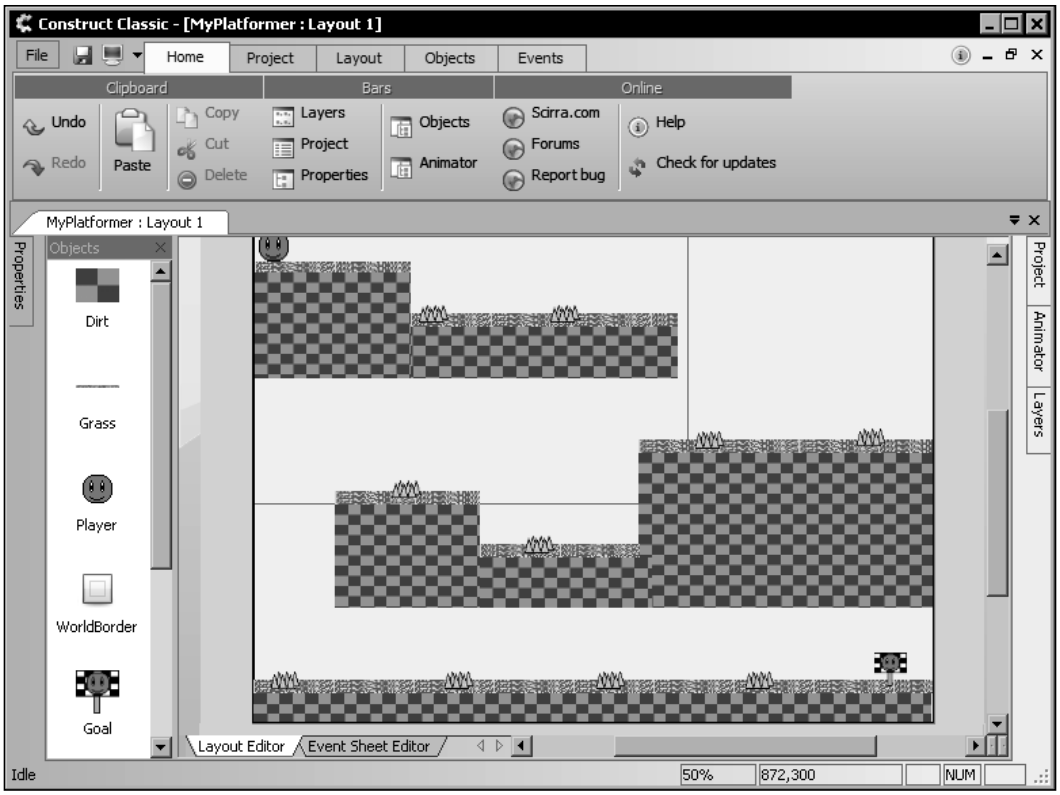
Time for action – very eventful games

When the player's character touches deadly spikes, we know they should lose health or die, but how does the game know this? That's where we are heading. First, we need to learn how to create events in the Event Sheet Editor.

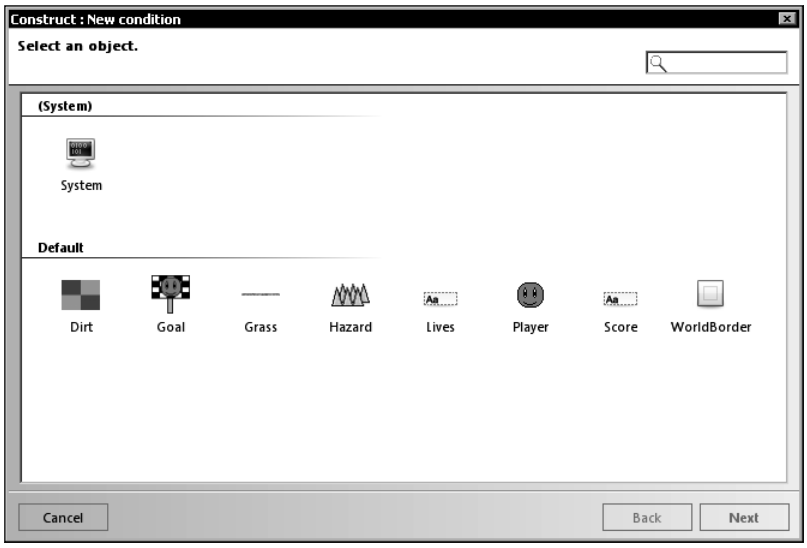
1. Add a new sprite to the Game layer, which will be the end goal, and place it at the end of your level. Name it `Goal`, and give it the `Bounding box` collisions mode.



