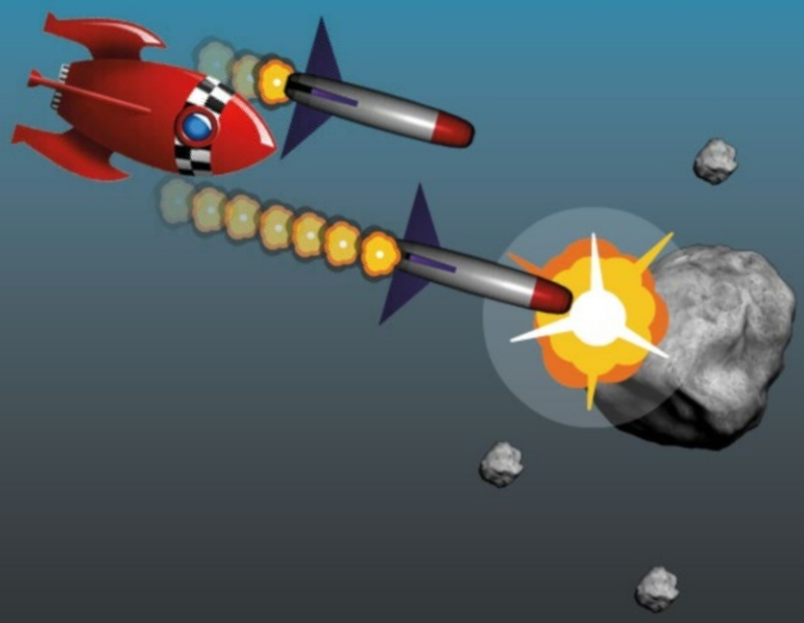

Practical GameMaker: Studio

Language Projects

Ben Tyers

Apress®

Practical GameMaker: Studio

Language Projects

Ben Tyers

Apress®

Practical GameMaker: Studio

Ben Tyers
Worthing, West Sussex, United Kingdom

ISBN-13 (pbk): 978-1-4842-2372-7 ISBN-13 (electronic): 978-1-4842-2373-4
DOI 10.1007/978-1-4842-2373-4

Library of Congress Control Number: 2016962191

Managing Director: Welmoed Spahr
Lead Editor: Steve Anglin
Technical Reviewer: Dickson Law
Editorial Board: Steve Anglin, Pramila Balan, Laura Berendson, Aaron Black, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao, Gwenan Spearing
Coordinating Editor: Mark Powers
Copy Editor: Karen Jameson
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springer.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text are available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper

Contents at a Glance

Contents

About the Author

Ben Tyers is a freelance programmer and technical writer by day, and a sci-fi horror novel writer by night. He made his first computer game way back in 1984, on a ZX Spectrum 48K computer, when he was eight years old. His passion for creation has continued since then. He holds a number of computer-related qualifications. When relaxing, Ben has an infatuation for old-school horror and sci-fi films, particularly 1960s B-movies.

About the Technical Reviewer

Dickson Law is a GameMaker hobbyist, commentator, and extension developer with six years of community experience. In his spare time, he enjoys writing general-purpose libraries, tools, and articles covering basic techniques for GameMaker: Studio. As a web programmer by day, his main areas of interest include integration with server-side scripting and API design. He lives in Toronto, Canada.

Acknowledgments

Yellow Afterlife – Thanks for your help

Thanks to the following for your support:

Nathan Brown

Loukas Bozikis

Alesia Buonomo

Kehran Carr

Arik Chadima

Rom Haviv

Zachary Helm

Credit also to the following for permission to reuse their assets:

Kenney.nl – Playing card sprites

http://millionthvector.blogspot.de – Spaceship Sprites

Napoleon – Missile Sprite

New_Regime by Nuclear_Spring – Backing Music

JM.Atencia – Enemy Spaceship Sprite

Cover Art:

Phaelax – Asteroid

JM.Atencia – Enemy Spaceship

Napoleon – Missile Sprite

KennyLand – Explosion

The original version of this book was revised. An erratum to this book can be found at DOI 10.1007/978-1-4842-2373-4_41

Introduction

This book serves as an introduction to using GML (GameMaker Language) for creating games using the popular software, GameMaker: Studio. GameMaker: Studio is a software package created by YoYo Games that allows the creation of software for different platforms including Windows, HTML5, Android, iOS, and Mac OS X. The software allows for quick prototyping of games. Its IDE allows for the creation of games using its D&D (Drag & Drop) system, which allows the creation of games with minimal programming experience, and the more versatile scripting language, GML. It allows you to export to various platforms with only minor changes to code. According to their website, their software allows faster coding than native languages, and rapid prototyping.

Using this book you'll learn 24 programming elements that are important when creating a game. Each section includes an introduction to a new programming element, some examples, a worksheet with answer key, mini projects to apply your to new knowledge (with example GMZ project file for each), and after each element there is information on how this learned code will be applied in a final end of book game. After completing all sections, you will put into action what you have learned to create an arcade style shooting game.

There are then a number of assignments, from which you may choose, to create a final project. If you are teaching in schools you may include this as part of your students' coursework.

This book is suitable for home study or in a classroom.

GML code is provided in the following style:

```
GML code in this style.
Comments in this style.
```

Assets in the resources tree are in this style.

***Health, lives, score,* are in this style.**

Events are in this style. User interface elements (e.g., buttons) are also shown in this style.

For example:

```
draw_self(); //draws sprite assigned to this object
draw_set_font(font_hud); //set font
draw_text(100,100, "Hello World"); //draw text
```

This book assumes some basic knowledge of using GameMaker: Studio. This introduction covers the basics needed to attempt the exercises in this book. If you're using this book in a school, it's recommended that you either cover this first, or photocopy it and hand out before the first class.

By following this introduction you'll create a basic click-the-object game. You'll learn the basics of how to set up and use the following:

Rooms	Sounds	Sprites
Fonts	Objects	Drawing
GUI	Alarms	INI Files
Randomization	Create Events	Mouse Events
Step Events		

All the above will be introduced as you create a simple click-the-object game. There will be an object that appears at random positions and the aim of the game is to click it before the timer runs out. Each time you successfully click the object the timer will get faster. If you don't click the object before the time runs out, the game ends.

