

# 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  
DOI 10.1007/978-1-4842-2373-4

ISBN-13 (electronic): 978-1-4842-2373-4

Library of Congress Control Number: 2016962191

Copyright © 2016 by Ben Tyers

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

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](mailto:orders-ny@springer-sbm.com), or visit [www.springer.com](http://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](mailto:rights@apress.com), or visit [www.apress.com](http://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](http://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](http://www.apress.com). For detailed information about how to locate your book's source code, go to [www.apress.com/source-code/](http://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

<b>About the Author .....</b>	<b>xiii</b>
<b>About the Technical Reviewer .....</b>	<b>xv</b>
<b>Acknowledgments .....</b>	<b>xvii</b>
<b>Introduction .....</b>	<b>xix</b>
<b>■ Chapter 1: Variables .....</b>	<b>1</b>
<b>■ Chapter 2: Conditionals .....</b>	<b>9</b>
<b>■ Chapter 3: Drawing.....</b>	<b>15</b>
<b>■ Chapter 4: Drawing Continued.....</b>	<b>25</b>
<b>■ Chapter 5: Keyboard Input and Simple Movement .....</b>	<b>35</b>
<b>■ Chapter 6: Objects and Events.....</b>	<b>39</b>
<b>■ Chapter 7: Sprites.....</b>	<b>47</b>
<b>■ Chapter 8: Health, Lives, and Score.....</b>	<b>55</b>
<b>■ Chapter 9: Mouse .....</b>	<b>63</b>
<b>■ Chapter 10: Alarms.....</b>	<b>69</b>
<b>■ Chapter 11: Collisions.....</b>	<b>75</b>
<b>■ Chapter 12: Rooms .....</b>	<b>85</b>
<b>■ Chapter 13: Backgrounds .....</b>	<b>93</b>
<b>■ Chapter 14: Sounds and Music.....</b>	<b>99</b>
<b>■ Chapter 15: Splash Screens and Menu.....</b>	<b>105</b>
<b>■ Chapter 16: Random.....</b>	<b>113</b>
<b>■ Chapter 17: More Movement (Basic AI).....</b>	<b>117</b>

■ Chapter 18: INI Files .....	125
■ Chapter 19: Effects .....	129
■ Chapter 20: Loops.....	133
■ Chapter 21: Arrays .....	137
■ Chapter 22: ds_lists.....	147
■ Chapter 23: Paths .....	153
■ Chapter 24: Scripts.....	159
■ Chapter 25: Hints and Tips .....	165
■ Chapter 26: Creating a Game – Outline .....	171
■ Chapter 27: Creating a Game – Sprites .....	173
■ Chapter 28: Creating a Game – Sounds .....	179
■ Chapter 29: Creating a Game – Backgrounds.....	181
■ Chapter 30: Creating a Game – Paths.....	183
■ Chapter 31: Creating a Game – Fonts .....	185
■ Chapter 32: Creating a Game – Scripts .....	187
■ Chapter 33: Creating a Game – Parent Objects .....	193
■ Chapter 34: Creating a Game – Objects.....	199
■ Chapter 35: Creating a Game – Rooms.....	277
■ Chapter 36: Creating a Game – Progress Sheet.....	285
■ Chapter 37: Creating a Game – Marking Guide .....	287
■ Chapter 38: Creating a Game – End of Projects Assignments .....	289
■ Chapter 39: End of Project Test .....	295
■ Chapter 40: Summary.....	307
Index.....	309

# Contents

<b>About the Author .....</b>	<b>xiii</b>
<b>About the Technical Reviewer .....</b>	<b>xv</b>
<b>Acknowledgments .....</b>	<b>xvii</b>
<b>Introduction .....</b>	<b>xix</b>
<b>■ Chapter 1: Variables .....</b>	<b>1</b>
Worksheet – Variables.....	4
Worksheet – Variables – Answer Sheet.....	6
Basic Projects.....	7
Advanced Project .....	7
End of Book Game Variables .....	8
<b>■ Chapter 2: Conditionals .....</b>	<b>9</b>
Worksheet – Conditionals.....	11
Worksheet – Conditionals – Answer Sheet .....	12
Basic Projects.....	12
Advanced Projects.....	13
End of Book Game Conditionals .....	13
<b>■ Chapter 3: Drawing.....</b>	<b>15</b>
Worksheet – Drawing.....	19
Worksheet – Drawing – Answer Sheet.....	20
Basic Projects.....	21
Advanced Project .....	21
End of Book Game Drawing.....	23

■ <b>Chapter 4: Drawing Continued</b> .....	<b>25</b>
Worksheet – Drawing Continued.....	30
Worksheet – Drawing Continued – Answer Sheet.....	31
Basic Projects.....	32
Advanced Projects.....	32
End of Book Game Drawing Continued .....	33
■ <b>Chapter 5: Keyboard Input and Simple Movement</b> .....	<b>35</b>
Worksheet – Key Presses and Simple Movement .....	37
Worksheet – Key Presses and Simple Movement – Answer Sheet.....	37
Basic Projects.....	38
Advanced Project .....	38
End of Book Game Keypresses and Simple Movement.....	38
■ <b>Chapter 6: Objects and Events</b> .....	<b>39</b>
Worksheet – Objects .....	43
Worksheet – Objects – Answer Sheet.....	44
Basic Projects.....	45
Advanced Project .....	45
End of Book Game Objects.....	45
■ <b>Chapter 7: Sprites</b> .....	<b>47</b>
Worksheet – Sprites.....	51
Worksheet – Sprites – Answer Sheet.....	52
Basic Projects.....	52
Advanced Project .....	53
End of Book Game Sprites.....	53
■ <b>Chapter 8: Health, Lives, and Score</b> .....	<b>55</b>
Worksheet – Lives, Health, & Score .....	59
Worksheet – Lives, Health, Lives, & Score – Answer Sheet.....	60

Basic Projects.....	60
Advanced Projects.....	61
End of Book Game Health, Lives, & Score .....	61
<b>■ Chapter 9: Mouse .....</b>	<b>63</b>
Worksheet – Mouse Movement.....	66
Worksheet – Mouse Movement – Answer Sheet.....	66
Basic Projects.....	67
Advanced Projects.....	67
End of Book Game Mouse Movement .....	67
<b>■ Chapter 10: Alarms.....</b>	<b>69</b>
Worksheet – Alarms .....	71
Worksheet – Alarms – Answer Sheet.....	71
Basic Projects.....	72
Advanced Projects.....	73
End of Book Game – Ten Alarms .....	73
<b>■ Chapter 11: Collisions.....</b>	<b>75</b>
Worksheet – Collision Events .....	81
Worksheet – Collision Events – Answer Sheet.....	82
Basic Projects.....	82
Advanced Projects.....	83
End of Book Game Collisions.....	83
<b>■ Chapter 12: Rooms .....</b>	<b>85</b>
Worksheet – Rooms .....	90
Worksheet – Rooms – Answer Sheet.....	90
Basic Projects.....	91
Advanced Project .....	91
End of Book Game Rooms .....	92



■ <b>Chapter 13: Backgrounds</b> .....	<b>93</b>
Worksheet - Backgrounds.....	96
Worksheet – Backgrounds – Answer Sheet.....	96
Basic Projects.....	97
Advanced Projects.....	97
End of Book Game Backgrounds.....	97
■ <b>Chapter 14: Sounds and Music</b> .....	<b>99</b>
Worksheet – Sounds & Music .....	102
Worksheet – Sounds & Music – Answer Sheet.....	102
Basic Projects.....	103
Advanced Projects.....	103
End of Book Game Sounds & Music .....	103
■ <b>Chapter 15: Splash Screens and Menu</b> .....	<b>105</b>
Worksheet – Splash Screens & Menu .....	109
Worksheet – Splash Screens & Menu – Answer Sheet.....	110
Basic Projects.....	111
Advanced Project .....	112
End of Book Game Splash Screens & Menu.....	112
■ <b>Chapter 16: Random</b> .....	<b>113</b>
Worksheet – Random.....	114
Worksheet – Random – Answer Sheet.....	115
Basic Projects.....	116
Advanced Project .....	116
End of Book Game Random.....	116
■ <b>Chapter 17: More Movement (Basic AI)</b> .....	<b>117</b>
Worksheet – More Movement .....	122
Worksheet – More Movement – Answer Sheet.....	123

Basic Projects.....	124
Advanced Projects.....	124
End of Book Game .....	124
<b>■ Chapter 18: INI Files .....</b>	<b>125</b>
Worksheet – INI Files .....	126
Worksheet – INI Files – Answer Sheet .....	127
Basic Projects.....	127
Advanced Project .....	128
End of Book Game INI files .....	128
<b>■ Chapter 19: Effects .....</b>	<b>129</b>
Worksheet – Effects .....	130
Worksheet – Effects – Answer Sheet.....	131
Basic Projects.....	132
Advanced Projects.....	132
End of Book Game Effects .....	132
<b>■ Chapter 20: Loops.....</b>	<b>133</b>
Worksheet – Loops.....	134
Worksheet – Loops – Answer Sheet.....	134
Basic Projects.....	135
Advanced Projects.....	135
End of Book Game Loops .....	136
<b>■ Chapter 21: Arrays .....</b>	<b>137</b>
Worksheet – Array .....	142
Worksheet – Array – Answer Sheet.....	142
Basic Projects.....	145
Advanced Projects.....	145
End of Book Game Arrays.....	145

■ <b>Chapter 22: ds_lists</b> .....	<b>147</b>
Worksheet – ds_lists.....	150
Worksheet – ds_lists – Answer Sheet.....	150
Basic Projects.....	151
Advanced Project .....	151
End of Book Game ds_list .....	152
■ <b>Chapter 23: Paths</b> .....	<b>153</b>
Worksheet – Paths .....	156
Worksheet – Paths – Answer Sheet.....	157
Basic Projects.....	157
Advanced Projects.....	157
End of Book Game Paths.....	157
■ <b>Chapter 24: Scripts</b> .....	<b>159</b>
Worksheet – Scripts .....	161
Worksheet – Scripts – Answer Sheet.....	162
Basic Projects.....	163
Advanced Projects.....	163
End of Book Game Scripts.....	163
■ <b>Chapter 25: Hints and Tips</b> .....	<b>165</b>
Scripts Tricks.....	165
Testing.....	167
Assets Handling.....	167
Projects .....	169
■ <b>Chapter 26: Creating a Game – Outline</b> .....	<b>171</b>
■ <b>Chapter 27: Creating a Game – Sprites</b> .....	<b>173</b>
■ <b>Chapter 28: Creating a Game – Sounds</b> .....	<b>179</b>
■ <b>Chapter 29: Creating a Game – Backgrounds</b> .....	<b>181</b>
■ <b>Chapter 30: Creating a Game – Paths</b> .....	<b>183</b>

■ Chapter 31: Creating a Game – Fonts .....	185
■ Chapter 32: Creating a Game – Scripts .....	187
■ Chapter 33: Creating a Game – Parent Objects .....	193
■ Chapter 34: Creating a Game – Objects .....	199
■ Chapter 35: Creating a Game – Rooms .....	277
■ Chapter 36: Creating a Game – Progress Sheet.....	285
■ Chapter 37: Creating a Game – Marking Guide .....	287
■ Chapter 38: Creating a Game – End of Projects Assignments .....	289
Endless Runner .....	289
Shoot The Ducks .....	290
Pontoon .....	291
Side-Scrolling Shooter .....	292
End of Project Marking Guide.....	293
■ Chapter 39: End of Project Test .....	295
Test Paper Answers.....	299
■ Chapter 40: Summary.....	307
<b>Index.....</b>	<b>309</b>

# 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

# 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.

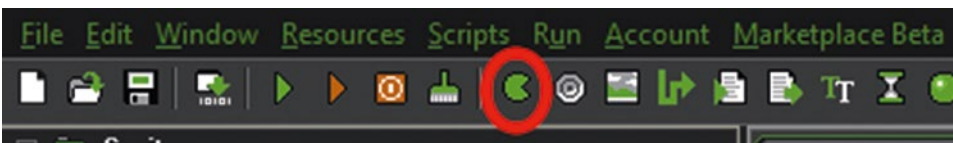
**Resources**

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

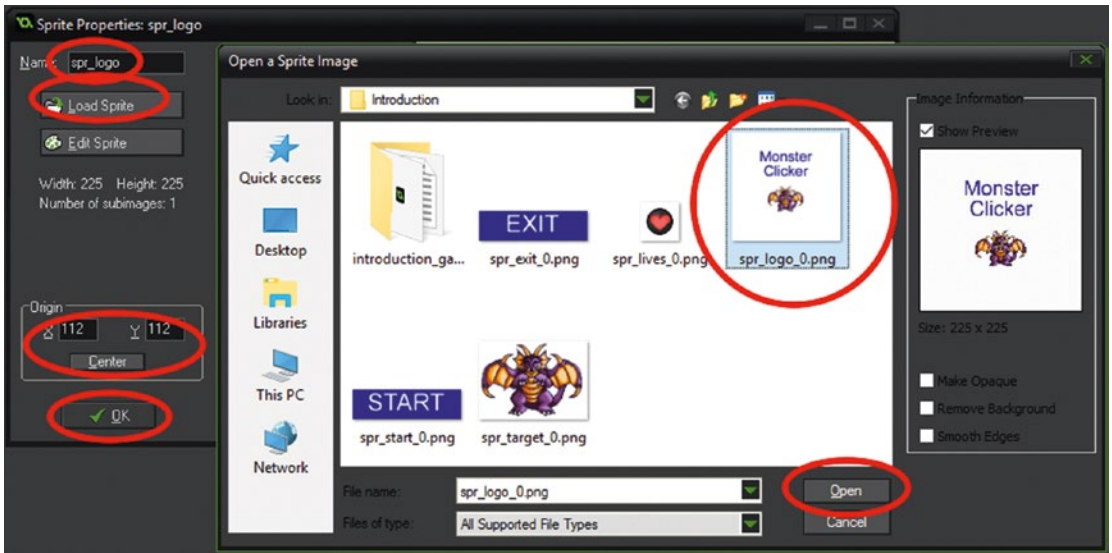
Sprites are images that will be used in your game. You'll use them to display the player, enemy, and other graphics in your game. They are in the downloadable resource folder: **Assets Used In Book > Introduction**. Sprites will be drawn by referencing or assigning them to objects. Next, load in the sprites for this game. There are five of them, **spr\_logo**, **spr\_start\_game**, **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\_health**, **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.



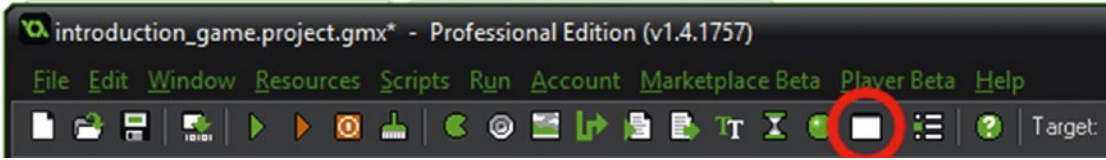
*Figure i-3. The sprite section will look like this*

## Rooms

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.

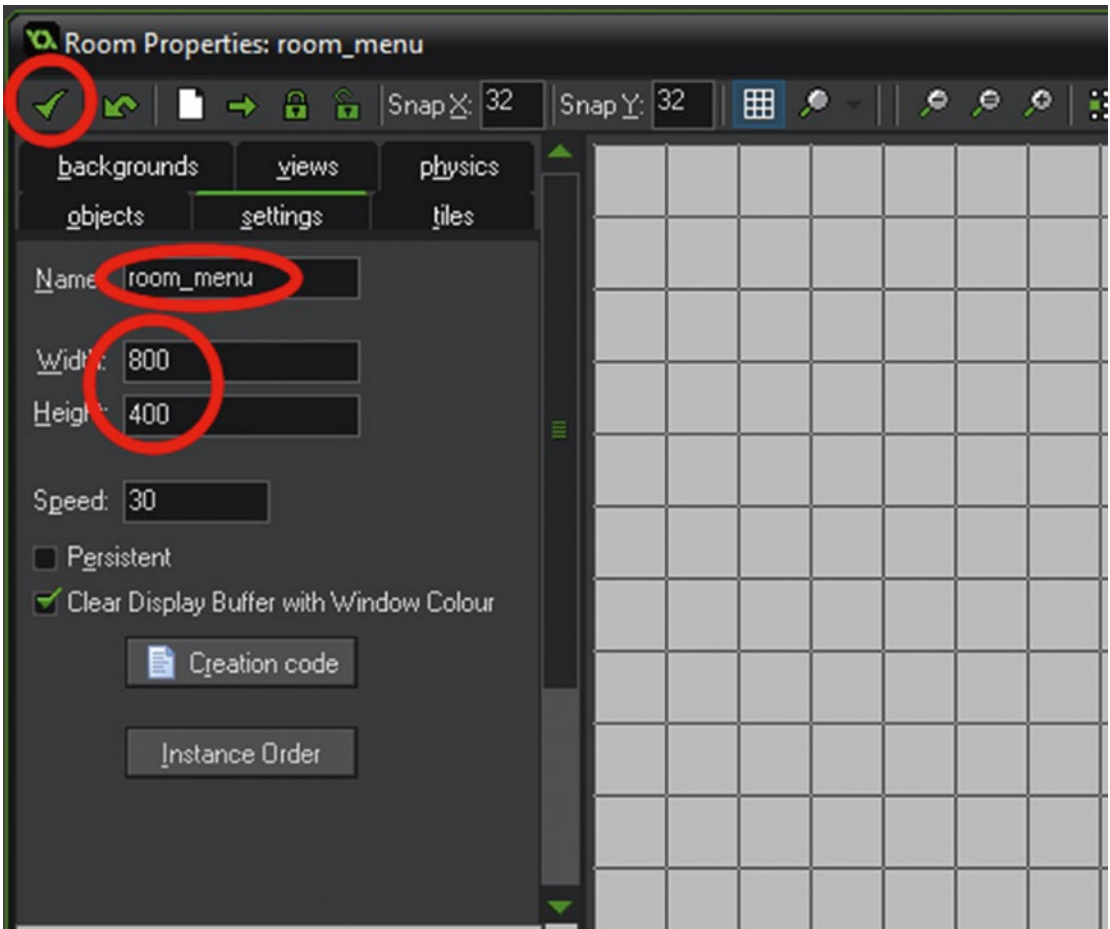
You can do this by clicking the **Create a room** button as shown circled in Figure i-4:



**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

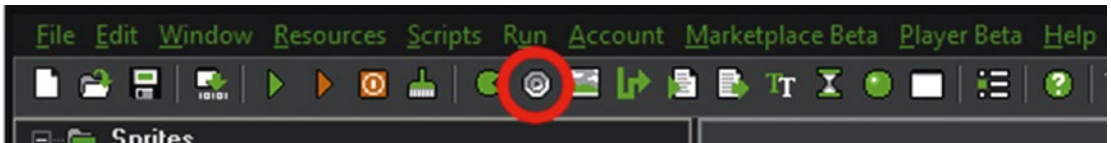
Now that you've created a room, you have a place for objects to be added to. Repeat this process for **room\_game**, again setting the dimensions to 800 by 400.