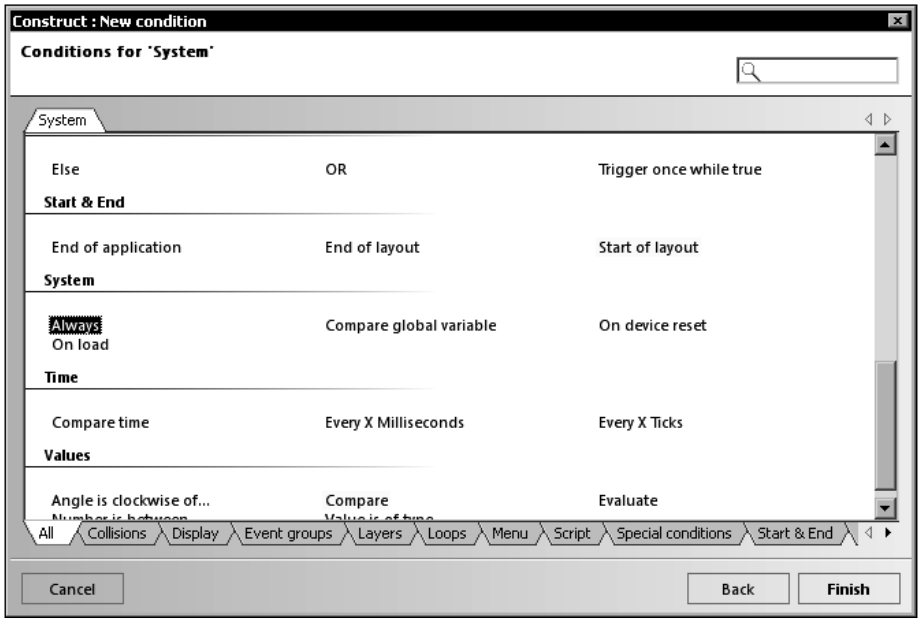
2. Now put some sprites around your level named `Hazard`. They can be lava, saw blades, spikes, or any other contraption you can think of. However, this time they will keep `Per Pixel` collisions mode.