Resources

The resources for this book can be downloaded via the Download Source Code link at `http://www.apress.com/us/book/9781484223727`. These include all resources broken down by chapter, all GML in the book, and all resources and GML for the final game. Example answers for each project are also included. The resources for this introduction are in the folder: **Project Assets & GMZ Files ➤ Assets Used In Main Chapters ➤ Introduction.** There is a project file for this introduction.

Sprites

Sprites are images that will be used in your game. You'll use them to display the player, enemy, and other graphics in your game. They are in the downloadable resource folder: **Assets Used In Book ➤ Introduction**. Sprites will be drawn by referencing or assigning them to objects. Next, load in the sprites for this game. There are five of them, **`spr_logo`**, **`spr_start`**, **`spr_target`**, **`spr_exit`**, and **`spr_lives`**.

You can create a new sprite by clicking the **Create a sprite** button as shown in Figure i-1:

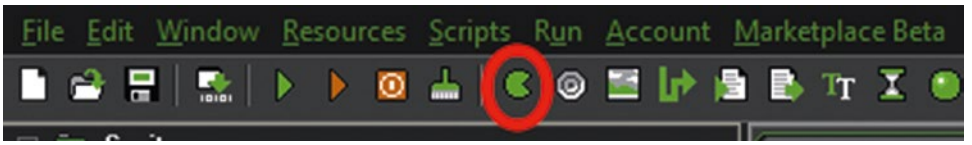

Figure i-1. *Create sprite button*

Name the sprite `spr_logo` and click `Load Sprite`, select the file shown below, then `Open`, set the `Origin` to the `Center` and click `OK`. This process is shown in Figure i-2:

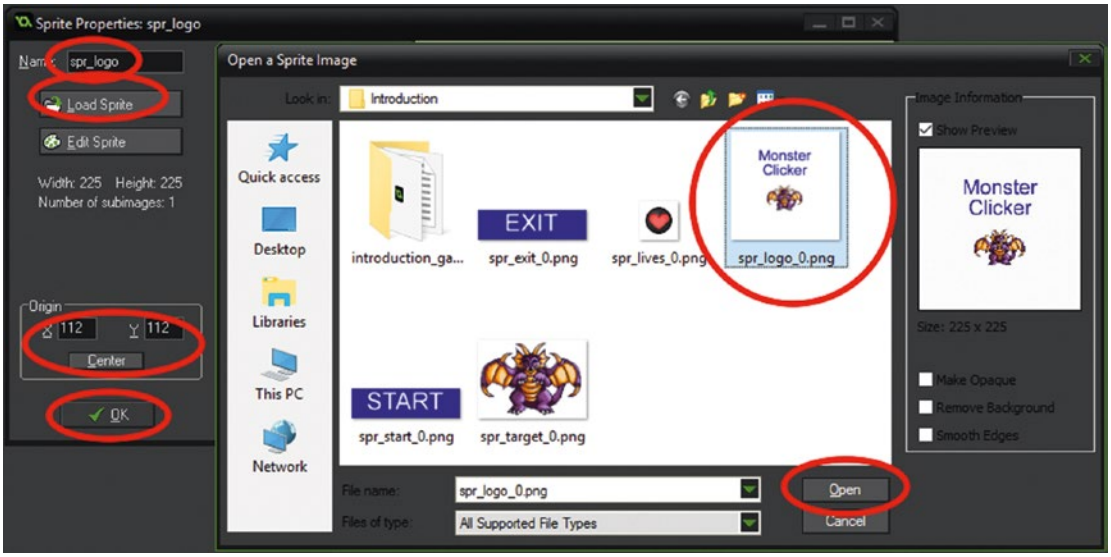

Figure i-2. *Naming and loading a sprite – step 1*

Repeat this for the remaining sprites, naming them **`spr_exit`**, **`spr_lives`**, **`spr_start`**, and **`spr_target`**. Set the origin of all sprites to center. The origin is point in the sprite where it will be positioned in the room. More information is provided in the sprite section. For example, using the sprite in Figure i-2, the origin is 112,112.

If you've followed along correctly so far, your resources tree will look like Figure i-3.

Figure i-3. *The sprite section will look like this*

Rooms

Note When starting out, ensure all rooms have the same size settings, for example a width of 800 and a height of 400. If rooms have different sizes then anything drawn may be distorted.

Rooms are where the action takes place and where you put your objects. You'll use these objects to display graphics, process user input, play sounds, and make other things happen.

We will create two rooms. The first will be a splash screen that shows some information about the game, while the second room will be where the actual gameplay takes place. First create two rooms; name them **`room_menu`** and **`room_game`**. Set the room size for each as 800 by 400.

You can do this by clicking the **Create a room** button as shown circled in Figure i-4:

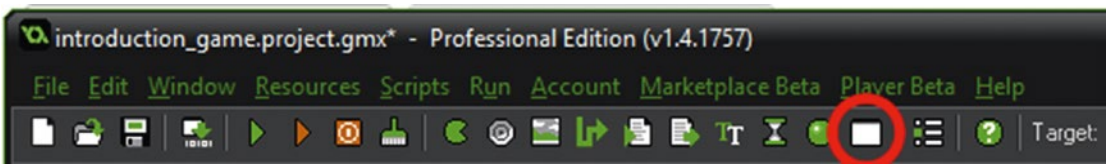

Figure i-4. *Create room button*

Follow these actions to create a new room:

1. Click the Create a room button as shown in Figure i-4.
2. Name the room and set the dimensions as shown in Figure i-5.
3. Save the room by clicking the green tick in the top left as in Figure i-5.