## 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.

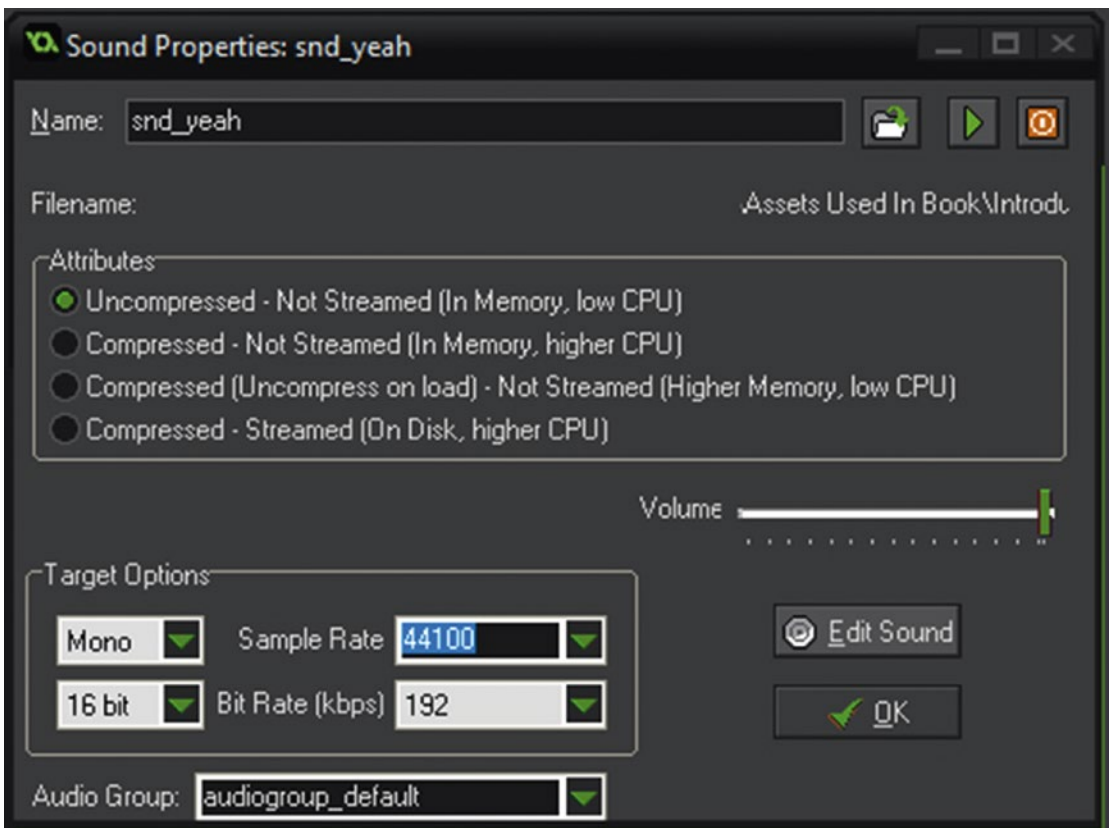
The example uses two sounds: **snd\_yeah** and **snd\_you\_are\_dead**.

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:

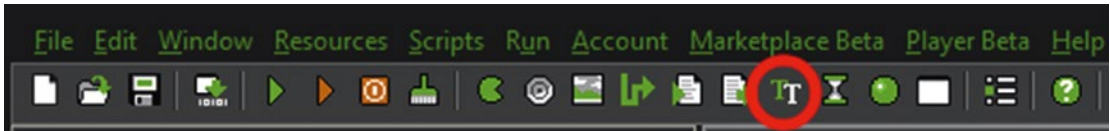
 snd_yeah	14/09/2014 19:50	Wave Sound	83 KB
 snd_you_are_dead	14/09/2014 19:50	Wave Sound	175 KB

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

Repeat this with the sound file **snd\_you\_are\_dead**.

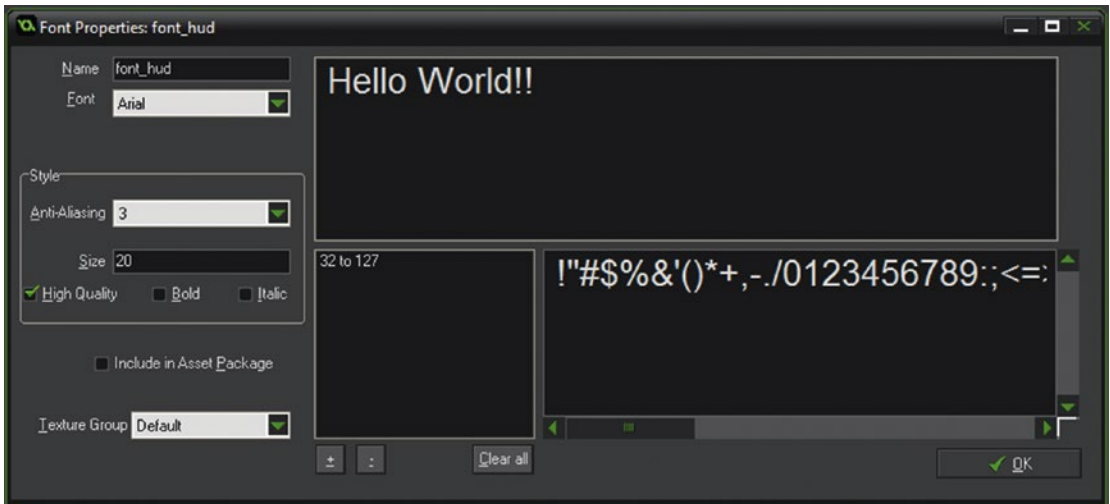
## Fonts

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

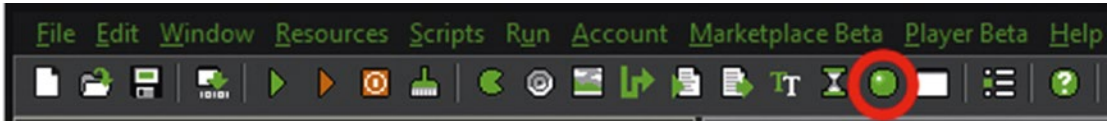
# Objects

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\_game**, **obj\_target**, **obj\_exit**, and **obj\_hud**.

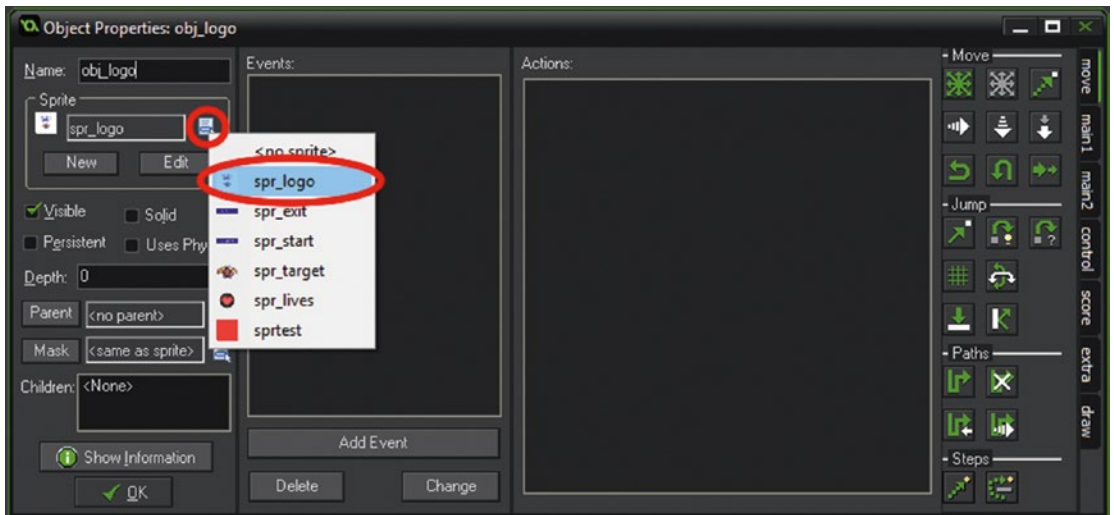
First create the object **obj\_logo** and assign the sprite to it.

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\_game** 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:

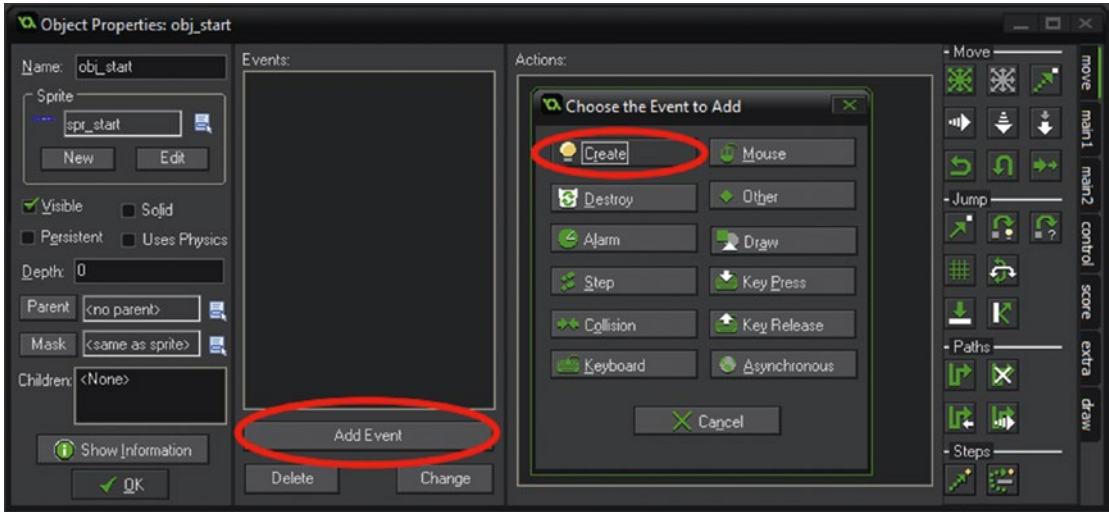


Figure i-13. Making a create event

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

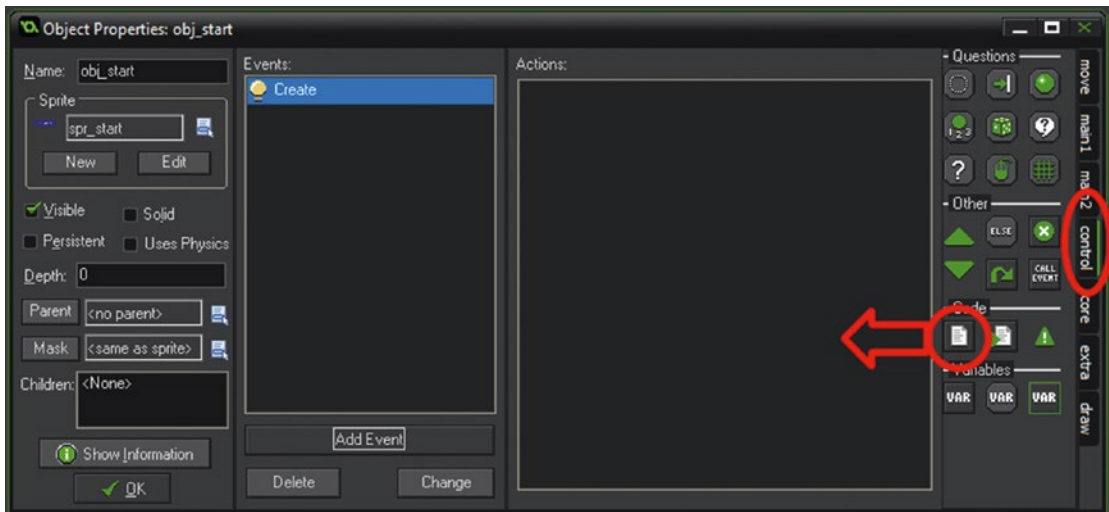
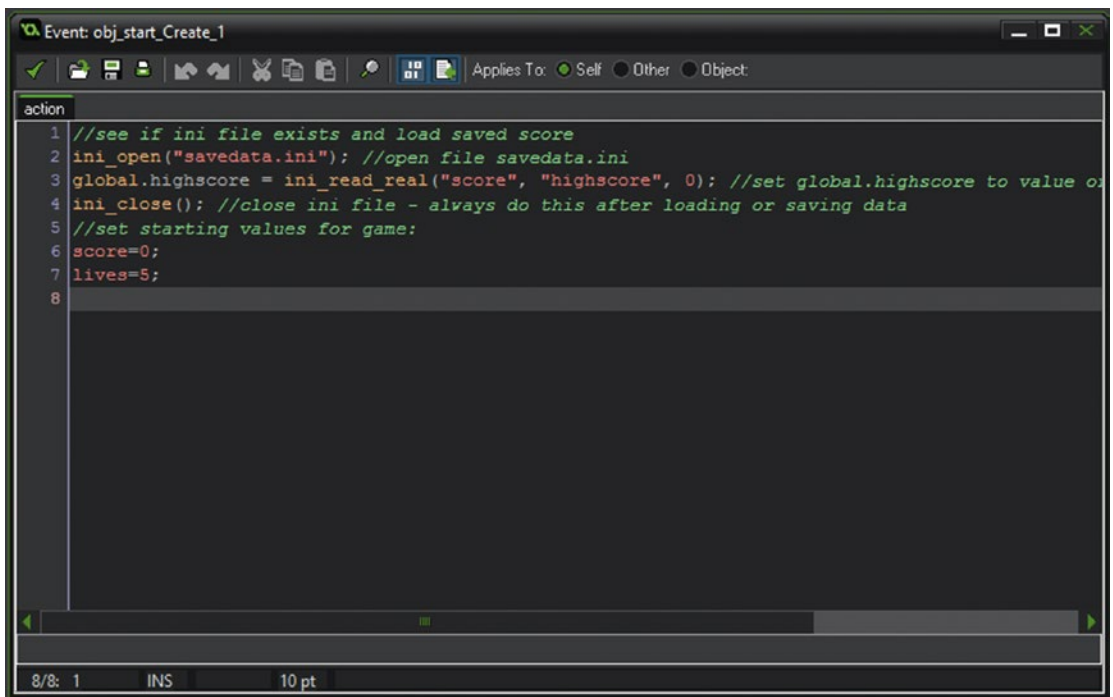


Figure i-14. Adding a code action – step 5

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;
```

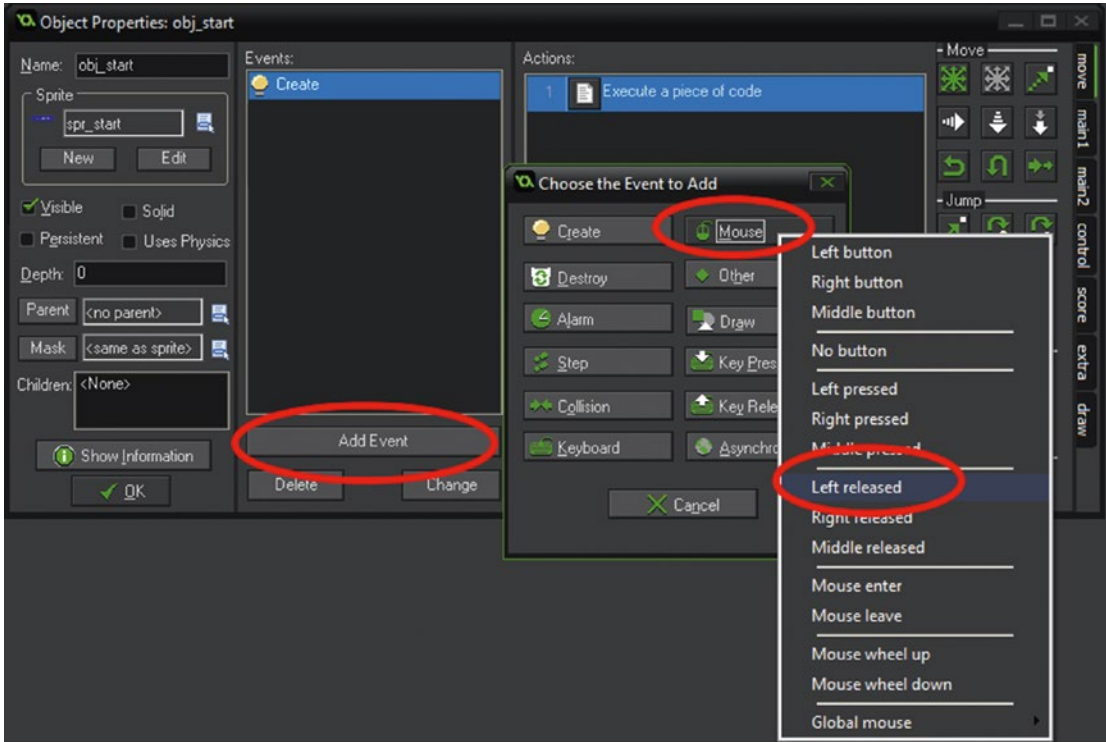
This code will load any high score from a previous play of the game to the variable `global.highscore`, set current **score** to 0, and **lives** to 5. It is not important at this stage to understand this code. The purpose of this exercise is to learn how to add GML code to an event. When you've added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



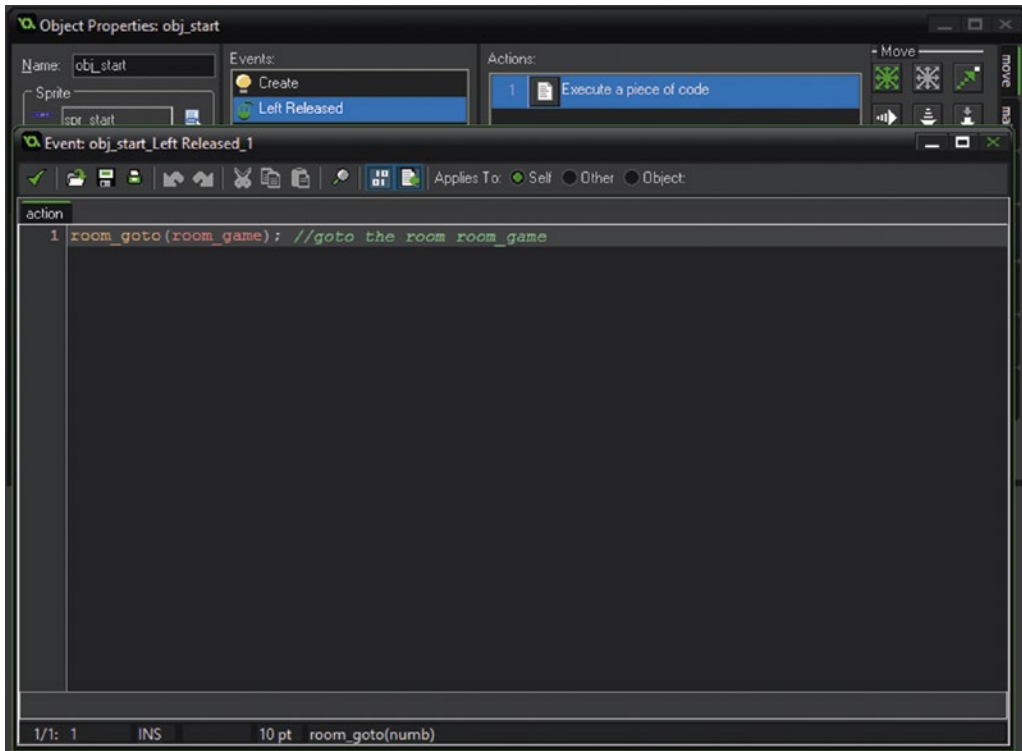
**Figure i-16.** Creating a mouse left button released event - step 7

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
```