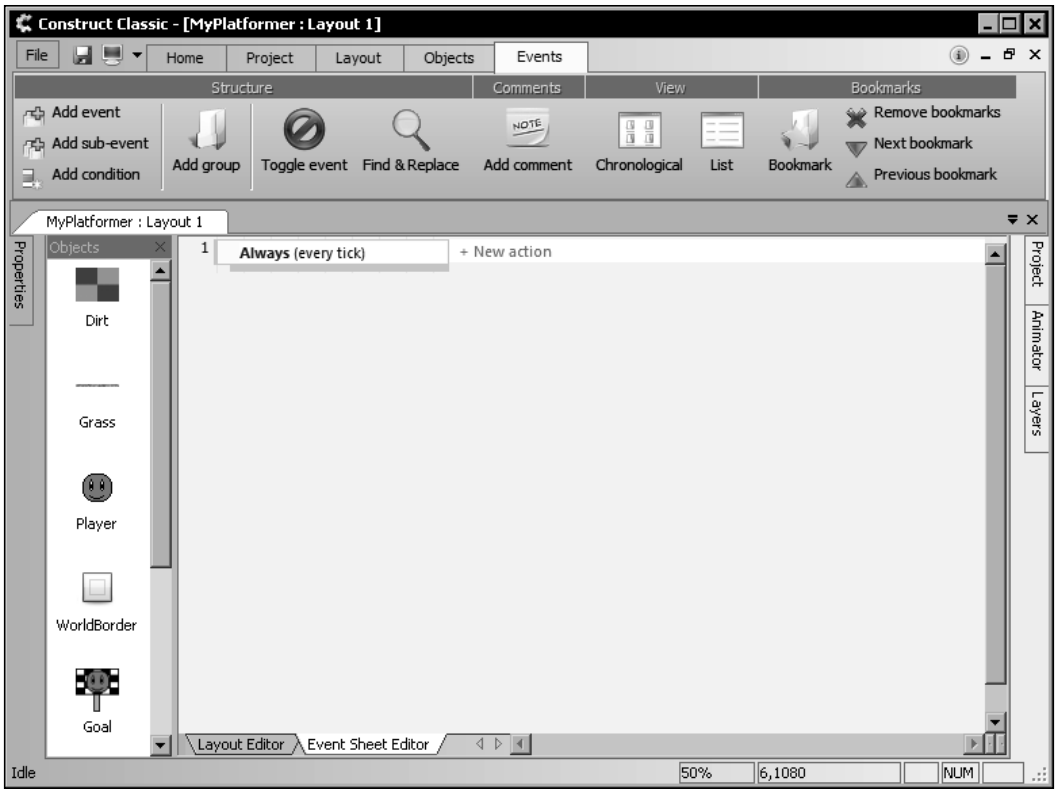
3. Now we are ready to switch to the **Event Sheet Editor** tab. Right-click while in the **Event Sheet Editor**, and click on **Insert event** in the context menu that appears.



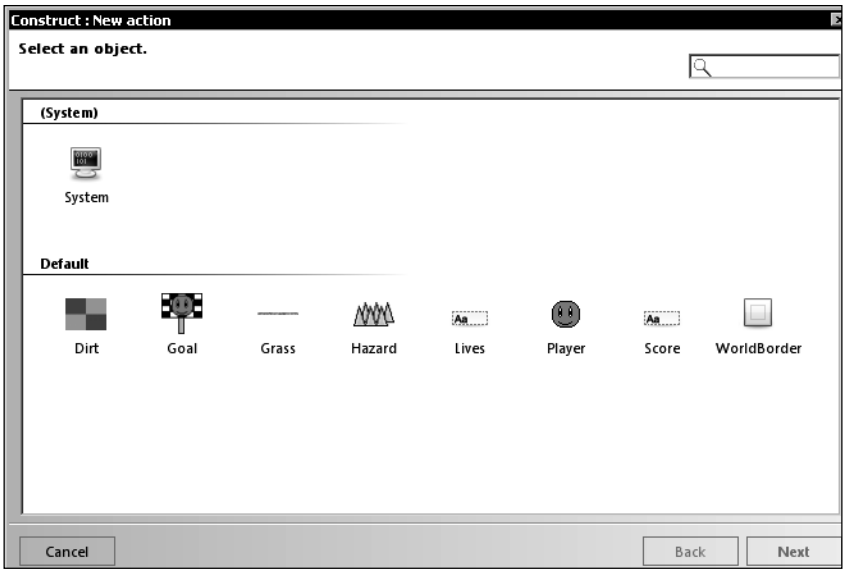
4. Each object can have its own conditions, but for now we are going to create an `Always` condition from the menu opened by double-clicking on the `System` object. The `System` object is included in every game.