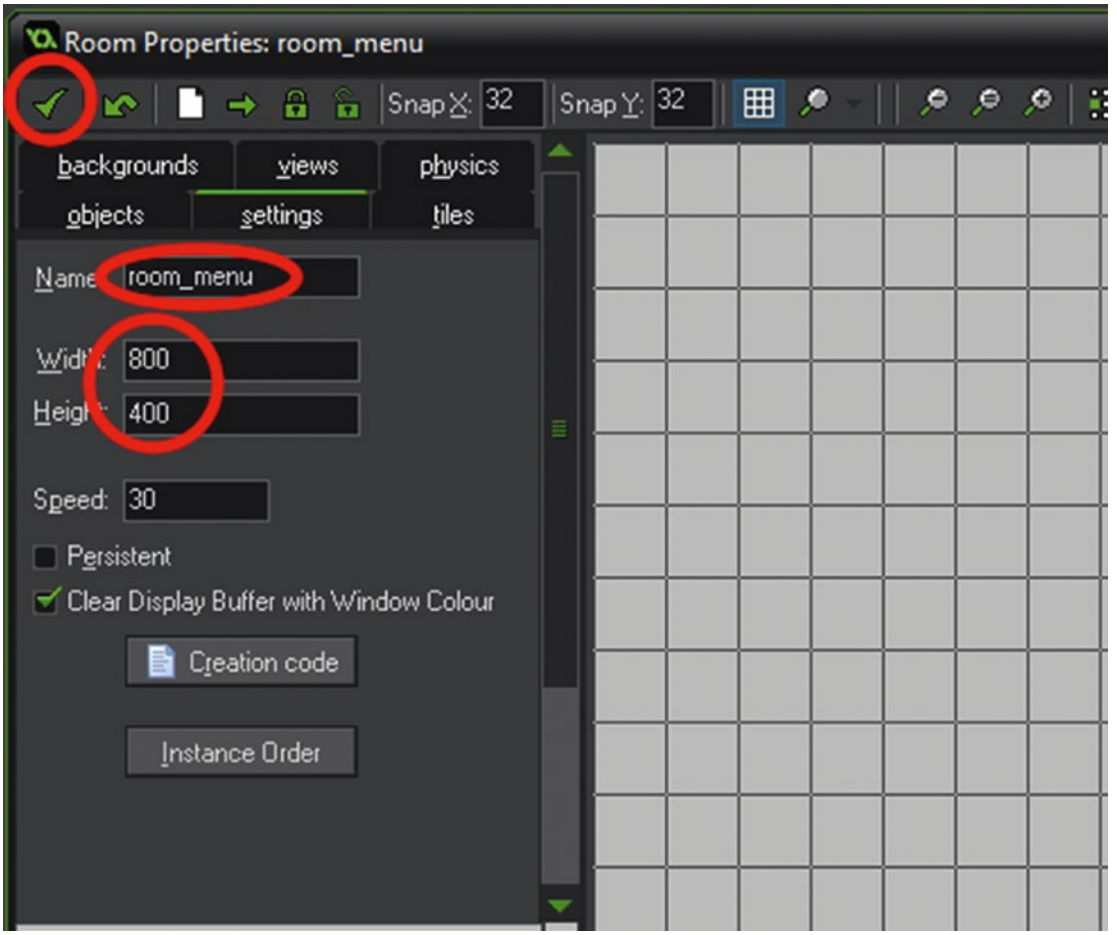

Figure i-5. *Name room and set dimensions – step 2*

Now that you've created a room, you have a place for objects to be added to.
Repeat this process for **`room_game`**, again setting the dimensions to 800 by 400.

Sounds

Sounds can be music or sound effects. You name each one and use code later to play the sound when you want to hear it. We will load them now, so we can simply refer to them later.

The example uses two sounds: **`snd_yeah`** and **`snd_you_are_dead.`**
You can do this by clicking the **Create a sound** button as shown in Figure i-6:

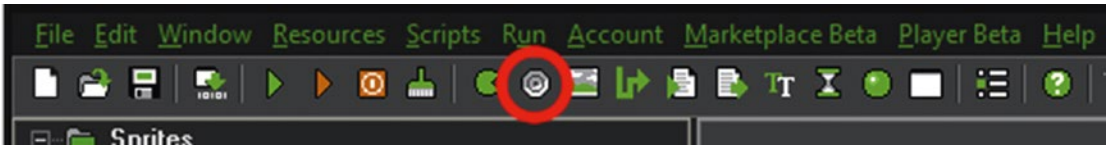

Figure i-6. *Create a new sound*

Then navigate to where the resource file is stored. Give the sound a name – **snd_yeah** – you can use the default settings, and then click **OK**. This step is shown in Figure i-7:

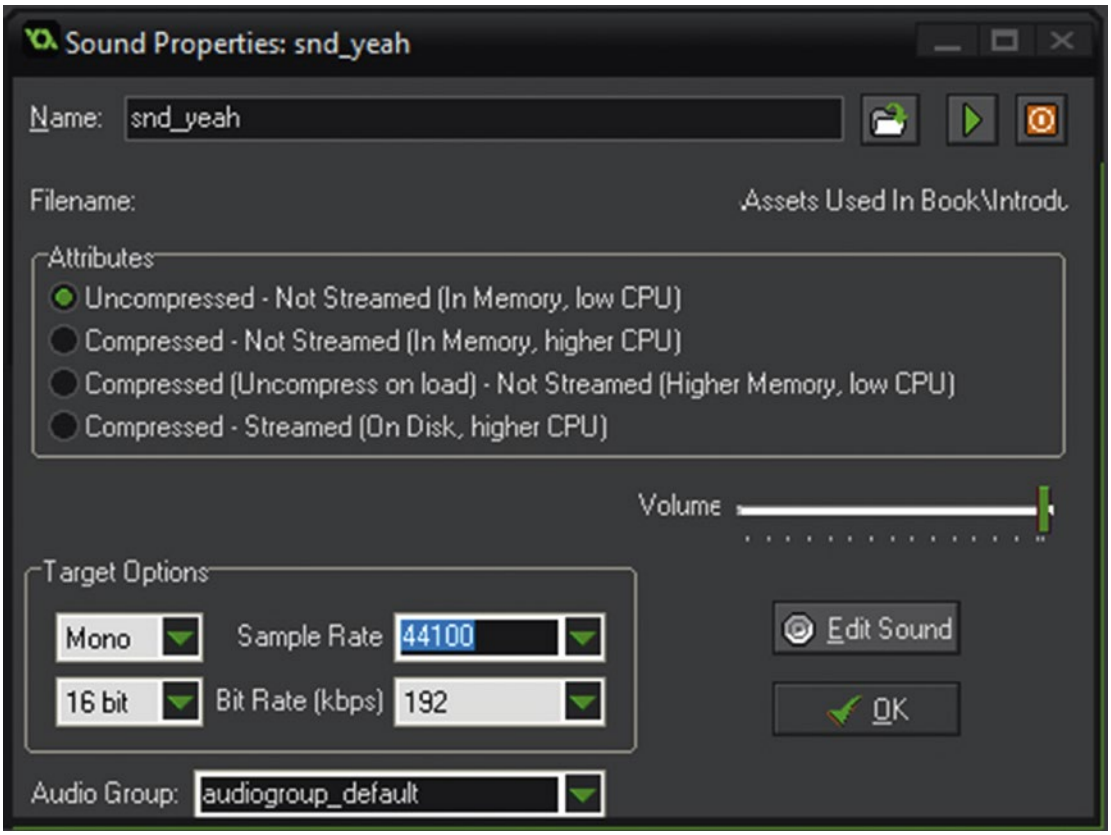

Figure i-7. *Name a sound and load it from the resources folder – step 3*

Select the appropriate sound from the resources folder as shown in Figure i-8:

Figure i-8. *Choosing a sound file to load*

Repeat this with the sound file **`snd_you_are_dead`**.

Fonts

If you want to display text or variables on screen in your game, you're going to need to define and name some fonts. You can then set drawing to this font when you want it displayed. A font can be created by clicking the **Create a font** button as shown in Figure i-9:

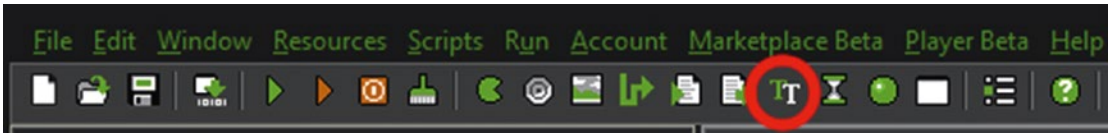

Figure i-9. *Creating a font*

Set the font name as **`font_hud`** and the size as 20 **`Arial`** as shown in Figure i-10:

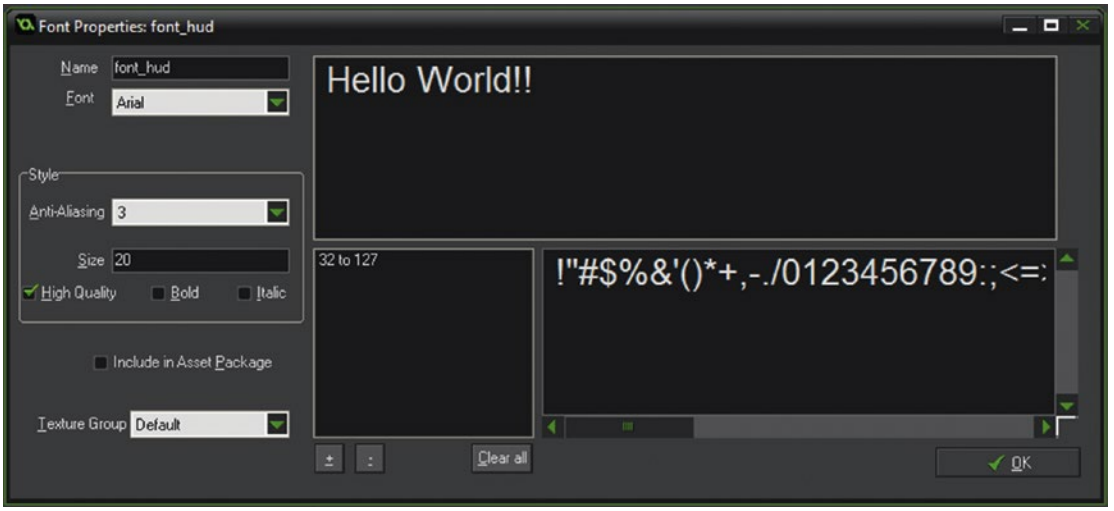

Figure i-10. *Naming and setting a font – step 4*

Objects

Objects are the life blood of GameMaker: Studio. Objects will be used for displaying sprites, playing sounds, drawing text, detecting movement, processing functions, performing math calculations, and more.

Next we'll create the objects. There are five of them: **obj_logo**, **obj_start**, **obj_target**, **obj_exit**, and **obj_hud.**

First create the object **obj_logo** and assign the sprite to it.

This can be done by clicking the **Create Object** button shown in Figure i-11:

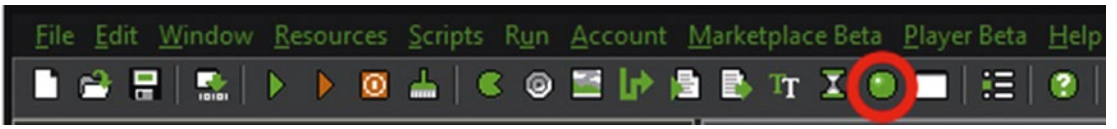

Figure i-11. *Creating a new object*

Next is to assign a sprite to this object, assign the sprite **spr_logo** as shown in Figure i-12:

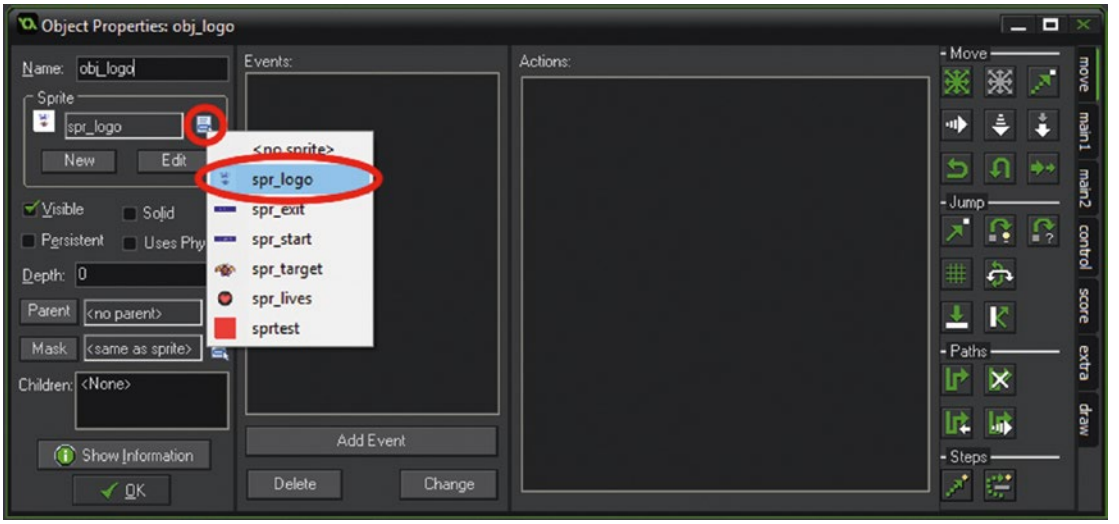

Figure i-12. *Assigning a sprite to an object – step 5*

Then click ok.

Next create a new object, **obj_start** and assign the sprite **spr_start_game**.

The next step is to program some **Events**. Events are things that happen. The events you'll use most are the **Create Event**, **Step Event**, **Alarm Event**, and **Draw Event**. These can be set up using GameMaker: Studio's built-in GUI.

Do this by clicking **Add Event** then **Create Event**, as shown in Figure i-13:

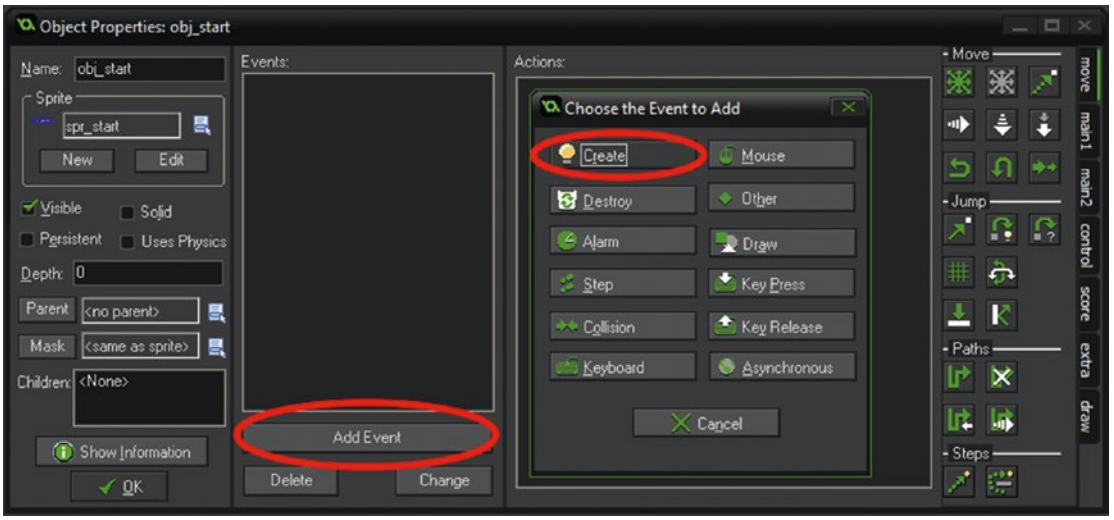

Figure i-13. *Making a create event*

Click on the control tab, and click and drag **Execute Code** to the actions window, shown below in Figure i-14:

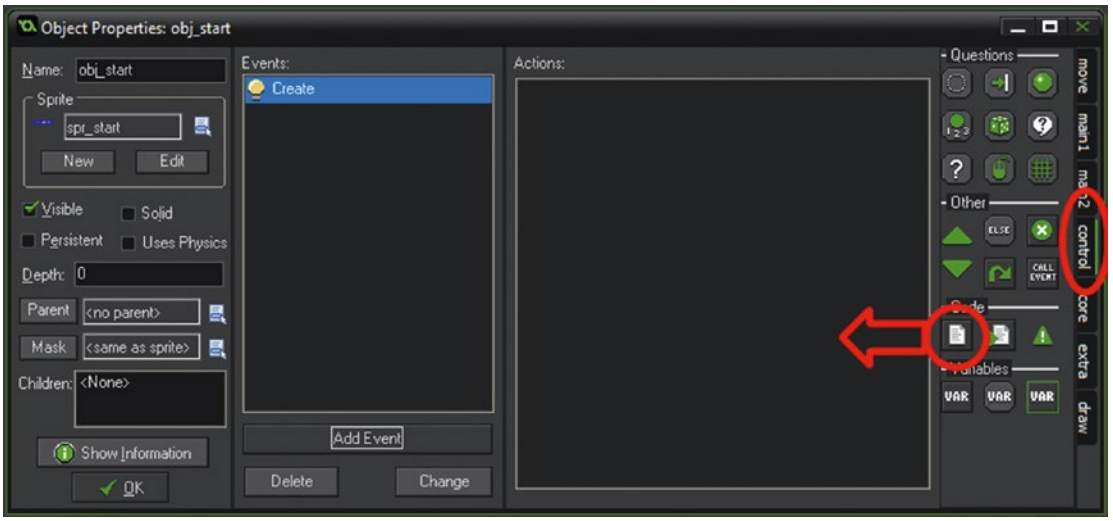

Figure i-14. *Adding a code action – step 5*

In the open window, enter the following code:

```
//see if ini file exists and load saved score
ini_open("savedata.ini"); //open file savedata.ini
global.highscore = ini_read_real("score", "highscore", 0); //set global.highscore to value
or set as 0 if no value present
ini_close(); //close ini file - always do this after loading or saving data
//set starting values for game:
score=0;
lives=5;
```

This code will load any high score from a previous play of the game to the variable `global.highscore`, set current ***score*** to 0, and ***lives*** to 5. It is not important at this stage to understand this code. The purpose of this exercise is to learn how to add GML code to an event. When you've added the code, the open window will look as shown in Figure i-15.