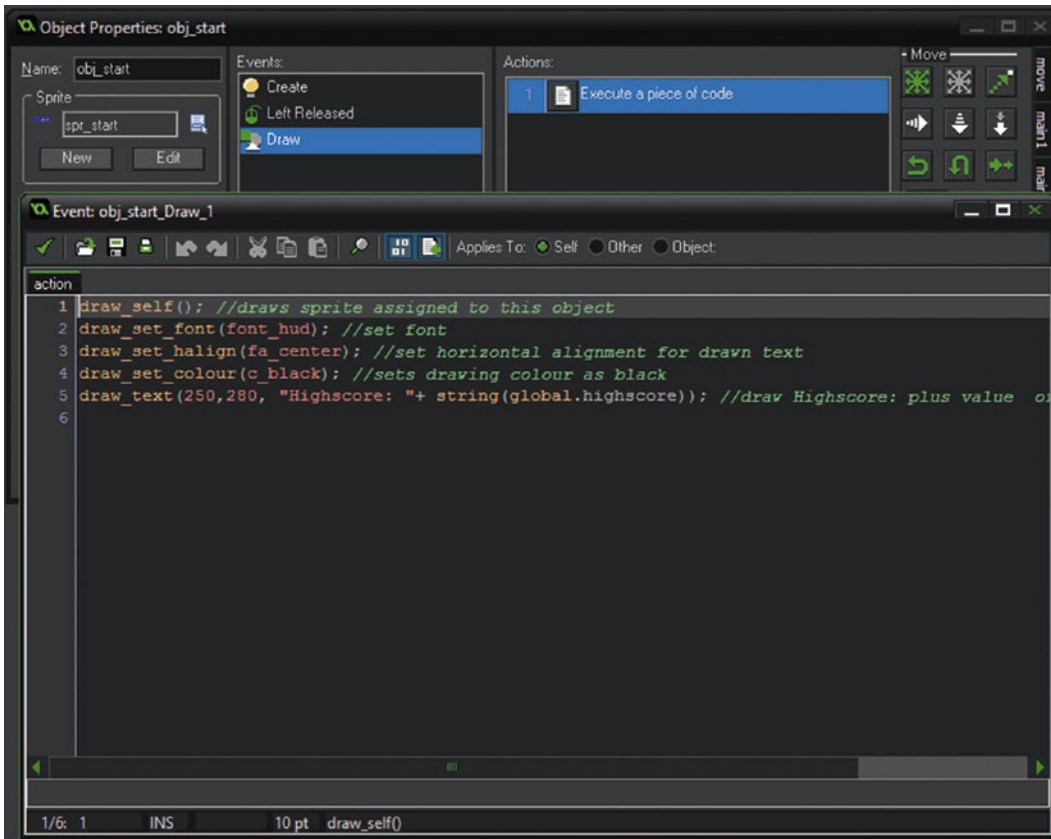
Next add a **Draw Event** by clicking **Add Event** followed by **Draw Event**, then drag across the **Execute Code**. Add the following GML to this:

```
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.

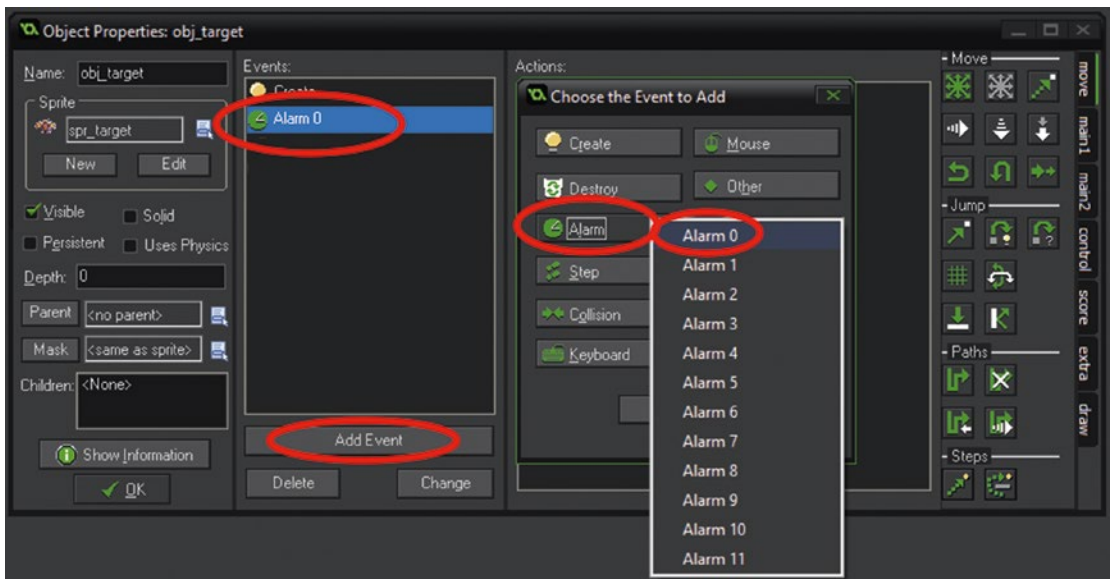
We'll use this event to create at a random position across the screen, X, and down the screen Y between 100 and 700.

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
```

## ■ INTRODUCTION

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:

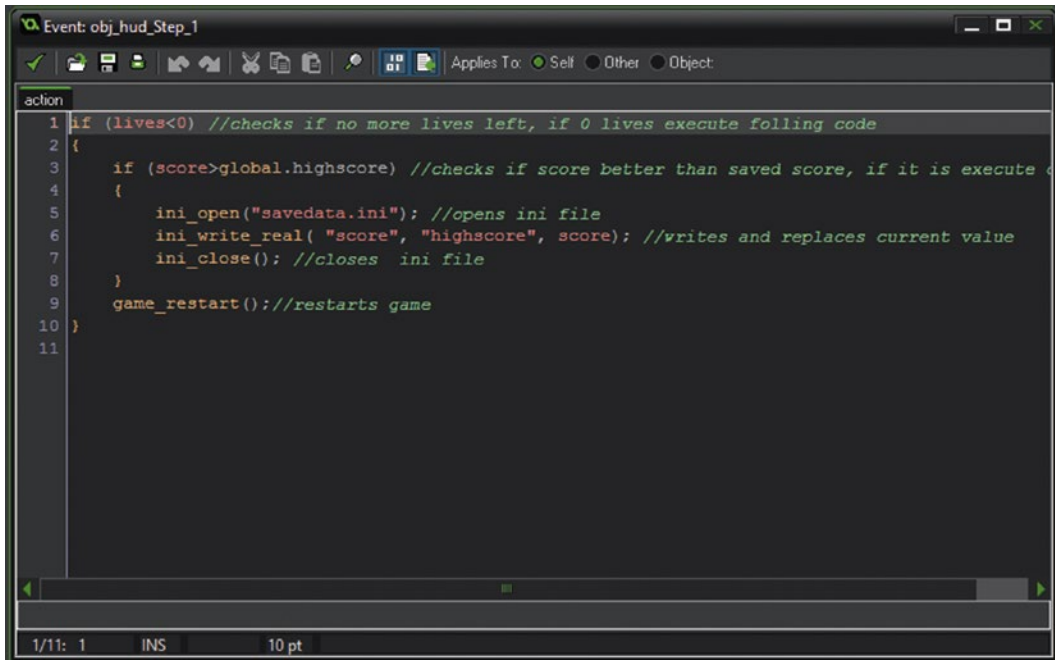
```
draw_self(); // draws assigned sprite
draw_set_colour(c_red); //sets drawing colour
draw_rectangle(x-(alarm[0]/2), y-30, x+(alarm[0]/2), y-25,0); //draws a rectangle that
reduces size based on alarm[0] value
```

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`.



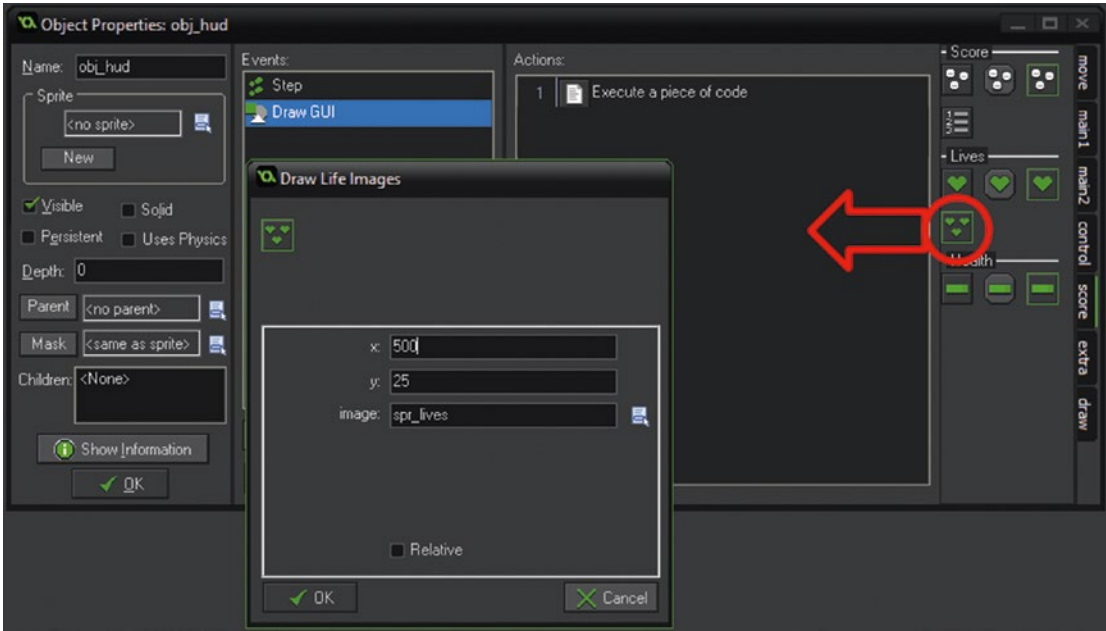
**Figure i-20.** Step event code for obj\_hud – step 11

Create a **Draw GUI Event**, under the draw tab. This code sets up the font, alignment, and drawing colour. Then it draws the **score** and as a high score if bigger than previous `global.highscore`.

```
draw_set_font(font_hud); //sets the font
draw_set_halign(fa_left); //sets alignment
draw_set_colour(c_blue); //sets drawing colour
draw_text(25, 25, "Score: "+string(score)); //draws score: + score
if (score <= global.highscore) { draw_set_colour(c_red); } draw_text(300, 25, "
(Highscore: " + string(global.highscore));
```

This makes use of a **Draw GUI Event**. This type of event will draw above any other objects in the room and is independent of any views. This type of event is commonly used to display health stats, scores, player info, weapon info, etc.

Next is to draw the **lives** as images. There is a Drag & Drop action for this in the **Score** section. Drag this across and set to draw at 500,25 using the sprite **spr\_lives** as shown in Figure i-21:

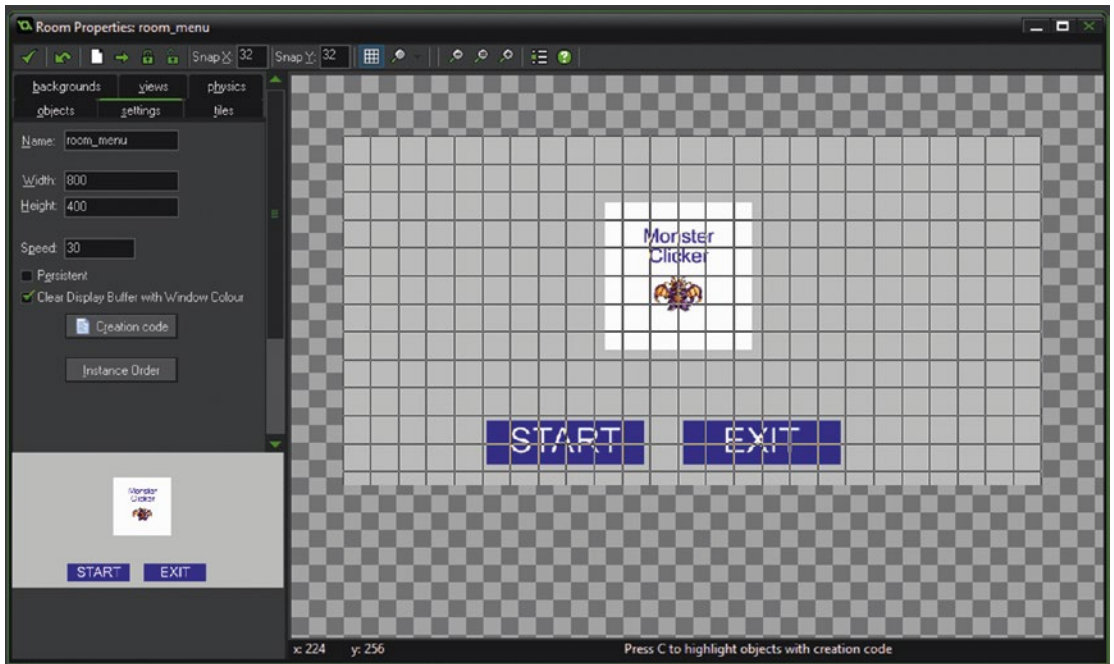


*Figure i-21. Drawing lives on screen as images – step 12*

Click OK twice to close open windows.

Open room **room\_menu** for editing by clicking on it in the resource tree.

Use the object tab to select objects and then place one each of **obj\_logo**, **obj\_start** and **obj\_exit**. Select the object then click in the room to place it. This step is shown in Figure i-22:



**Figure i-22.** *Placing objects in room – step 13*

Click the green tick in the top left of the window to save these settings.

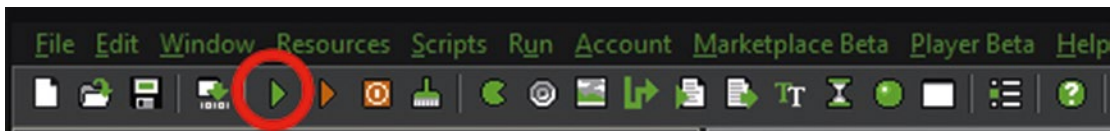
Next open **room\_game**.

Place one instance of **obj\_target** and one of **obj\_hud** in the room. It doesn't matter where you place them.

Click the green tick in the top left to apply changes.

Now click **File** and **Save As**. Give your game a name and save it.

Click the green triangle, shown in Figure i-23, at the top left to play your game.



**Figure i-23.** *Testing the game – step 14*

## Comments

Although well-formatted code with appropriately named assets can be easy to read, it's always worth adding comments to any code. When you come back or share the code with someone, you don't have to waste time trying to figure out or explaining what a certain code does.

In GameMaker: Studio there are three type of comments you can use. The first type is using the double //

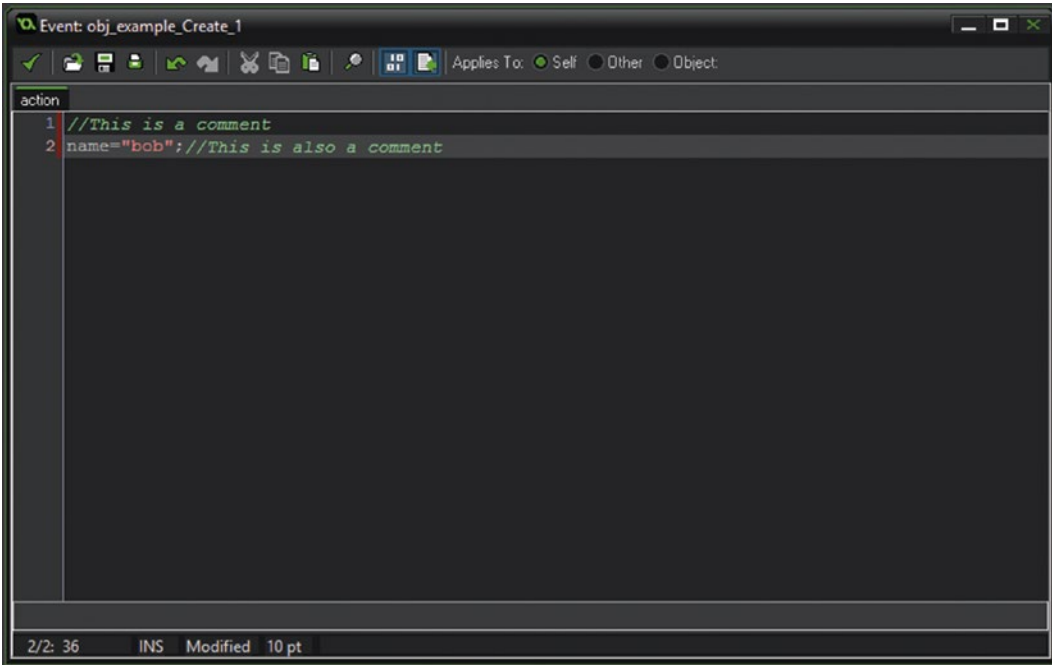


■ INTRODUCTION

An example in code would look like this:

```
//This is my comment
```

In your game, this would look like this as shown in Figure i-24. Anything written after the // is commented out and will not be processed. This can be a line on its own, or after code. Figure i-24 shows both being used.