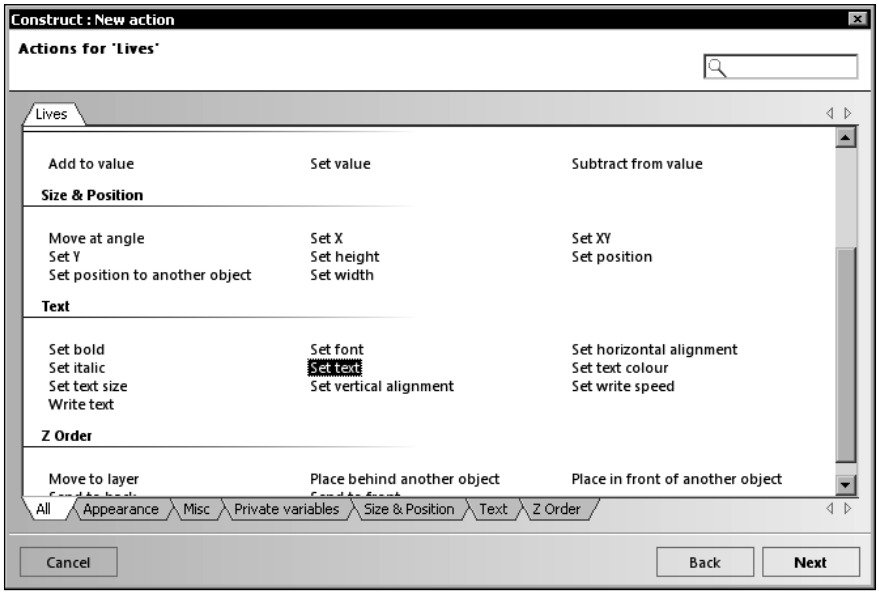
5. We now have an `Always` condition. Conditions affect when events trigger and start the actions on the right. Currently, there are no actions triggered by the `Always` condition.



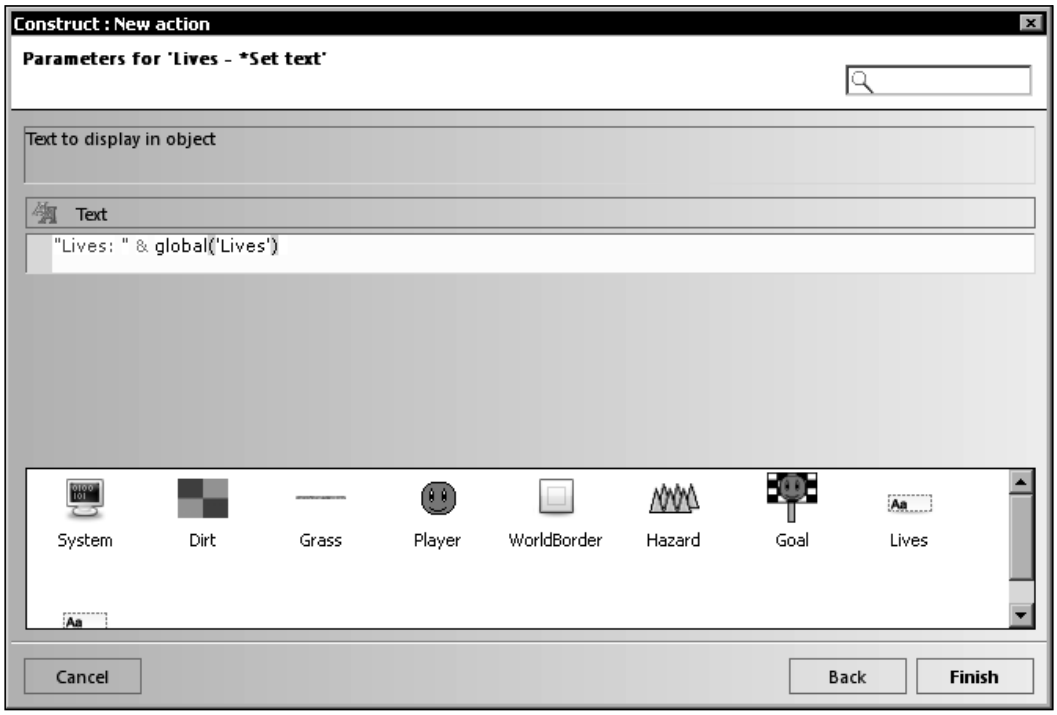
6. Click on `New action` text to bring up a menu very similar to the condition choice screen. However, this menu is used to choose what happens when the conditions of the event are all true (as an event can have more than one condition).