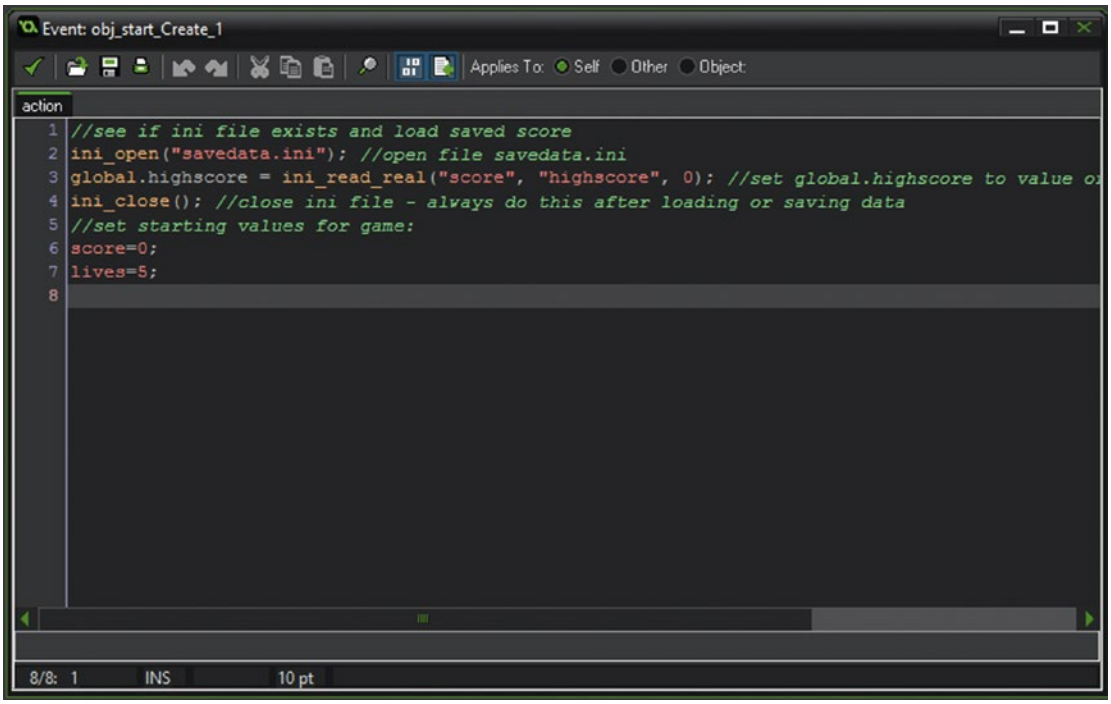

Figure i-15. *Adding code to action – step 6*

Next create a new event, a **Mouse Left Button Released Event** as shown in Figure i-16:

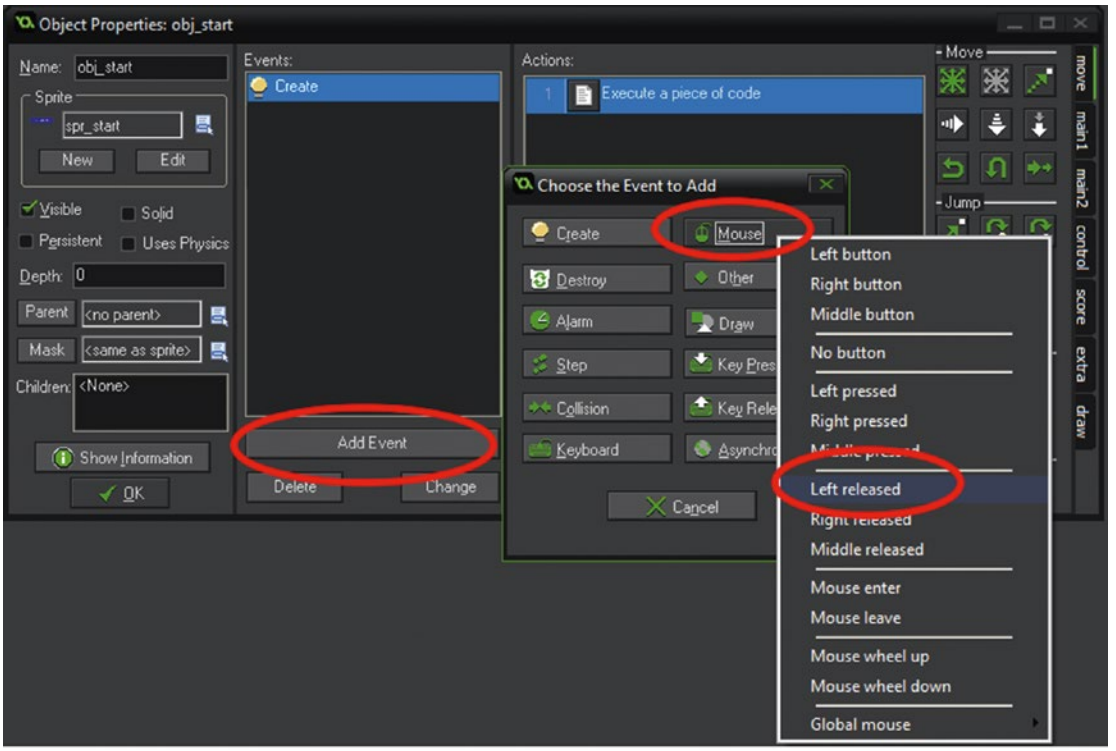

Figure i-16. *Creating a mouse left button released event – step 7*

Again drag over the **Execute Code** action and add the following code, as shown in Figure i-17, to this action:

```
room_goto(room_game); //goto the room room_game
```