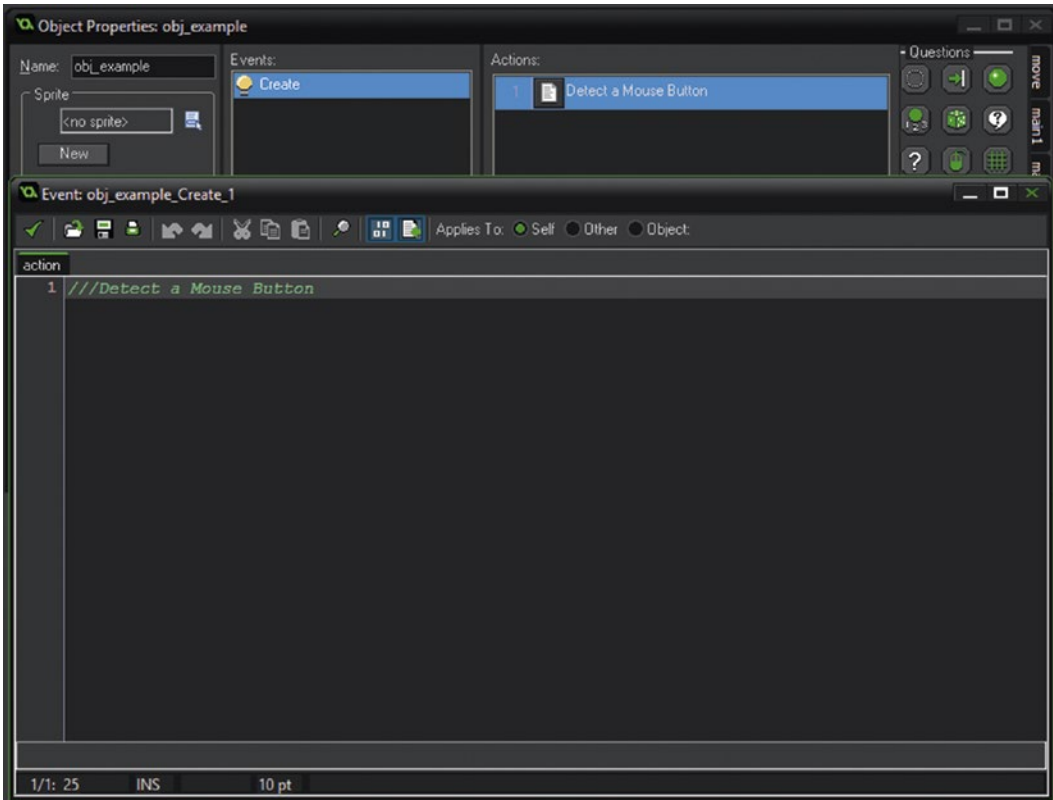
**Figure i-24.** Showing a comment

The next type uses the triple //. When placed at the top of code block it changes the default **Execute a piece of code** to the comment. This can also be used at the beginning of a piece of script to make it appear in Auto-Complete.

An example of this code would be:

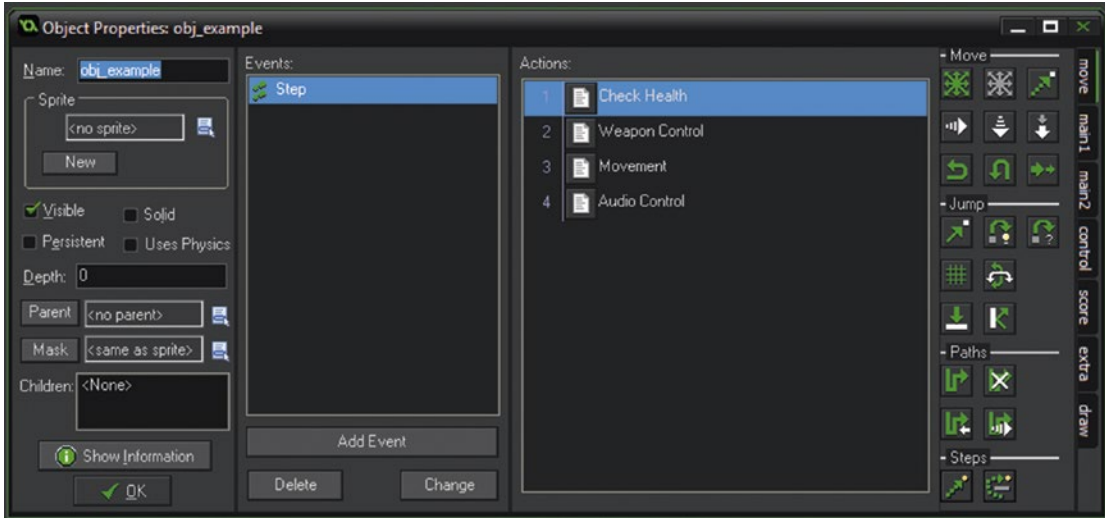
```
///Detect Mouse Movement
```

In game this would look like as shown in Figure i-25. This is a great to keep track of your code blocks, especially if you break them down into smaller sections.



**Figure i-25.** Naming a code block

You can also use separate code blocks together while making use of the triple `///`, for example, as shown in Figure i-26.



**Figure i-26.** Using multiple named code blocks

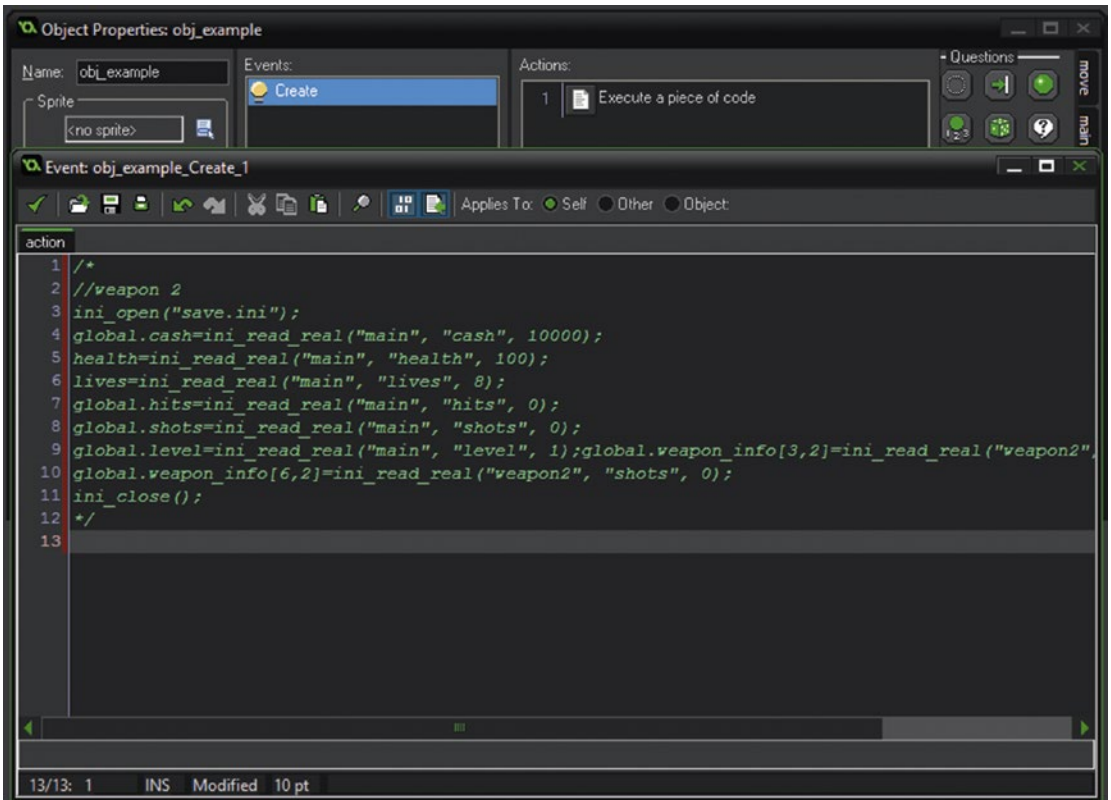
The third type allows you to comment out multiple lines. Any lines commented out will not be executed when the game is run.

You start this section with `/*` and end with `*/`.

For example:

```
/*
//weapon 2
ini-open("save.ini");
global.cash=ini-read_real("main", "cash", 10000);
health=ini-read_real("main", "health", 100);
lives=ini-read_real("main", "lives", 8);
global.hits=ini-read_real("main", "hits", 0);
global.shots=ini-read_real("main", "shots", 0);
global.level=ini-read_real("main", "level", 1);global.weapon_info[3,2]=ini-read_
real("weapon2", "bullets", 10000);
global.weapon_info[6,2]=ini-read_real("weapon2", "shots", 0);
ini-close();
*/
```

This would look like that shown in Figure i-27:



**Figure i-27.** Commenting out multiple lines

As before, any commented out code will not be executed.

You should now be aware of the basic elements that make up a game, and how to add GML.

# CHAPTER 1



# Variables

This section deals with the two main variable types: strings and numbers (also known as real values). You need to learn the different types, what you can do with them, how to combine them, and how to draw them on the screen.

Variables are an important part of every game. You can use variables for things such as the following:

- Keeping track of score, health, and lives
- Processing data
- Performing math's functions
- Moving objects
- Drawing data / text on screen
- Keeping track of a player's progress
- Making a game easier / harder

---

■ **Note** There are a number of variable types. The ones focused on in this book are built-in, instance, and global.

---

Built-in variables include **health**, **score**, and **lives**. These are automatically global in nature and can be accessed by any other object. User-defined global variables start with `global:` for example, `global.weapon`, which can also be accessed by any other object. You'll learn more about instance and global variables, and how to use them as you work through this book.

Instance variables relate only to the instance using it: for example `x` and `y`, and `size`.

The basic code for drawing text is:

```
draw_text(x_position, y_position, text);
```

An example would be:

To draw text:

```
draw_text(100, 100, "Hello World");
```

---

**Electronic supplementary material** The online version of this chapter (doi:[10.1007/978-1-4842-2373-4\\_1](https://doi.org/10.1007/978-1-4842-2373-4_1)) contains supplementary material, which is available to authorized users.

To draw a variable with a number (real value), for example:

```
weight=250;  
draw_text(100, 120, weight);
```

Create an object, **obj\_example\_1**, add a **Create Event** by clicking **Add Event**, followed by **Create Event**.

Add the following GML to the **Create Event**, and drag it across the **Execute Code** action entering the following with your own name:

```
example_text="My Name Is Ben";
```

Create a **Draw Event** and add the following code. To do this, select the **Draw Event**, then draw, and then drag across the **Execute Code** action.

```
draw_text(200,200,example_text);
```

Click the OK button in the bottom left to apply changes. This will draw the value of **example\_text** at the screen position 200,200, with 0,0 being at the top left. An example is shown in Figure 1-1:

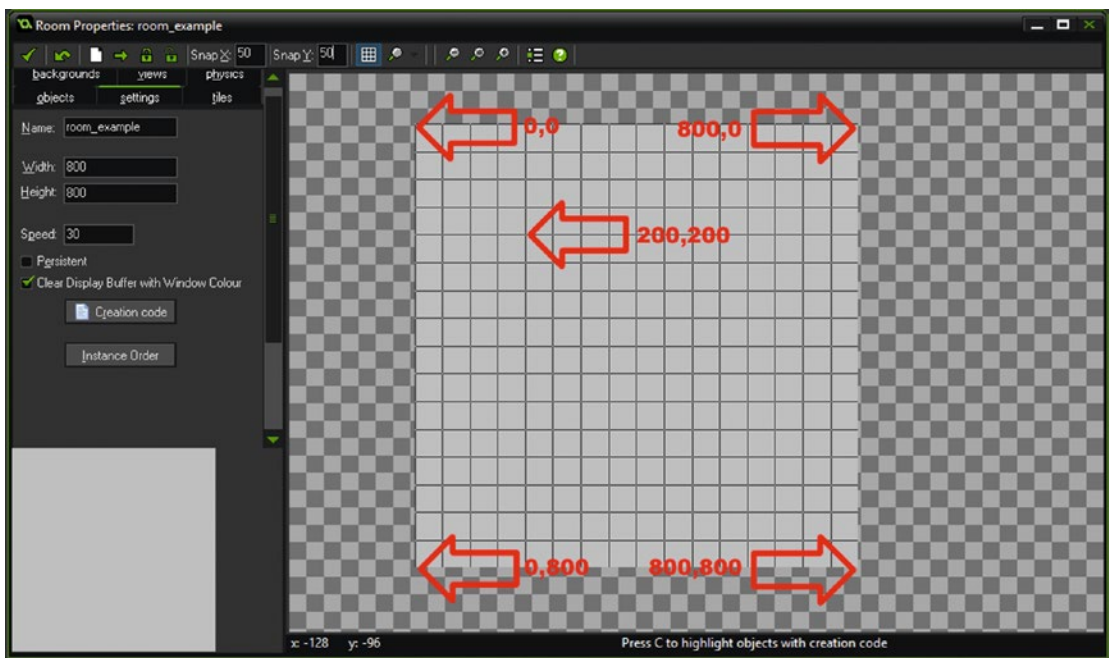


Figure 1-1. Showing various locations in a room

Create a room and place one instance of this object in the room. Do this by clicking the **Create a Room** button at the top of the screen. In the room editor, in the settings tab, set the name as room\_1, click the object tab, and then click in the room to create an instance. Close the room, click **File** and **Save As**, and then give the project the name **example 1**.

Real numbers can be whole integers: for example, 20; or include decimals, for example, 3.14.

Double-click on **obj\_example\_1** in the resource tree. In the **Create Event**, double-click on the code icon and add the following code on the next page to the bottom, then click the green tick on the top left:

```
my_age=21;
```

Then add the following to the **Draw Event**:

```
draw_text(100, 120, my_age);
```

Save and test.

You can add strings together using concatenation:

```
first_name="Samuel";
last_name="Raven";
my_name=first_name+" "+last_name;
```

You can add strings together, like in the example above.

You can do mathematical operations on numbers:

```
cakes=8;
cost=5;
total_cost=cakes*cost;
```

You can perform mathematical calculations on real numbers, for example, +, -, \*, and /.

GameMaker also allows use of other operators such as mod and div.

For example:

```
a=20;
b=7;
```

Where:

$c = a \text{ mod } b$ ; would set  $c$  as 6; ( $b$  goes into  $a$  twice with a remainder of 6).

$c = a \text{ div } b$ ; would set  $c$  as 2 ( $b$  goes into  $c$  2 times).

You can generate random numbers using a number of functions:

```
number=irandom(20);
```

The above would give an integer (a whole number) between 0 and 20 inclusive (any number in the given range can be used).

To make testing easier, GameMaker: Studio will create the same sequence of numbers. You can override this setting by using the following code:

```
randomize();
```

This only needs to be performed once, for example, in the room creation code.

You cannot add together numbers and strings directly. For example, this would create an error:

```
example_text="My age is:";
my_age=17;
name_and_age=example_text+my_age; //This Line Creates an Error
```

You can convert a number to a string using this:

```
name_and_age=example_text+string(my_age);
```

This works, as it converts the number to a string and adds to the other string.

```
draw_text(50,50,name_and_age);
"My age is: 17" at position 50,50.
```

Equally, you can change a string to a variable; however it will cause an error only when it doesn't correspond to a number. For example "-5" and "2.45" consist of more than just numbers, but `real()` can process them fine.

```
a="3.14 ben";
b=real(a);
```

Would set a b as 3.14.

**Extra Useful Code:**

You can get a user to enter integer/string with this:

```
age=get_integer("Age? ", 1);
name=get_string("Name? ", "Enter Your Name Here");
```

---

■ **Note** These functions above should really be only used for testing purposes and are fine for beginners. As you advance, you should use `get_integer_async` and `get_string_async`. There is an example for each of these in the manual.

---

Variables can also be set to **true** or **false** (which also return as **1** or **0**, respectively, but you should really always try to use the built-in constants, **true** and **false**).

There are also built-in constants and you can also define your own as macros.

You should now be aware of the two main types of variables: first, numbers, such as these:

```
age=10;
pay_per_hour=2.17;
bonus=5000;
```

And second, strings, such as these:

```
name="Ben";
level_name="Dungeon";
food="Cheese";
date="Twentieth";
```



## Worksheet – Variables

1. Circle the correct definitions of these strings and numbers:

"Ben" string / number

"thirty-two" string / number

32 string / number

"56.5" string / number

39.4 string / number

"Ben is 21" string / number

"London" string / number

"First" string / number

"nine plus 4" string / number

Using the following values:

a=5;

b=10;

c=20;

Work out answers to the following, and then check them in GameMaker.

a+b+c

c div a

(a\*b)+c

b mod a

(b+c) div a

2. Given the following variables:

age=21;

month="April";

year="2006";

money=2716;

place="London";

time="2:30";

book="Harry Potter";

dollars=98.55;

day="Tuesday";

Which of the following is allowed? Write the correct code if wrong.

details=book+" is "+string(age);

location=place+" at "+time;

date=year+month;

cash=age+money;

payment=day+dollars;

birthday=day+" "+month+" "+year;

celebrate\_at=place+" "+string(age);

something=real(year)+money;

total=age\*(money/real(year));

# Worksheet – Variables – Answer Sheet

1. Circle the correct definitions of these strings and numbers:

"Ben" **string**

"thirty-two" **string**

32 **number**

"56.5" **string**

39.4 **number**

"Ben is 21" **string**

"London" **string**

"First" **string**

"nine plus 4" **string**

Using the following values:

a=5;

b=10;

c=20;

Work out answers to the following, and then check them in GameMaker.

a+b+c **35**

c div a **4**

(a\*b)+c **70**

b mod a **0**

(b+c) div a **6**

2. Given the following variables:

age=21;

month="April";

year="2006";

money=2716;

place="London";

time="2:30";

book="Harry Potter";

dollars=98.55;

day="Tuesday";

Which of the following is allowed? Write the correct code if wrong.