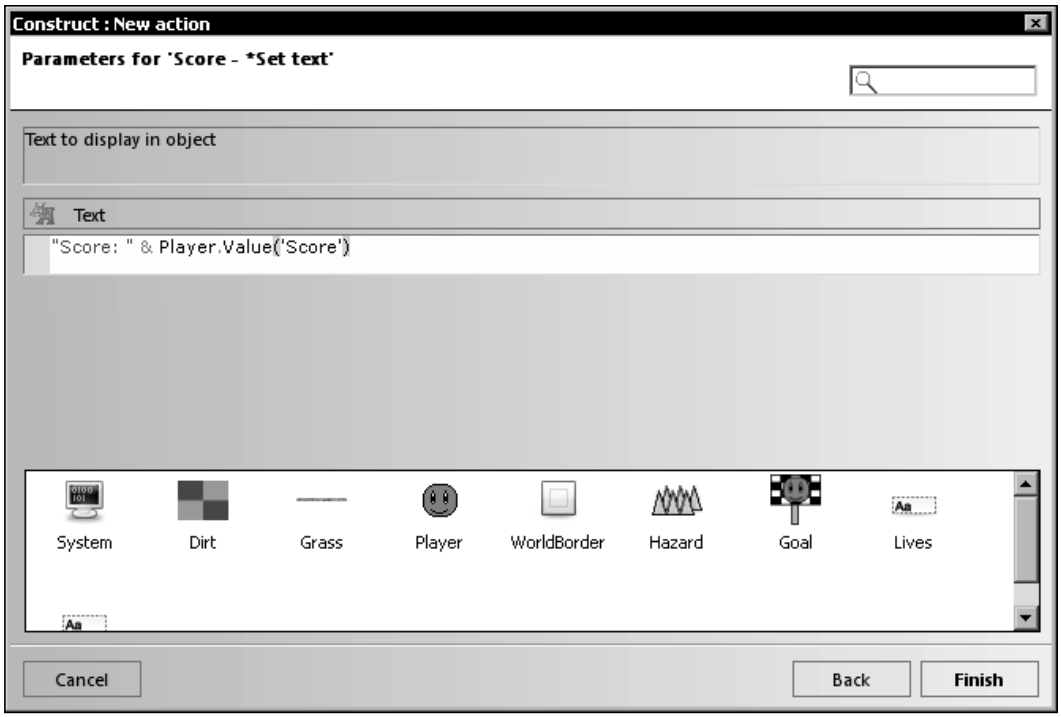
7. Now double-click on the **Lives** textbox to get to its actions list. Choose the action **Set Text**.