Next add a **Draw Event** by clicking **Add Event** followed by **Draw Event**, then drag across the **Execute Code**. Add the following GML to this:

```
draw_self(); //draws sprite assigned to this object
draw_set_font(font_hud); //set font
draw_set_halign(fa_center); //set horizontal alignment for drawn text
draw_set_colour(c_black); //sets drawing colour as black
draw_text(250,280, "Highscore: "+ string(global.highscore)); //draw Highscore: plus value
of global.highscore
```

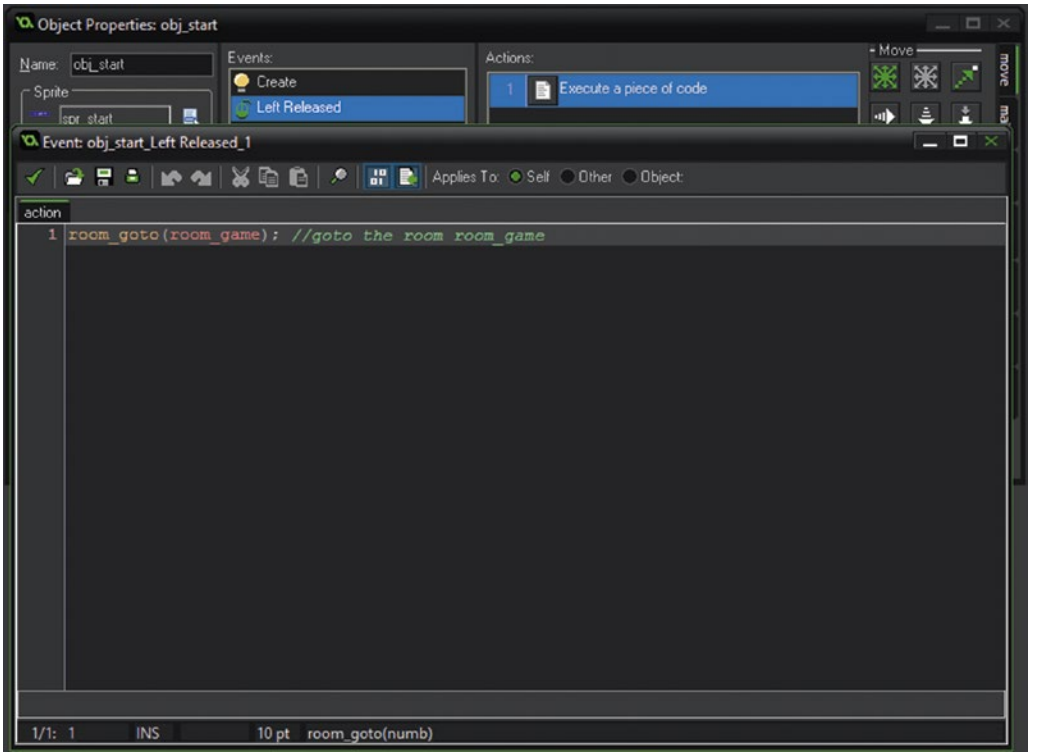

Figure i-17. *Adding code to the left mouse button released event – step 8*

A **Draw Event** is where you place your code, or D&D, to place text and images on the screen. Drawing functions, such as `draw_text` and `draw_self`, **must** be placed in **Draw Event**. Figure i-18 shows this setup with code added.

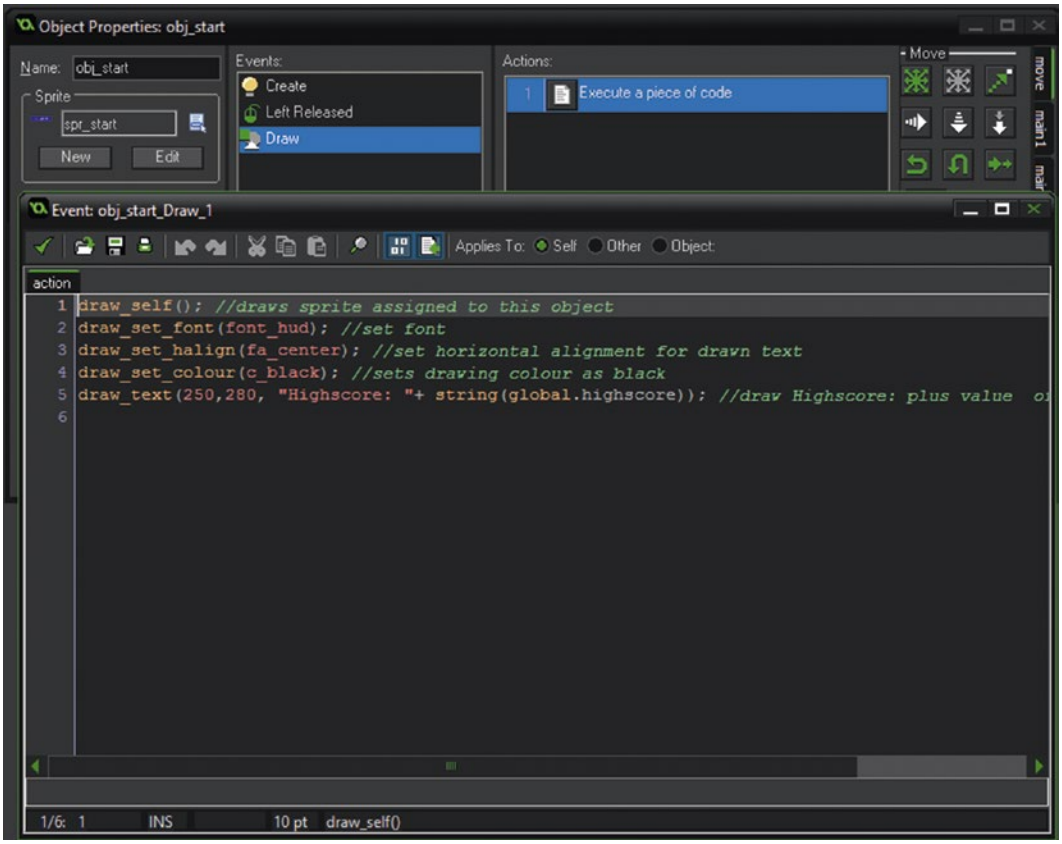

Figure i-18. *Adding code to a draw event – step 9*

Click OK to save all changes.