---

■ **Note** In some cases, more than one answer is possible; for example, you could turn a string into a real and add to a real, or convert the real to a string and add to a string.

---

```
details=book+" is "+string(age); correct  
location=place+" at "+time; correct  
date=year+month; correct  
cash=age+money; correct  
payment=day+dollars; incorrect - day+string(dollars);  
birthday=day+" "+month+" "+year; correct  
celebrate_at=place+" "+string(age); correct  
something=real(year)+money; correct  
total=age*(money/real(year)); correct
```

## Basic Projects

- A) Make a program that takes in name, age, and date of birth and displays it on the screen.  
1 Point for attempting this question - 1 Point for making a working example  
1 Point for using good variable names and tidy GML formatting
- B) Make a program that takes in five numbers and calculates the average.  
1 Point for attempting this question - 1 Point for using good variable descriptions

## Advanced Project

- C) Make a program where you enter the date and the program displays correct tag, like 1st, or 23rd.  
1 Point for attempting this question  
1 Point for good formatting  
2 Points for using their own data input system - 1 Point for displaying output nicely on screen

### PROJECTS NOTES

```
if (number mod 2==0)
{
    // will draw if number is even
    draw_text(50, 50, string(number)+ " is even");
}
if (age==20)
{
    // will draw "You Are Twenty" if age is equal to 20
    draw_text(50,50, "You Are Twenty");
}
```

Conditional statements are used to check and compare variables (and other values such as instance ids, if sounds are playing, keypresses, mouse position and more).

---

## End of Book Game Variables

The game that you'll create at the end of the book will use a fair amount of global and instance variables.

They'll be global variables for things such as the current level, **score**, **cash**, **lives**, **health**, and current weapon; and for weapon info such as the name of the weapon, its power, cost, and current quantity.

The asteroids and enemy will have their own custom health that will reduce when hit by a player's bullet.

The player will control a ship and be able to use a keypress to fire the currently selected weapon, if that weapon has ammo available. If there is, it will create a bullet and reduce the weapon count.

Asteroids will have instance variables for its **hp**, **speed**, and **direction**.

When the game starts it will look for saved data from the last time the game was played. If there is data to be read, the game will load all of the variables into memory to be used.

Upon running the game, the player will be briefly presented with a loading splash screen, and then be taken to the level select screen. There will be a number of buttons to play levels 1 to 4. Each button will check what the current value of the global level is, in order to allow the player to only play levels that have been unlocked. When the player runs the game the first time, only level 1 will be unlocked, level 2 will only unlock once the player has successfully completed level 1, and so on with levels 3 and 4. When a player successfully completes a level, all the variables relating to the player and weapons will be saved, so upon running the game again the player can continue from where they left off.

A control object will display the vital variables at the top of the screen, using a combination of graphics and text. It will also monitor the values of **health** and **lives**; if the player runs out of **health**, having crashed into too many asteroids, the player will lose a life. If the player loses all **lives**, the game is over.

Think about what variables will be needed to accomplish this. Think of suitable names for the variables, and whether they should be instance or global variables.

## CHAPTER 2



# Conditionals

Conditional statements are used to check and compare variables (and other values such as instance ids, if sounds are playing, keypresses, functions, mouse position, and more).

Therefore, conditional statements will be used often. Having a strong understanding of them is very important. Conditionals can combine with other functions. Conditionals, or combinations of them, can be used to make things happen (or not happen). For example:

- Make a ball bounce when it hits a wall
- Make an enemy fire a bullet if it can see the player
- Play sound effects when an object loses some of its **health**
- Unlock a level if a **score** is met
- Make a player move if a mouse button or key is pressed
- Detect the middle mouse button to change a weapon
- See if a player has enough cash to buy an upgrade
- Check if a player is jumping or not
- Create an effect if a weapon is fired, etc.

Explained in the most basic sense, conditionals evaluate expressions, and will execute and perform actions accordingly. For example, taking the following values:

```
a=3;  
b=2;  
c=5;
```

Would give the following results:

(a+b)==c returns **true**.

(a==b) returns **false**.

---

■ **Note** Use == when using conditionals, rather than a single =.

Actual code will look like this:

```
if (a+b)==c  
{
```

```

    //do something if true
    show_message("true");
}
else
{
    // do something if false
    show_message("false");
}

```

---

You can add **!**, which means **not**. So **!** is an expression that negates a logic sentence. So a true sentence turns into a false sentence, and a false sentence turns into a true sentence:

`!(a==b)` returns **true** if a is not equal to b.

You can test if a sound is playing or not:

```

if audio_is_playing(snd_background_music)
{
    //do something
}

```

You can test the pressing of a mouse button:

```

if (mouse_check_button_pressed(mb_left))
{
    //do something
}

```

You can also check for keyboard presses, for example:

```

if keyboard_check_pressed(ord('Q'))
{
    //Do something here
}

```

`ord` is a function that identifies a keypress of letters and numbers in a way that GameMaker: Studio can understand. This is known as virtual keycodes, and also includes a series of constants starting with `vk_`.

Variables can also be set to **true** or **false**:

```

answer=true;
alive=false;

```

so:

```
if (answer)
{
    //Do Something
}
```

would perform any code between { and }.

```
if (alive)
{
    //do something first part
}
else
{
    //do something second part
}
```

would not execute the first part, but it would execute the second part.

You can also use operands and mathematical comparisons when checking a conditional:

```
a=3;
b=2;
c=5;
```

(a < b) returns **false**, (c > b) returns **true**.

You can also use <= to check if a value is equal to or less than, and >= to check if a value is equal to or greater than.

You can use the following logic operators, && and and for and, || and or for or. For example, the following will execute code if A and the right arrow are pressed:

```
if (keyboard_check(ord('A'))&& keyboard_check(vk_right))
{
    //do something if A and right arrow is pressed
}
```

The following will check either, so it will execute any code if A is pressed or the right arrow is pressed or both are pressed:

```
if (keyboard_check(ord('A'))|| keyboard_check(vk_right))
{
    //do something if A or right arrow is pressed (or both)
}
```



## Worksheet – Conditionals

Given the following variables:

```
first_name="Bob";
age=24;
city="London";
surname="Scott";
distance=48
seconds=50;
friend=true;
enemy=false;
time=7;
full_name="Bob Scott";
xx=50;
```

Determine if the following are true or false:

<code>(age&gt;20)</code>	<b>true / false</b>
<code>!(first_name=="Bob")</code>	<b>true / false</b>
<code>(distance&lt;time)</code>	<b>true / false</b>
<code>(!friend)</code>	<b>true / false</b>
<code>(enemy)</code>	<b>true / false</b>
<code>((first_name+" "+surname)==full_name)</code>	<b>true / false</b>
<code>(city=="Paris")</code>	<b>true / false</b>
<code>(time&lt;seconds)</code>	<b>true / false</b>

## Worksheet – Conditionals – Answer Sheet

Given the following variables:

```
first_name="Bob";
age=24;
city="London";
surname="Scott";
distance=48
seconds=50;
friend=true;
enemy=false;
time=7;
full_name="Bob Scott";
xx=5;
```

Determine if the following are true or false:

(age>20)	<b>true</b>
!(first_name=="Bob")	<b>false</b>
(distance<time)	<b>false</b>
(!friend)	<b>false</b>
(enemy)	<b>false</b>
((first_name+" "+surname)==full_name)	<b>true</b>
(city=="Paris")	<b>false</b>
(time<seconds)	<b>true</b>

## Basic Projects

- A) Create a password system where the user has to enter a correct password to continue.  
1 Point for a working example
- B) Display an object at a random position on the screen for one second. Player must then click where the object appeared. Award points depending on how close the player clicked.  
2 Points for a working example

## Advanced Projects

- C) Create a simple text input system using keypresses. Allow the user to enter their name. Then store as `global.name` when enter is pressed.  
Project Note: Look up usage of `keyboard_string`  
3 Points for a working example
- D) Make a game where the player selects a number at random from 1 to 100. Player enters a value (for example 25) and the game will tell you if you are too high, too low, or correct.  
2 Points for a working example  
2 Points for good code formatting

## End of Book Game Conditionals

The game you'll make at the end of the book will contain lots of conditionals, and they'll be used for such things as the following:

- Detecting presses of arrow keys to move the player's ship
- Selecting and firing weapons through keypresses
- Checking whether the player has enough ammo to fire a bullet
- Checking a player's **health** and **lives**
- Allowing the purchase of a weapon in the shop if the player has enough money
- Checking if a level is unlocked and clickable
- Checking if an asteroid exists for the homing missile weapon to lock on to
- Checking asteroids and enemy **health**
- Reducing the alpha (transparency) of a sprite to make the weapon fade away
- Detecting mouse clicks on buttons, such as play game from the shop
- Deleting a save file if present
- Destroying a cloud object if it's finished moving
- Allowing something to happen/not happen while something is true or false

Have a go at trying to plan out what some of the code for the above conditionals may look like.

All conditionals follow the same premise: if something is true, do something or don't do something. Or if something is not true, do something or do not do something else.

## CHAPTER 3



# Drawing

GameMaker: Studio has a number of built-in functions for drawing. These include setting drawing colours, setting text fonts, and drawing geometric shapes.

In the most basic terms, drawing items uses an X Y positional system. X relates to pixels across from the top left, Y the number of pixels down from the top. Drawing can be relative to the room position or a view. This and the next section assume drawing in a standard room using default room settings without the use of views. See Figure 1-1 in Chapter 1 for an explanation of coordinates.

This section serves as an introduction to drawing basic shapes on the screen and familiarization with using X and Y coordinates.

Basic geometric shapes are useful for the following:

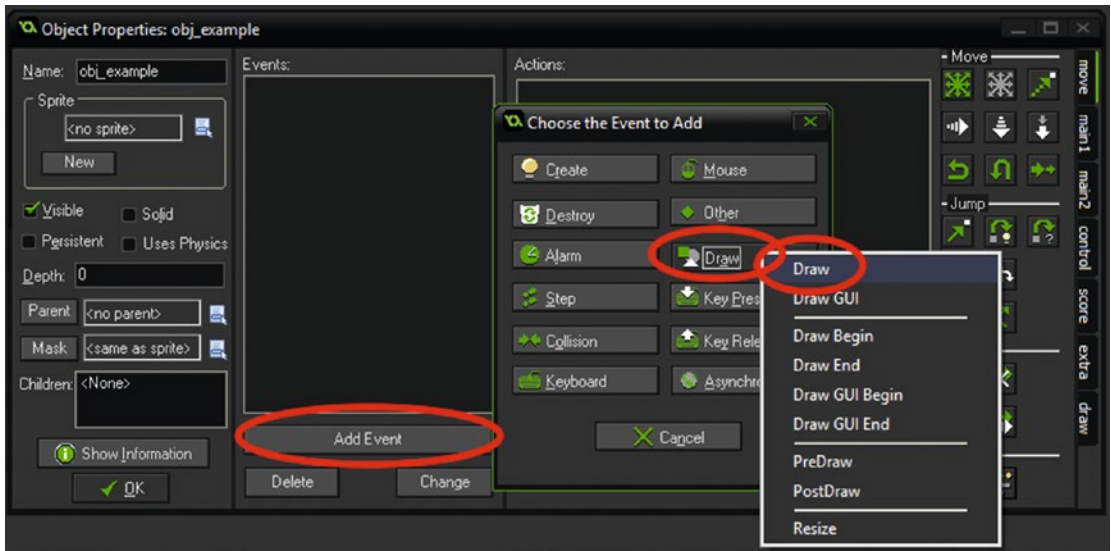
- Drawing a room border
- Creating pop-up boxes
- Creating room transitions
- Creating effects
- Drawing shadows of objects

---

■ **Note** Due to YYG being a British company, the spelling used is *colour*, though *color* can also be used.



















---

Drawing code must be placed in a **Draw Event**. There are several options available, but for now we'll just use the main Draw Event. Figure 3-1 shows how to select this, and the options available.



**Figure 3-1.** Showing how to select draw event

Colour constants have built-in values:

Colour	Appearance	RGB Value
c_aqua		0,255,255
c_black		0,0,0
c_blue		0,0,255
c_dkgray		64,64,64
c_fuchsia		255,0,255
c_gray		128,128,128
c_green		0,128,0
c_lime		0,255,0
c_ltgray		192,192,192
c_maroon		128,0,0
c_navy		0,0,128
c_olive		128,128,0
c_orange		255,160,64
c_purple		128,0,128
c_red		255,0,0
c_silver		192,192,192
c_teal		0,128,128
c_white		255,255,255
c_yellow		255,255,0

The following code can be used to set a drawing colour:

```
draw_set_colour(c_blue);
```

Colour can also be set using hexadecimal values prefixed with a '\$' character, which in GameMaker: Studio is in the format BBGRRR:

```
draw_set_colour($2391FF);
```

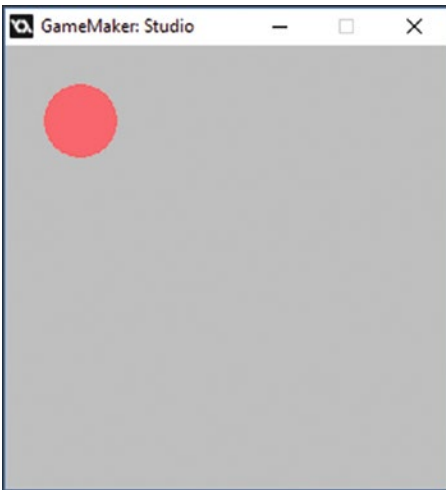
Or you can set the colour by setting each colour channel:

```
colour=make_colour_rgb(25,120,98);
```

You can also set the colour using RGB and saving this as a user-defined variable. Obviously, any value for `make_colour_rgb` should be in the range of 0 to 255. For example:

```
my_colour=make_colour_rgb(240, 90, 100);
draw_set_colour(my_colour);
draw_circle(50, 50, 25, false);
```

The above example would draw the following on screen as shown in Figure 3-2, when the object is placed into a room; note the drawing code needs to be in a **Drawing Event**.



*Figure 3-2. Showing circle having been drawn*



Which will draw a red circle at position 50,50 with a radius of 25 and using **false** draws as a solid circle. If you were to use **true** it would only draw the outline.

This code would draw a line from position 100,100 to 200,200 in blue:

```
draw_set_colour(c_blue);  
draw_line(100, 100, 200, 200);
```

The following will draw a solid gray rectangle from 5,5 to 110,110. The last **false** sets the rectangle to be filled in. Using **true** would draw the outline only.

```
draw_set_colour(c_gray);  
draw_rectangle(5, 5, 110, 110, false);
```

Other drawing functions that you can use include (again **true** or **false** draws filled or border only), for example:

```
draw_ellipse(x1, y1, x2, y2, true); //draw an ellipse with outline  
draw_point(x, y); // draws a single pixel  
draw_roundrect(x1, y1, x2, y2, false); //draws a solid rounded rectangle  
draw_line_width(x1, y1, x2, y2, width); //draws a line of given width  
draw_triangle(x1, y1, x2, y2, x3, y3, false); //draws a solid triangle
```

If you're looking for something more advanced, you can look up primitives in the manual. You can open the manual by pressing F1 in GameMaker: Studio.

## Worksheet – Drawing

1. Draw each of the following on graph paper, and then write what the picture is. Then enter the GML code into a **Draw Event** of an object and see if you drew it correctly.

Picture 1:

```
c_face_colour=make_colour_rgb(204, 153, 0);
draw_set_colour(c_face_colour);
draw_circle(50, 50, 40, 0);
draw_set_colour(c_white);
draw_circle(40, 40, 5, 0);
draw_circle(60, 40, 5, 0);
draw_set_colour(c_red);
draw_ellipse(30, 70, 70, 80, 0);
```

Picture 2:

```
draw_set_colour(c_red);
draw_rectangle(30, 30, 90, 70, 1);
draw_rectangle(35, 40, 45, 50, 1);
draw_rectangle(75, 40, 85, 50, 1);
draw_rectangle(50, 50, 70, 70, 1);
draw_line(30, 30, 40, 10);
draw_line(40, 10, 80, 10);
draw_line(80, 10, 90, 30);
```

## Worksheet – Drawing – Answer Sheet

1. Draw each of the following on graph paper, and then write what the picture is. Then enter the GML code into a **Draw Event** of an object and see if you drew it correctly.