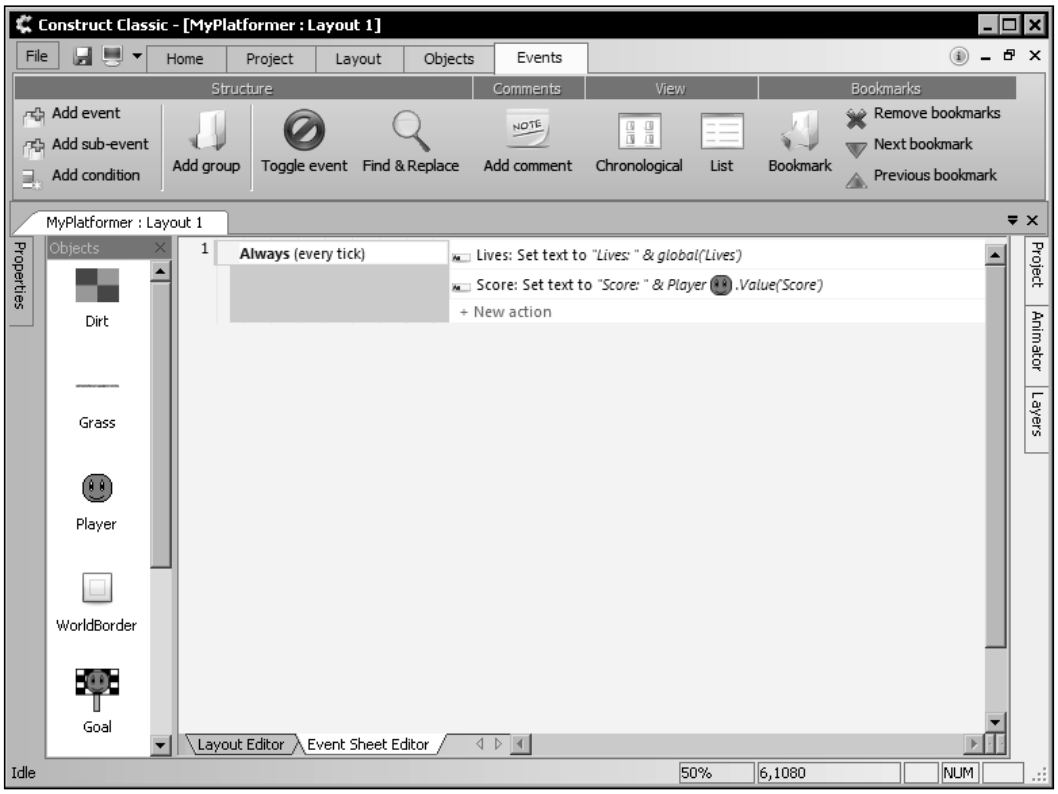
8. We are now shown an entry screen, which is expecting us to enter the text we want to display. For displaying our text, we will start with `"Lives: "`, and then display the global variable, which is accessed by typing `global('Lives')`. This makes the final string `"Lives: " & global('Lives')` (as the & symbol joins the two together).



9. We now have the Lives textbox updated. Repeat the steps previous, but enter `"Score: " & Player.Value('Score')` in the textbox. This is how private variables are addressed.



10. We now have our event to update both textboxes. Try changing the values in the editor to show they work.



What just happened?

Events are the backbone of games made in Construct and as such will take a bit longer than one chapter to learn. Luckily, we'll be learning them all throughout the book as we can't make a full game without them. Let's see what we've learned so far.

The sprites

We made the goal and hazard sprites so that we have them ready for when we make our events to control what happens when the player touches them. The goal would make the player win the level, while the hazard would kill them and take away a precious life. For now, however, they are merely graphical objects.

Events

Events are a container of conditions and actions. When the conditions are true, the actions are triggered. Any number of conditions and actions can be contained in them.

Conditions

As mentioned before, conditions are what cause the event actions to be performed. When all conditions in an event are true, the event will trigger. We have now met the `Always` condition, which will trigger always (or every tick, which is the smallest amount of time between cycles of events). Many conditions and actions will be named, or have descriptions, to help understand of their purpose. It is worth noting that an event with no conditions is treated as having an `Always` condition.

Actions

Actions are used to cause changes, whether value, graphical, or both. They can only be performed when the event is triggered.

Summary

Now that we have started learning how to make platform games, we've covered a lot of the basics needed to make one with Construct.

We started this chapter with animated sprites, which are the most important object in any graphical games made with Construct Classic. Then we learned about tiled backgrounds, which decrease the difficulty in making large maps. We then used some attributes to define which objects are solid and learned that the Player object must have the camera centered on it at all times.

After that, we gave the Player object movement with the Platform behavior, a pre-made movement designed for 2D platformer games. Our Player object then gained a personal score counter and some lives to add some challenge to the game. We then went on to add some textboxes that show the current score and lives, and then finished off the chapter with learning how to create events, the building blocks of any new gameplay elements we want to make.

Now that we've learned to do all this, we're ready to add some bad guys and make the hazards hazardous, which is the topic of the next chapter.

Where to buy this book

You can buy Construct Game Development Beginner's Guide from the Packt Publishing website: `http://www.packtpub.com/scirra-construct-game-development-beginners-guide/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.packtpub.com/scirra-construct-game-development-beginners-guide/book