Explanation of the code above:

```
draw_text(250,280, "Highscore: "+ string(global.highscore)); //draw Highscore: plus value
of global.highscore
```

This draws the text at position 250 across the screen and 280 down, in pixels.

`global.highscore` has a numerical value. Because we are drawing it with a string, "`Highscore:` ", we need to convert it also to a string. The code `string(global.highscore)` does this conversion. A more in-depth explanation of variable types is provided in Chapter 1.

Next create a new object **`obj_exit`** and assign the sprite **`spr_exit`**.

Create a **Left Mouse Button Released Event** and add this code:

```
game_end(); //closes game and returns to windows
```

That is all for this object. You should now know some of the basics of using **`Objects`**, such as creating a new object and setting a sprite.

Create a new object **`obj_target`** and set the sprite **`spr_target`**.

Next we'll use a **Create Event** to set up some initial variables. A **Create Event** is only run once when the object is created, or when a room starts if the object is already placed in it.

We'll use this event to create at a random position across the screen, X, and down the screen Y between 100 and 700.

We'll then start an Alarm with a value of 100 minus the score. This makes the alarm quicker as the score increases, making it get progressively harder to click in time.

In a **Create Event** put this code, which will choose a whole integer between 100 and 700, sets timer to 100 less the score, with a minimum value of 5, and then sets an alarm with the timer:

```
x=irandom_range(100,700); //sets x position at random between 100 & 700
y=irandom_range(100,300); //sets y position at random between 100 & 300
timer=100-score; //set timer as 100 less score - so it gets faster
if timer<=5 timer=5; //check if less than 5, set as 5 if it is
alarm[0]=timer;
```

Next create an **Alarm Event0** as shown in Figure i-19. This will activate if the player hasn't clicked the object in time. The GML will play a sound first, reduce the player's ***lives*** by 1, and create a new object, then destroy itself.

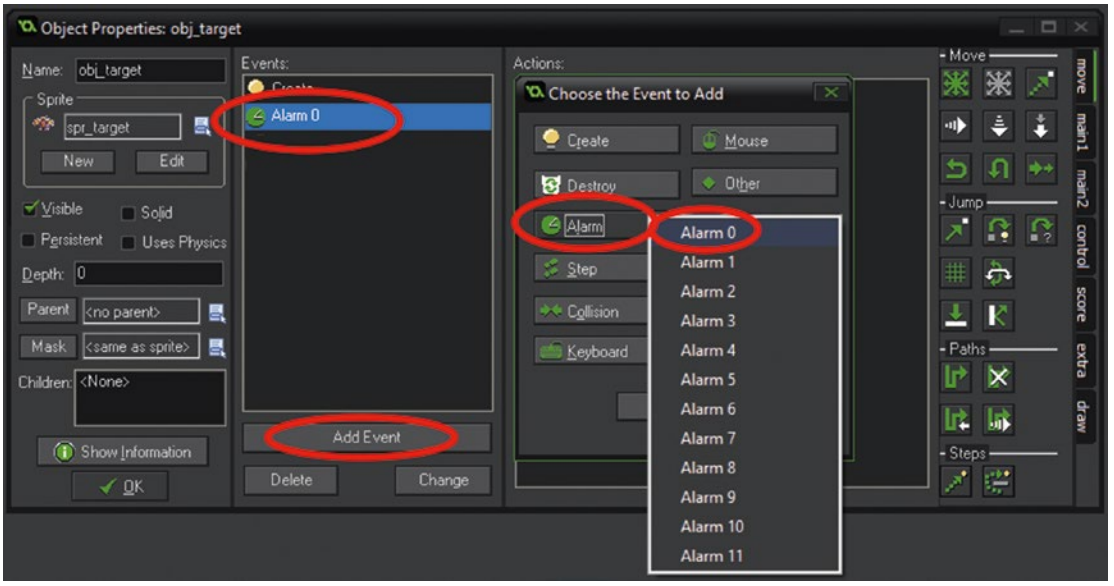

Figure i-19. *Creating an alarm event for alarm[0] – step 10*

Drag across **Execute Code** and add the following code:

```
audio_play_sound(snd_you_are_dead,1,false);//plays a sound
lives-=1; //reduce lives
instance_create(50,50,obj_target); // create a target
instance_destroy(); //destroy self
```

Create a **Left Mouse Button Released Event** into the same object, **obj_target** and put the following code in it:

```
score+=1; //add 1 to score
audio_play_sound(snd_yeah,1,false); //play sound yeah
instance_create(50,50,obj_target); //create new skull
instance_destroy(); //removes self from screen
```

In a **Draw Event** of **obj_target** put:

```
draw_self(); // draws assigned sprite
draw_set_colour(c_red); //sets drawing colour
draw_rectangle(x-(alarm[0]/2), y-30, x+(alarm[0]/2), y-25,0); //draws a rectangle that
reduces size based on alarm[0] value
```

The above code will draw the sprite for the object, set the drawing colour to red, and then draw a rectangle based on the current value of the **alarm** – this will serve as visual so the player knows how long they have to click the object. Save this object by clicking OK. You'll learn more about drawing geometric shapes in section 3.

Next create an object **obj_hud**. There is no sprite for this object. This object will be used as a control object that will be used to draw a HUD of the player's ***lives*** and ***score***. It will also monitor how many lives the player has, and if the player has lost all of their lives it will update the high score if the player has a new high score and then restart the game. You do not need to create this file; it will be created automatically upon saving if it doesn't already exist. Click **Add Event** and then **Step Event.** Add the following code to the **Step Event**:

```
if (lives<0)
{
  if (score>global.highscore)
  {
        ini_open("savedata.ini");
        ini_write_real( "score", "highscore", score);
        ini_close(); //closes  ini file
  }
 game_restart(); //restarts game
}
```

This is shown added in Figure i-20. This code will update the saved value in the INI file if the current ***score*** is bigger than `global.highscore`.