Picture 1 is a smiling face as shown in Figure 3-3:



**Figure 3-3.** Smiling face made with picture 1 code

Picture 2 is a house as shown in Figure 3-4:



**Figure 3-4.** House made with picture 2 code

## Basic Projects

- A) Draw a grid of black and white squares, suitable for playing chess or checkers on.  
3 Points
- B) Create a floor plan of the classroom; include furniture, windows, and doors (use different colour for each).  
3 Points

## Advanced Project

- C) Draw a picture of the *Mona Lisa* or one of Piet Mondrian's paintings using basic drawing.  
4 Points

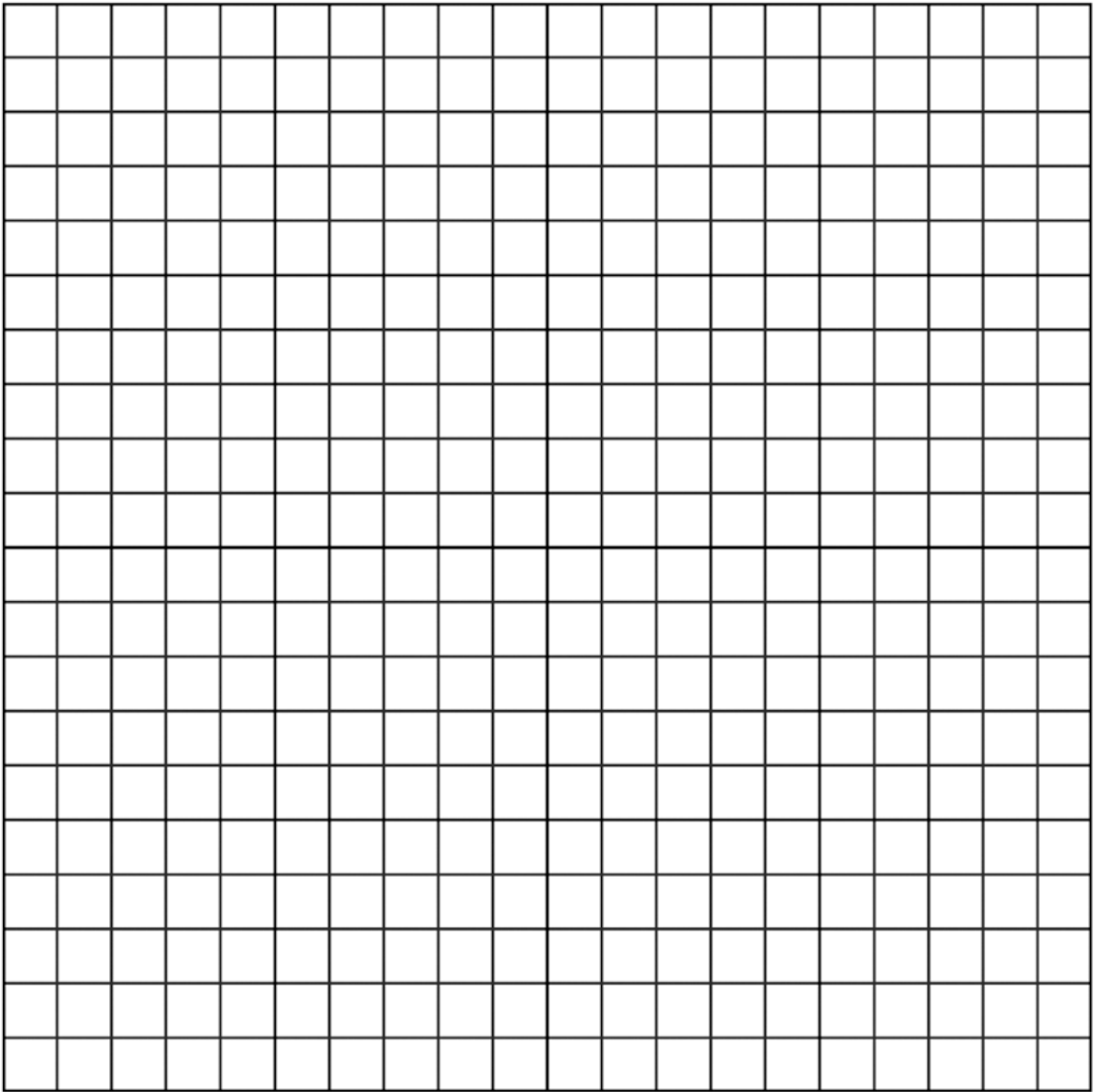
---

Note on projects for chapter 3:

---

It's also possible to draw a sequence of connected lines using primitives.  
For example:

```
draw_primitive_begin(pr_linestrip);  
draw_vertex(50,50);  
draw_vertex(150,50);  
draw_vertex(50,150);  
draw_vertex(250,50);  
draw_vertex(50,250);  
draw_primitive_end();
```



**Figure 3-5.** *Graph sheet for drawing on*

Scale: 1 Square=\_\_\_ Pixels

## End of Book Game Drawing

The game will utilize some basic drawing for the HUD. There'll be a space at the top of the screen for displaying current weapon, stats, and lives. The HUD will consist of a solid rectangle at the top, with another rectangle to display health. Lives will be drawn as images using the D&D Draw lives action. Text will be used to draw the weapon type, weapon strength, current ammo, total weapon fired (both current and all weapons), and cash.

In the bottom corner of the screen will be radar. The radar's scope will be drawn using circles and lines.

There will also be a room border.

The advantage of using the drawing functions is that while testing, it allows you to quickly change the colour, size, and shape of the area being drawn. If you were to use a sprite or background, you'd have to constantly edit the graphics file.

## CHAPTER 4



# Drawing Continued

There are a number of other functions for drawing images and variables. These can be used separately or combined to create a number of effects. In any game, you're likely to have a number of sprites and information you want to display on the screen.

For example, images can be used for drawing:

- The player
- Missiles and bombs
- Menu buttons
- Walls and platforms

Text can be used for:

- Scores and health
- Player names
- Game information
- Pop-up text
- Game timer

---

■ **Note** Only try to draw the value of a variable if it has already been declared in the **Create Event** or prior to drawing it; failing to do so may cause an error. Built-in variables are OK to draw without being declared.

---

Create a new project in GameMaker: Studio, along with a new object.

To use drawing functions, they need to be placed within a **Drawing Event**. Create a **Draw Event** for the object you just created.

```
draw_text(100, 100, "Hello World! ");
```

This will draw the **Hello World!** sentence in the room at position 100,100 – where this position is the top-left corner of this drawn text.

You can also include strings and reals, by converting the real to a string:

```
age=20;
draw_text(100, 100, "I am "+string(age)+ " years old");
```

This will draw the text ***I am 20 years old*** sentence on the screen. You can format text too:

- Use a different font
- Use a colour
- Have different horizontal and vertical alignment

Save the code you just wrote, and close the object. Create a new font using the Resources menu at the top of the screen. Give the font a name, something like **font\_Myfont**, and select a better-looking typeface, for example, **Calibri**.

Resize the font to about 30 pixels, and make it bold, so the user can see it better. Now, save the font and return to your object's **Draw Event**.

Formatting functions need to be applied before drawing text, and they can be applied in any event; however, the best practice is to set drawing directly before drawing any text. This can be in code or by calling a script you've set up. For example you can set font, colour, and alignment:

```
draw_set_font(font_Myfont); //Use this font for drawing text
draw_set_colour(c_blue); //Make the text blue
draw_set_halign(fa_center); //Center the text to the x position
draw_set_valign(fa_middle); // Center the text vertically to the y position
```

---

■ **Note** When you apply formatting, it will remain in place **for all objects**; ideally you should set the formatting right before any drawing code.

---

Now, the text will be significantly bigger, since you created a bigger font. It will also appear blue, and its position will be changed because of the horizontal and vertical alignment settings.

Here are some more arguments that you can use with the alignment functions.

For horizontal alignment: `fa_left`, `fa_center`, `fa_right`

For vertical alignment: `fa_top`, `fa_middle`, `fa_bottom`

Notes:

- You can insert a new line using the # symbol.
- If you need an actual # symbol, you can do \#.
- If you want to draw a value of a variable that is not a string, use the `string()` function before the variable name, and this will convert it into a string. This will allow you to combine strings and real. If you are drawing just a real, you do not need to convert to a string, though it's best practice to always convert reals to strings for drawing.
- When you apply drawing formatting, like font, colour, or alignment, it will apply to all drawing, including other objects, until you change to something else. For this reason it is a good idea to apply formatting right before you do any drawing.

For example, you do the following code in the **Create Event** of an object, **obj\_example**:

```
name="Ben";
age=28;
country="England";
food="Pizza";
```



Which would look like Figure 4-1:

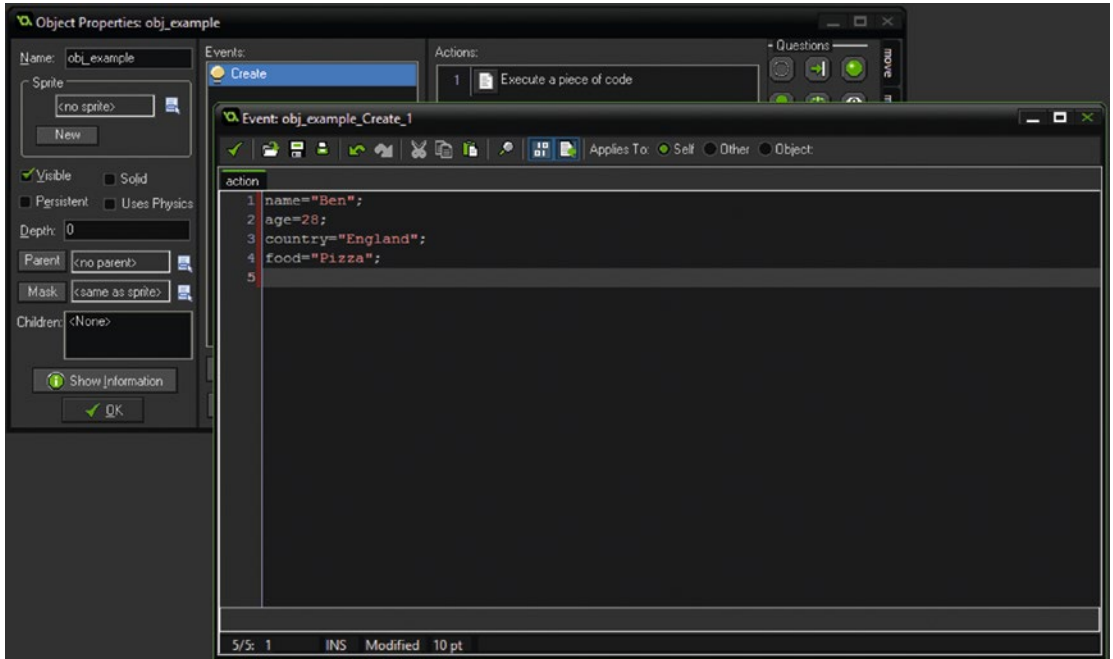


Figure 4-1. Showing create event

You can then set a font, for example, as shown in Figure 4-2:



Figure 4-2. Setting a font

You can then apply the settings and draw the text on screen, by putting the following code in the **Draw Event** of **obj\_example**:

```
draw_set_font(font_example);
draw_set_halign(fa_center);
draw_set_valign(fa_middle);
draw_set_colour(c_red);
draw_text(300,200,"His name is "+name+".#He is "+string(age)+" years old.#He lives in "+country+".#His favourite food is "+food+".");
```

Create a room, **room\_example**, and place one instance of **obj\_example** in it. When run, you will see that shown in Figure 4-3:



*Figure 4-3. Showing example output*

There is a GMZ for this example in the resources: **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 4 ► example 1**

Create a new project. Load a sprite into the same project as before, and give it a practical name, something short like **spr\_test**.

Our goal is to draw this sprite in a few different ways, so open up the **Draw Event** inside your object, and use this code to draw a normal sprite on the screen.

```
draw_sprite(spr_test, 0, 100, 200);
```

This will draw the sprite **spr\_test**, using sub image 0 and position 100, 200.

Sub image refers to which frame of the sprite to use. A sprite can have 0 (which can be useful tool in certain circumstances), 1, or multiple sub images. They can be used for animations, or to show a different image when facing different directions or performing an action like shooting or climbing a ladder.

If you run the game now, you will see your sprite at the 100,200 position, but what if we want to make the sprite look different? For extra formatting options, use the `draw_sprite_ext` function:

```
draw_sprite_ext(sprite, sub image, x, y, xscale, yscale,
rotation, colour, 1);
```

The above is used when you want more flexibility in drawing the sprite. It may also be used to draw the default sprite. It will draw the `sub image` frame, at the given `x` and `y` location, while `xscale` and `yscale` set its size, 1 is 100% size, 0.5 would be half size, 2 would be double size. Rotation changes the angle of the image counterclockwise. Colour blends the image colour. An example using `draw_sprite_ext()`; would be the following, which would draw the sprite **spr\_enemy**, sub image 0, at position 180,120, 50% larger, rotated 25' counterclockwise with a reddened colour:

```
draw_sprite_ext(spr_enemy, 0, 180, 120, 1.5, 1.5,
25, c_red, 1);
```

The colour blending can be used to great effect to give a visual reference of something happening. For example, blending with red can visualize that the enemy has been hit by a bullet.

If your sprite has just one sub image, and no other drawing actions, you don't need to add anything in the **Draw Event** as the sprite will be automatically drawn. If you are drawing text or want to draw multiple sprites from a single object and your object has a sprite, you can add this:

```
draw_self();
```

If using `draw_self()`; you may want to manually set which sub image (if you have multiple sub images). You can do this using:

```
image_index=1;
image_speed=0;
```

Which would set the sub image **1** as the sub image to be drawn. Sub images start with an index of **0**. Setting the image speed to **0** prevents it from automatically animating.

You can also set the speed the sub images will play at using, for example:

```
image_speed=2;
```

Which would set the animation speed at 2. The speed is a scalar value, so 0.5 will draw the same sub image for two steps, 0.25 for four steps, and that larger values like 2 will "skip" a sub image and only show every second sub image per step.

You can also set the angle of an image (its rotation). This can be a value between 0 and 359. For example:

```
image_angle=45;
```

You can set an object moving, for example the following will make the object move to the right at a speed of 2:

```
motion_set(0,2);
```

`draw_self()` is the same as `draw_sprite_ext()` using only all the default image variables, which is the same as letting GM default draw (i.e., no draw event defined, so GM draws the given sprite). You can use `draw_sprite_ext()` with the default settings, for example:

```
draw_sprite_ext(sprite_index, image_index, x, y, image_xscale, image_yscale, image_angle,
image_blend, image_alpha);
```

You can change the variables to change how the image is drawn.

---

■ **Note** Image blending works better with lighter sprites, and best with ones that are white.

---

For example, the following will stretch the sprite 80% on the length and by 120% on its height; rotate by 45 degrees and blend with the colour `c_blue`:

```
draw_sprite_ext(spr_example, 0, 200, 200, 0.8, 1.2, 45, c_blue,1);
```

Figure 4-4 shows a sprite drawn normally, and with the code above:



**Figure 4-4.** Showing sprite drawn normally, and with `draw_sprite_ext`

## Worksheet – Drawing Continued

1. True or False
  - A) You can draw a value of a variable when using the `draw_text` function. T / F
  - B) You should apply formatting changes before drawing the text. T / F
  - C) Only the horizontal alignment of the text can be changed. T / F
  - D) You can change the size and rotation of drawn sprites. T / F
  - E) You cannot change the colour of drawn sprites. T / F
2. What is the name of the function for advanced sprite drawing?
3. Look up each of the following in the help manual, and write an example for each:  
  
`draw_sprite_general`  
`draw_sprite_tile`  
`draw_text_transformed`
4. How would you do each of the following in GML:
  - A) Draw the 4th sub image of a sprite
  - B) Set the animation speed to 5
  - C) Rotate the sprite 180 degrees
  - D) Draw text in red size 20 in Arial.
  - E) Draw text right justified at position 140,80

## Worksheet – Drawing Continued – Answer Sheet

### 1. True or False

- A) You can draw a value of a variable when using the `draw_text` function. **T**
  - B) You should apply formatting changes before drawing the text. **T**
  - C) Only the horizontal alignment of the text can be changed. **F**
  - D) You can change the size and rotation of drawn sprites. **T**
  - E) You cannot change the colour of drawn sprites. **F**
2. What is the name of the function for advanced sprite drawing? `draw_sprite_ext()`;
3. Look up each of the following in the help manual, and write an example for each:

```
draw_sprite_general(spr_example, 1, 10, 10, 50, 100, 120, 150, 1, -1, 45,
c_white, c_white, c_red, c_red, 0.5);
draw_sprite_tiled(spr_brick, 0, 25, 25);
draw_text_transformed(200, 200, "Hello World", 3, 0.8, 270);
```

### 4. How would you do each of the following in GML:

- A) Draw the 4th sub image sprite `image_index=3`; `draw_self()`; or equally `draw_sprite(sprite_index, 3, x, y)`;
- B) Set the animation speed to 5 `image_speed=5`;
- C) Rotate the sprite 180 degrees `draw_sprite_ext( spr_example, 0, 100, 100, 1, 1, -90, c_white, 1 )`; or equally `image_angle=180`; `draw_sprite()`;
- D) Draw text in red size 20 in Arial Create a font **font\_example**,
 

```
draw_set_font(font_example);
draw_set_colour=c_red;
```
- E) Draw text right justified at position 140,80 `draw_set_halign(fa_right)`; `draw_text(140,90, "example")`;

Any other suitable code is also a valid answer

## Basic Projects

- A) Make a program that draws a rotating sprite.  
2 Points
- B) Make a program that writes a formatted message on the screen. Set a font type, colour, and alignment.  
2 Points

## Advanced Projects

- C) Make a program that draws randomly positioned cloud sprites moving to the left at various speeds, with varying size and opacity (alpha).  
2 Points
- D) Get user to enter their name. Draw this on the screen, formatted, moving from the top of screen to the bottom. Destroy the object when it reaches the bottom  
4 Points

## End of Book Game Drawing Continued

The game will involve drawing various graphics and text.

Using text, for example:

- Basic stats on the level select screen
- Info on weapons in the shop
- A HUD in game to display stats and weapon info
- Displaying various variables for testing purposes
- Testing purposes

We'll use a selection of different fonts and font sizes for the text.

Sprites will be used for:

- Buttons that the player clicks to start levels
- Buttons to quit and restart game
- Buttons
- Asteroids
- Ship
- Bonus items
- Lives image
- Radar blips



## CHAPTER 5



# Keyboard Input and Simple Movement

When playing your game, players will need some method of interaction with the sprites on the computer screen. The main types you're likely to use are keypress, mouse movement, and mouse buttons. This section deals with keyboard and mouse input.

You can use this input to make things happen, for example:

- Make a player move
- Fire a weapon
- Add text to string
- Create an effect
- Execute other GML code

Keypresses can be detected using GML in the **Step Event**. In its basic form, GML uses an expression and performs accordingly.

A very basic example that would perform while X is being pressed down:

```
if (keyboard_check(ord('X')))  
{  
    //do something  
}
```

---

■ **Note** Keyboard letters, that is, 'X' must be in capital when used with `ord`.

---

In addition to `keyboard_check`, you can also use `keyboard_check_pressed`. Note that there is a **strong distinction** between these two:

`keyboard_check` checks whether the key is currently being pressed.

`keyboard_check_pressed` checks whether the key has just been pressed.

Another option is detecting when a key is released:

`keyboard_check_released`

As well as keypresses, you can detect mouse button presses, also in the **Step Event** for example. As with `key_check`, there is a difference between `mouse_check_button`, `mouse_check_button_pressed` and `mouse_check_button_released`:

```
if (mouse_check_button(mb_left)) // Checks if left mouse button is being held down.
{
    // do something
}
```

You can move an object by changing its x and y positions. x is the position in pixels across the screen, and y is how many down.

For example, you could put the following into a **Step Event**:

```
if (keyboard_check(ord('A'))) {x-=5;}
if (keyboard_check(ord('D'))) {x+=5;}
if (keyboard_check(ord('W'))) {y-=5;}
if (keyboard_check(ord('S'))) {y+=5;}
```

or

```
if (keyboard_check(vk_left)) {x-=5;}
if (keyboard_check(vk_right)) {x+=5;}
if (keyboard_check(vk_up)) {y-=5;}
if (keyboard_check(vk_down)) {y+=5;}
```

You can also use Boolean values as multipliers, since a value of false will return **0**, and a value of **1** will return true, but this can be a bit confusing at first. The following allows you to move an object with key presses. `vk_right` is the built-in constant for the right-arrow key; it will return as **true** when the right-arrow key is pressed. The same applies for the other arrow keys:

```
x+=5*(keyboard_check(vk_right)-keyboard_check(vk_left));
y+=5*(keyboard_check(vk_down)-keyboard_check(vk_up));
```

See **Reference ► Mouse, Keyboard and Other Controls ► Keyboard Input** in the GameMaker: Studio manual for more keycodes.

You can also get the value of the last key that has been pressed with

```
keyboard_lastkey
```

Using `keyboard_lastchar` example, you make a string of what has been typed.

In the **Create Event** of an object, **obj\_example** put:

```
typed="";
```

In the **Step Event** place:

```
typed=typed+keyboard_lastchar;
keyboard_lastchar="";
```

And in a **Draw Event** put:

```
draw_text(100,100,typed);
```

Put this object in a room and then test it.

There is an example for the above in the resources folder:

**Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 5**

## Worksheet – Key Presses and Simple Movement

Which of the following are correct? Write a correct version if incorrect.

```

if (keyboard_check(vk_left))
if (keyboard_check(right))
if (keyboard_check(shift)&&keyboard_check(ord('X')))
if (keyboard_check(spacebar)||keyboard_check(vk_enter))
if (keyboard_check(ord('1'))
if (keyboard_check(ord('p')))
if (keyboard_check_released(vk_up))
if (keyboard_pressed_check ('vk_backspace'))
if (keyboard_check('vk_right') || keyboard_check(ord('d'))

```

### Write code for the following:

If J is being pressed:

If Left arrow is released:

If Shift and U are being pressed:

If 1 and 8 are being pressed:

Enter has just been pressed:

## Worksheet – Key Presses and Simple Movement – Answer Sheet

Which of the following are correct? Write a correct version if incorrect.

```

if (keyboard_check(vk_left)) correct
if (keyboard_check(right)) incorrect if (keyboard_check(vk_right))
if (keyboard_check(shift)&& keyboard_check(ord('X'))) incorrect if (keyboard_check(vk_shift)
&& keyboard_check(ord('X')))
if (keyboard_check(spacebar)||keyboard_check(vk_enter)) incorrect if (keyboard_check(vk_
space) || keyboard_check(vk_enter))
if (keyboard_check(ord('1')) correct
if (keyboard_check(ord('p'))) incorrect (keyboard_check(ord('P')))
if (keyboard_check_released(vk_up)) correct
if (keyboard_pressed_check('vk_backspace')) incorrect if (keyboard_check_pressed(vk_
backspace))
if (keyboard_check('vk_right') || keyboard_check(ord('d'))) incorrect if (keyboard_check(vk_
right) || keyboard_check(ord('D')))

```

### Write code for the following:

```

If J is being pressed: if (keyboard_check(ord('J')))
If Left arrow is released: if (keyboard_check_released(vk_left))
If Shift and U are being pressed: if (keyboard_check(vk_shift) && keyboard_
check(ord('U')))
If 1 and 8 are being pressed: if (keyboard_check(ord('1'))) && (keyboard_
check(ord('8')))
Enter has just been pressed: if (keyboard_check_pressed(vk_enter))

```

## Basic Projects

- A) Make a movable object that can wrap around the screen, so if it goes off of the screen it appears on the opposite side.

2 Points

- B) Create a simple two-player game, one player using WSAD and the other with arrow keys. One player must chase the other player around the room.

4 Points

## Advanced Project

- C) Create a maze that the player should navigate.

4 Points

Note: You can check for the presence of another object at a location using, for example:

```
if place_meeting(x,y+4,obj_wall)
{
    //do something
}
```

## End of Book Game Keypresses and Simple Movement

The game will use a combination of mouse and keyboard input.

Mouse input will be used solely to detect button presses.

The player will control their ship and select weapon type using the keyboard.

Keypresses will also be used for testing purposes, that is, changing a variable's value such as `global.level` or ***health***.

The arrow keys will be used for movement and number keys 1 through 4 for weapon selection.

## CHAPTER 6



# Objects and Events

This chapter describes using objects and reflects what has been learned in previous chapters.

Objects are the lifeblood of GameMaker: Studio. You use objects to do the following:

- Make moving sprites
- Insert code blocks of GML
- Combine with events to make things happen
- Detect collisions with other objects
- Detect keypresses and mouse input

Objects consist of events. You put your code (or Drag & Drop) in these events to create, change, detect, draw, or make things happen.

The main events you will use most often:

### Create Event

This event is executed when the object is created or at start of the room if already present. This event is only executed once. It is useful for defining variables, and for any other sort of setup associated with new instances of the object, for example:

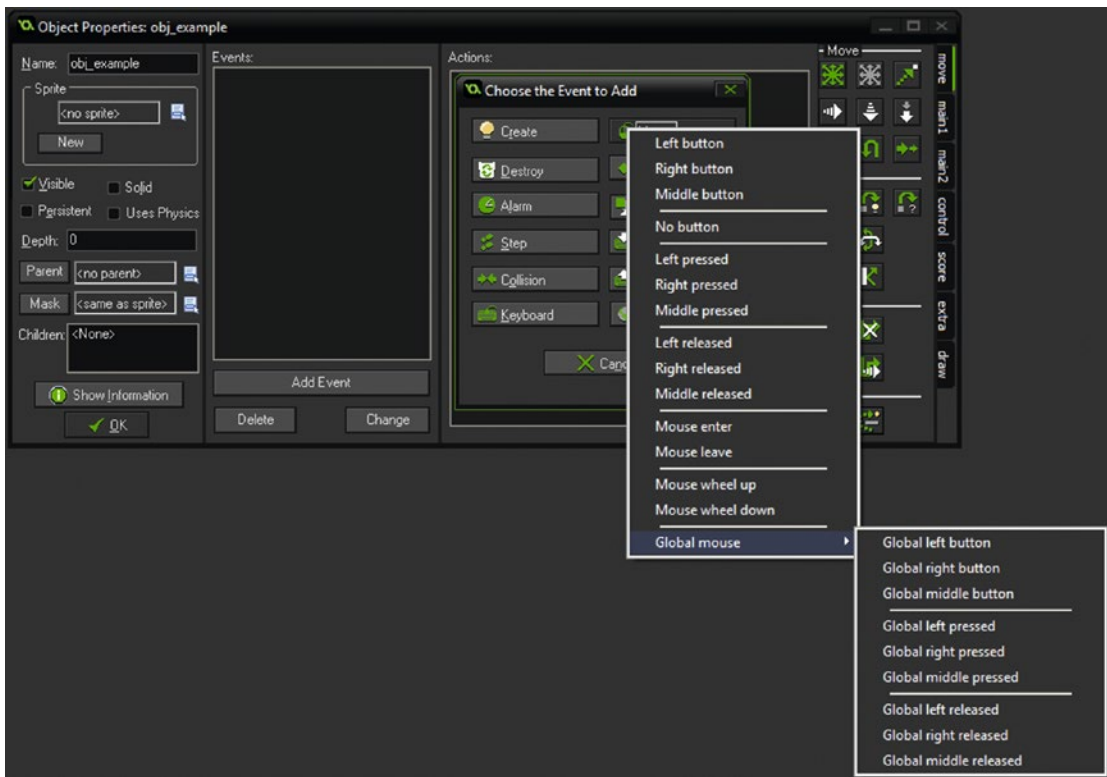
```
my_health=50;  
lives=5;
```

### Mouse Events

These are great for things such as creating an object when the mouse button is clicked, or changing the sub image of a sprite when mouse is over it. This can be used to execute code/actions if the mouse condition is true. This can be done using GML code **Mouse Events**.

Also note: global mouse allows actions to be done if the mouse button is clicked anywhere on the screen, not just over the sprite of the object.

Figure 6-1 shows the **Mouse Event** options available:



**Figure 6-1.** Showing available mouse events

Mouse interaction can also be done in GML in a **Step Event**: for example, the following will play a sound when the left mouse button is released over the objects sprite:

```
if position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left)
{
    audio_play_sound(snd_bounce,1,false);
}
```

The equivalent event for the above code would be **Mouse Left released**, and the code would be:

```
audio_play_sound(snd_bounce,1,false);
```

**Destroy Event**

Code/actions in this event will be executed when the object is destroyed. It's great for changing global variables or playing a sound when it's destroyed. For example, when an enemy object loses all its health and you destroy the object, this can also be achieved in code:

```
instance_destroy();
```

In the **Destroy Event** you could put:

```
score+=10;
```



It is worth noting that Destroy Events don't run upon changing rooms. This has several knock-on effects involving on-death effects and cleanup.

#### **Alarm Event**

Code / actions here will be executed when the chosen **alarm** reaches 0.

Alarms lose 1 for each step of the game. The default room speed is 30 frames per second. So an alarm set for 60 will trigger after 2 seconds. You can set an alarm using GML and then use an **Alarm Event** to execute code when the alarm triggers.

For example, you could use this as controller for a splash\_screen to show a sprite for 5 seconds:

In the **Create Event**:

```
alarm[0]=room_speed*5;
score=0;
lives=5;
global.level=1;
```

And in an **Alarm0 Event**:

```
room_goto(room_menu);
```

Alarm Events must be present for the corresponding alarm[ ] to count down.

#### **Draw Event**

Your code / D&D actions for drawing should be put here, drawing text, shapes, or sprites.

It should be noted that wherever possible, only drawing code should be placed in a **Draw Event**.

If you have any code in a **Draw Event** you will also have to force the object draw the sprite, for example, using it in the simplest form:

```
draw_self();
```

You could add to this, for example, which would draw the score at the top of the screen with the caption Score :

```
draw_text(10,10,"Score "+string( score));
draw_self();
```

#### **Step Event**

Code/actions here are executed every step (frame). At the default room speed this will be 30 frames per second. This is most likely where you'll use the most code. An example would be check the value of **health** and reduce **lives** accordingly:

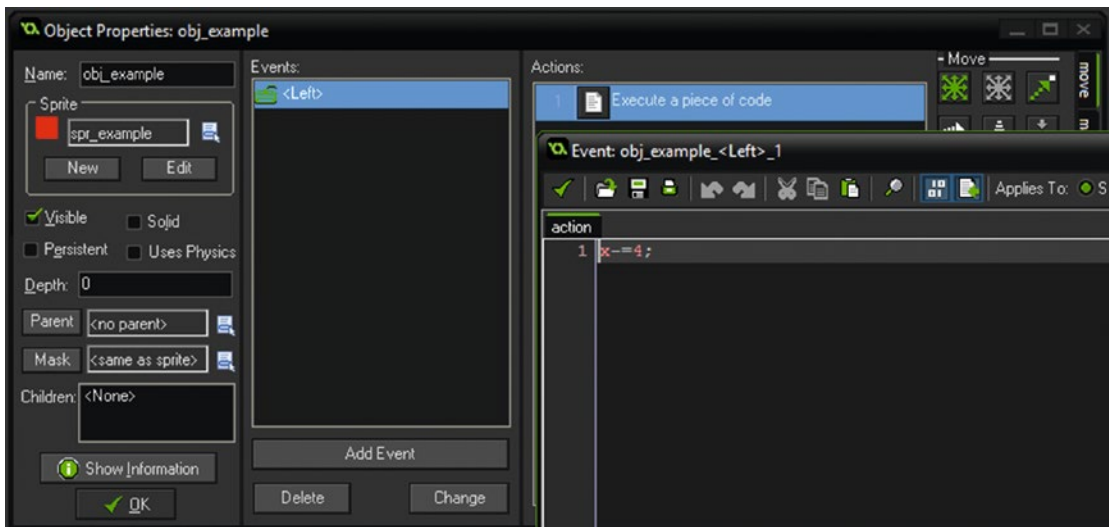
```
if health<0
{
    lives-=1;
    health=100;
    if lives==0 room_goto(room_game_over);
}
```

There may times that you want to execute code before or after a main **Step Event**. For this you can use **Begin Step** or **End Step** accordingly.

### Key Events

Will execute code/actions if a **Key Press Event** executes code or actions if the specified key is being pressed / released. In this book keypress events will be mostly checked using GML code. However you could use **Key Press Events**. An example would be creating moving an object 4 pixels left each time the left arrow is pressed, as shown in Figure 6-2.

■ **Note** The one-time nature of Key Press and Key Release Events: they will not execute each step, instead only when the key is pressed or released.



**Figure 6-2.** Showing an example using a keyboard event

If you want to make code execute while the key is being held down, use the **Keyboard Event**. Similarly for key releases, use the **Key Release Event**.

Detecting keypress and releases using GML code offers much more flexibility, and is the preferred method.

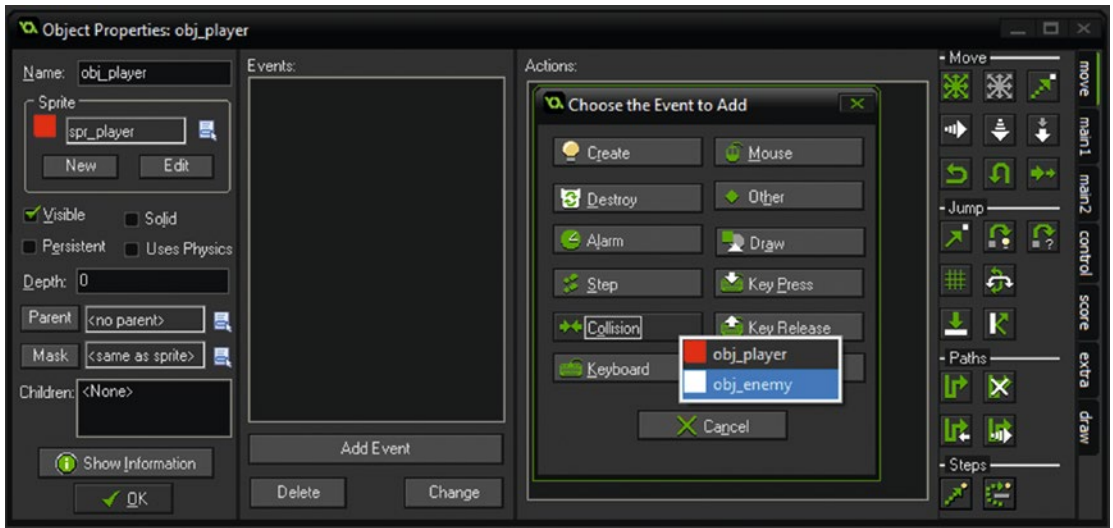
It's important to note that **the Key Press Event** will only execute once, when the key is pressed.

### Collision Event

Code in this section is executed if two instances (or their masks) collide. For this purpose of this book the **Collision Event** will be used rather than GML.

You select which object to test a collision with, and any code or D&D inside that event will be executed if a collision is taking place.

An example would be setting up a collision between **obj\_player** and **obj\_enemy**, as shown in Figure 6-3.



**Figure 6-3.** Setting up a collision event

In this event only, `other` can refer to the colliding instance.  
For example, based on Figure 6-3 above, the code could be:

```
with (other) hp-=1;
```

Then take one point off of the colliding instance's `hp` value for each frame (step).

#### **Draw GUI Event**

This event allows you to draw relative to the screen. It is mainly used for HUD elements, such as displaying the score, lives, bonuses, etc.

This draws independent of any view, so if the view moves, the GUI will not.

In most uses the GUI draws elements that cannot interact with the player.

## Worksheet – Objects

Which event would you place the following:  
(some have more than one method)

- Drawing the string "Score: "
- To do something when two objects touch each other.
- To change the background music when **score** is over 100.
- When your **health** is 0 (or below), lose a life.
- Destroy an object if it leaves the room.
- To check on left mouse button pressed anywhere in the room.
- To check for a keypress of "X."
- Do something when **alarm5** reaches 0.
- To force an object to draw the object's sprite image.
- Change a variables value when **health** is less than 50.
- Some GML you want to run every step.
- When the mouse cursor leaves an object.
- Execute code when an objects is destroyed

## Worksheet – Objects – Answer Sheet

Which event would you place the following:  
(some have more than one method)

- Drawing the string "Score: " In a **Draw Event**.
- To do something when two objects touch each other. In a **Collision Event**, or using GML in a **Step Event**.
- To change the background music when **score** is over 100. In a **Step Event**.
- When your **health** is 0 (or below), lose a life. In a **No More Health Event**, or using GML in a **Step Event**.
- Destroy an object if it leaves the room. In an **Intersect Room Boundary Event**, or using GML in a **Step Event**.
- To check on left mouse button pressed anywhere in the room. In **Global Left Mouse Button Event**, or using GML in a **Step Event**.
- To check for a keypress of "X." Using a **Key Press Event**, or using GML in a **Step Event**.
- Do something when **alarm5** reaches 0. Using **Alarm5 Event**. Assumes that an alarm[5] has been set and started.
- To force an object to draw the object's sprite image. In a **Draw Event** put draw\_self().
- Change a variable's value when **health** is less than 50. In a **Step Event**.
- Some GML you want to run every step. In a **Step Event**.
- When the mouse cursor leaves an object. Using a **Mouse Leave Event** or using GML in a **Step Event**.
- Execute code when an object is destroyed. Using a **Destroy Event** or using GML in a **Step Event**.

## Basic Projects

- A) Draw the health of a player as text in red above a player when health is less than 20. Set it up so P and L change the value of health.  
2 Points
- B) Make some text change colour, at random, each time the space bar is pressed.  
2 Points
- C) Create an object that changes colour when the mouse is over and when clicked on the object. Use a different sub image for each colour.  
2 Points

## Advanced Project

- D) Create a mini game that randomly displays three objects that move in random directions when created and when clicked by the player. If objects go off side of screen, wrap around screen. Player is to click objects to get points and display points onscreen.  
4 Points

## End of Book Game Objects

A number of objects will be used in the end game, and they will be for this:

- A splash screen object to load / define variables
- Menu and shop buttons
- Player's ship
- Bullets
- Asteroids
- Bonus objects
- Control objects for spawning asteroids
- Control object for a messaging system
- Parent objects for asteroids and bullets (makes coding easier)

Each of these objects will serve a different purpose, and as such will be programmed accordingly.

Think about a few of these objects. Try to work out some variables that each of these objects will need.

What will they do; and importantly, when and how will these objects interact with each other and the player?

## CHAPTER 7



# Sprites

Sprites are images or sets of images that are assigned to objects. Sprites are images or multiple images. Multiple images can be used, for example, to create animations, or a change of image when a mouse cursor is over it. An example animation would be a character running. An example of a single image would be a menu button. Sprites are the graphic element of an object, which are displayed in game. There are lots of ways you can change how a sprite is drawn, which can be used to create various effects. Sprites can be used for the following such things:

- Displaying player and enemies
- Missiles and weapons
- Walls and platforms
- Menu buttons
- Upgrade buttons
- Lives
- Moving objects
- Collectibles

You set an origin for a sprite. It is this point that will be used for displaying onscreen at an  $X$  and  $Y$  location. It is also worth noting that this point is also where transformations such as scaling and rotation are based around. It should be chosen carefully according to what the sprite will be used for. Figure 7-1 shows the sprite origin set as center.



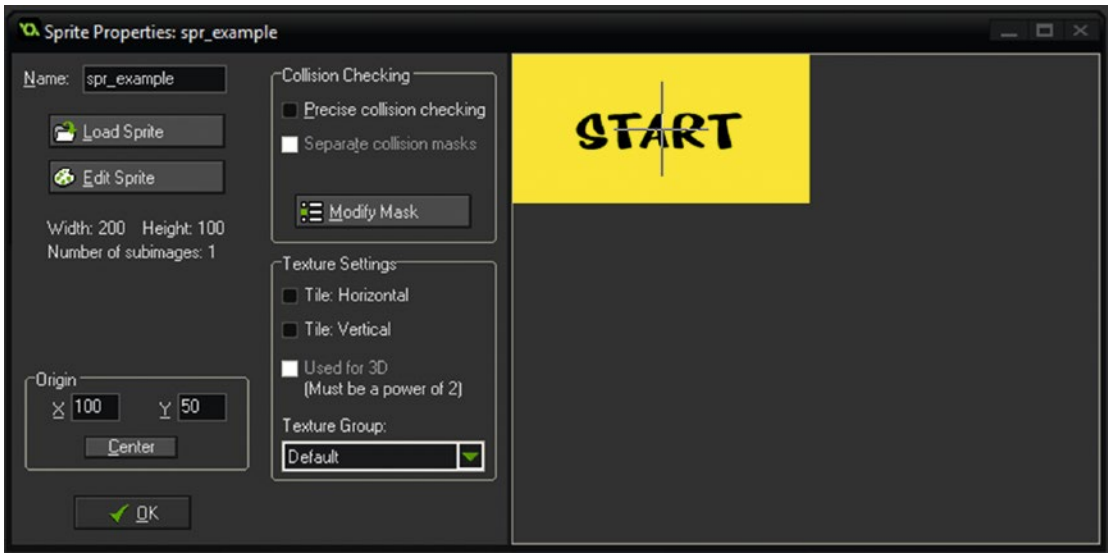


Figure 7-1. Showing sprite origin set at center

Create a new object, **obj\_coin**, and click **New** as shown in Figure 7-2.

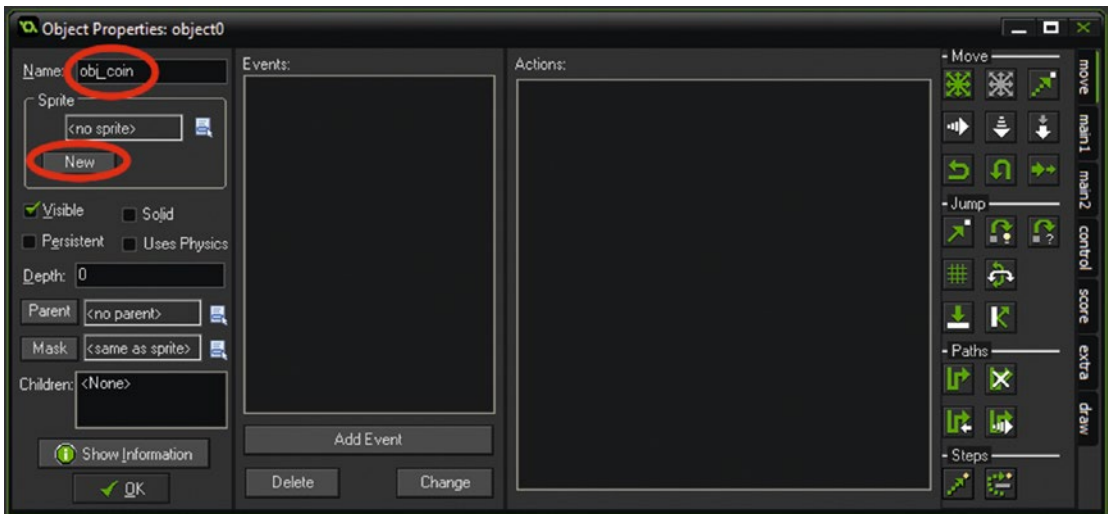
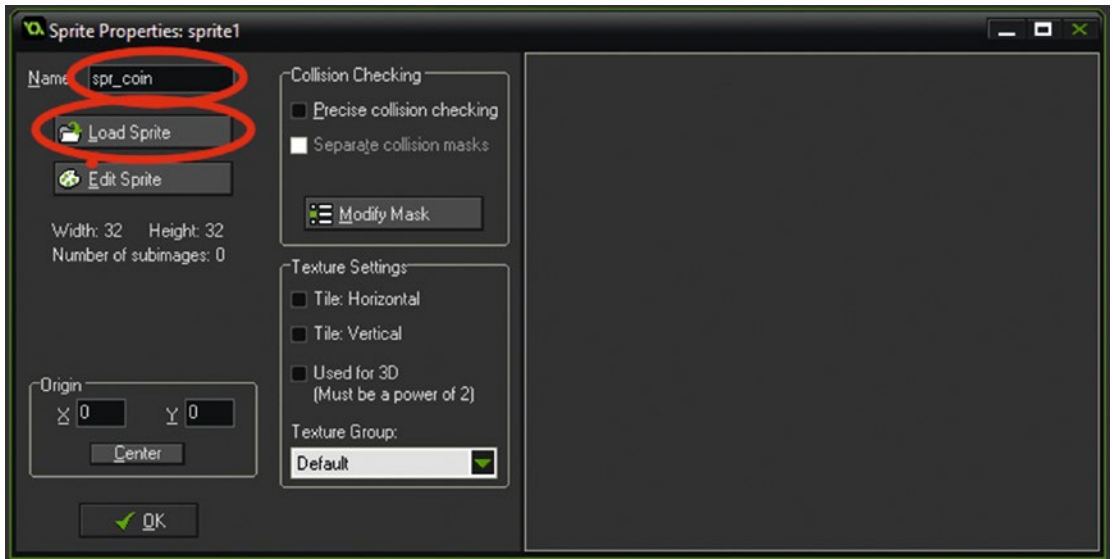


Figure 7-2. Creating a new object and loading sprite

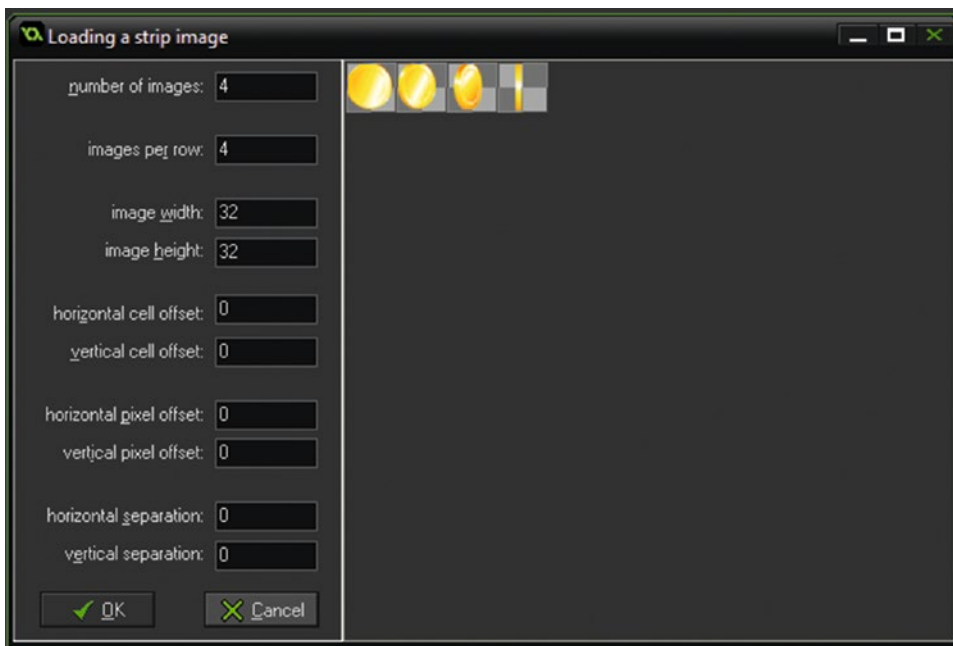
Next, name this sprite **spr\_coin** and click **Load Sprite**. This step is shown in Figure 7-3:



**Figure 7-3.** Naming sprite and loading images

Next load all the coins sprites. These can be found in the resources **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 7**. Click the first image, then scroll down to the last image and click this while holding shift. This will select all images as shown in Figure 7-4. Click **Open** to load them.

This coin is provided as a strip that contains all sub images in a single image. To load this create a sprite, **spr\_coin**, click **Edit ► File** and the Create From Strip. Load in as shown in Figure 7-4:



**Figure 7-4.** Loading in coin strip

Set the origin to center, and click OK twice to close all windows.

Sprites with multiple sub images may also be in a strip format. To load a strip, create a new sprite, click **Edit, File, and Create From Strip**, select the appropriate sprite strip, and apply settings as needed.

Note that all sprites have an origin point; this can be considered the point where the sprite will be “pinned” to the instance in the room. The origin point of the selected sprite will match with the chosen position.

Next create an object, **obj\_player**, and assign the player sprite from the resource folder.

Create a **Step Event**, and put in the following code. This code will detect keypresses of A, W, S, and D and change the X and Y location of the instance in the room. This will make the sprite appear to move.

```
if keyboard_check(ord("W")){y-=3;}
if keyboard_check(ord("A")){x-=3;}
if keyboard_check(ord("S")){y+=3;}
if keyboard_check(ord("D")){x+=3;}
```

Open **obj\_coin** again, and put the following code into a **Collision Event** with **obj\_player**, as shown in Figure 7-5.

```
x=irandom_range(16,room_width-16);
y=irandom_range(16,room_height-16);
```

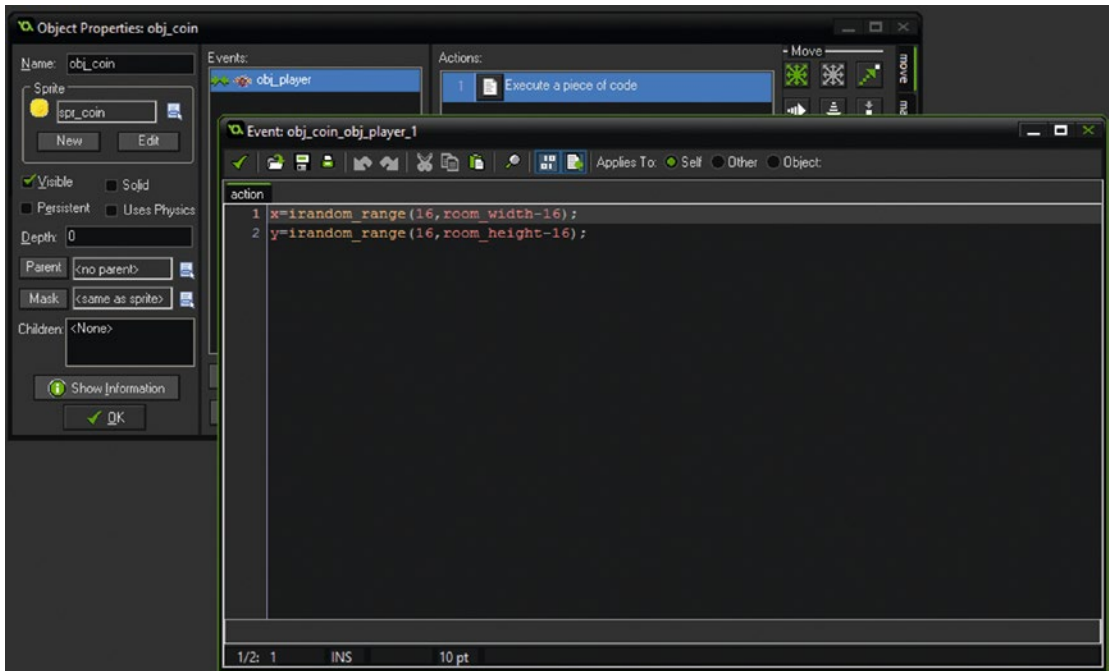


Figure 7-5. Adding code to a collision event

The above code will chose a random value with the given range. 16 is used so that the coin does not appear over the room's border.

Create a room, **room\_game**, set the dimensions as 800 by 400. Place one of each object in the room. Now test this game.

An example GMZ for this available in the resources at: **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 7**

For other useful code, see the manual for usage; the first three are useful for checking a value as well as setting it.

`image_angle` - Can be used to set the direction (rotation of a sprite).

`image_speed` - How quickly a sprite's sub images animate.

`image_index` - Set a specific sub image.

`sprite_get_number(index)` - Returns how many sub images a sprite has.

`draw_sprite_ext()`; - Allows drawing with additional settings.

For example, you may have a different sprite for the character moving left and moving right. These could be sub images 0 and 1. You can set the sub image that is being shown by using `image_index`, for example, when the player is moving left:

```
image_index=0;
```

And when it's moving right:

```
image_index=1;
```

Remember you can set `image_speed` to 0 to set the frame so it doesn't animate away on its own; do this in the object's Format for Event or change the above code to:

```
image_speed=0;
image_index=0;
```

And when it's moving right:

```
image_speed=0;
image_index=1;
```

## Worksheet – Sprites

1. Cross out the wrong answers.
  - Sprites are images that can be used inside a game.
  - The only way to create a sprite is to draw it using GameMaker: Studio.
  - Sprites must be manually drawn.
  - Sprites can be animated using more than one frame/sub image.
  - All sprites have an origin point.
  - All sprites must have precise collision checking enabled.
  - You can assign a sprite to any object in the game.
  - New sprites can be created from inside the game.
2. What is an origin point in a sprite?
3. Correct the mistakes in the following GML.

```
instance_destroy;  
imageIndex=2;  
image_speed=five;
```

## Worksheet – Sprites – Answer Sheet

1. Cross out the wrong answers.

Sprites are images that can be used inside a game. True

The only way to create a sprite is to draw it using GML. False

Sprites must be manually drawn. False

Sprites can be animated using more than one frame/sub image. True

All sprites have an origin point. True

All sprites must have precise collision checking enabled. False

You can assign a sprite to any object in the game. True

New sprites can be created from inside the game. True

2. What is an origin point in a sprite?

The point where a sprite will be drawn

3. Correct the mistakes in the following GML.

```
instance_destroy; instance_destroy();  
imageIndex=2; image_index=2;  
image_speed=five; image_speed=5;
```

## Basic Projects

- A) Draw an animated character sprite that animates when moving right.  
2 Points
- B) Set it so the coin animates through its cycle 4 times, then jumps to a new position and starts the cycle again.  
2 Points
- C) Make simple top-down maze game with a character that points in the direction the player is moving.  
3 Points

## Advanced Project

- D) Draw a sprite that changes perspective (size) depending on the y location.  
3 Points

## End of Book Game Sprites

Sprites will be used for:

- Asteroids, one for each size
- Menu button showing locked or unlocked
- Various game buttons
- Player ship
- Bullets
- Bonus objects
- Radar blips

Sprites required for the game are included in the resources download, but have a go at making your own. There are several graphics programs for creating sprites and images. A selection:

- GIMP
- Draw Plus Starter
- InkScape
- Paint.net
- Pyxel Edit – Although \$9, some consider this the best program for sprite creation

There's also a few programs designed for sprite editing / generation, which are worth a mention:

<http://www.piskelapp.com/> (online)

<http://esotericsoftware.com/> (download)

<http://gaurav.munjhal.us/Universal-LPC-Spritesheet-Character-Generator/> (online)



## CHAPTER 8



# Health, Lives, and Score

Most casual games will have some sort of health, lives, or score system. Players usually start off with full health and lose some of it when they get hit by an enemy bullet, land on spikes, or collide with something they shouldn't. Once a player loses all their health, they usually lose a life and are transported back in the level to a re-spawn site. Different games have different goals, though with a lot of them the aim is to achieve the highest score possible. Fortunately GameMaker: Studio makes it really easy to set up such a system described above.

**health**, **lives**, and **score** are built-in global variables. However, you don't need to put "global." before them.

You would use `global.` when using your own variables that you want multiple objects to be able to use, for example: `global.level`, `global.hp`.

Note: **health**, **lives**, and **score** are basically global variables; as such it should only be used for one instance, usually the player. If you need to monitor **health**, **lives**, or **score** from more than one object, you'll need to create your own instance or global variables, for example, `my_health` or `global.enemy_1_health`.

Most games you create are likely to have a number of **lives** and / or **health**, and a **score** to keep track of. You're not obliged to use the built in variables. Lots of casual games have an aim of trying to get the highest score.

**health**, **lives**, and **score** can be drawn on the screen in text or graphically.

At the start of the game you'll want to set the initial values for these (or load them as saved from a previous play). An example is shown below.

```
score=0;
health=100;
lives=5;
```

---

■ **Note** **health** starts with a default value of 100; however, I prefer to set it so I may change it later, depending on what's required for the game.

---

### Health

You can treat all of these: **health**, **lives**, and **score** the same as you would any variable; you can test, change, and draw these variables.

Some examples:

In the **Collision Event** of enemy bullet with the player.

---

■ **Note** it is very important to destroy the bullet at this point, to prevent health being reduced by 1 each step / frame.

---

For example, in a bullets **Collision Event** with a player object:

```
health-=1;
instance_destroy();
```

Or if you are doing the **Collision Event** within the player object:

```
health-=1;
with (other) instance_destroy();
```

In the **Collision Event** with a health bonus object. As before, destroy the health bonus straightaway to prevent it being added every step:

```
health+=5;
with (other) instance_destroy();
```

---

■ **Note** *health* is not capped at 100. You can make use of this depending on what your game requires. For example, you could test if it's greater than 100 and cap at 100, or create an extra life if it reaches a determined value.

---

In a **Step Event** you may want to constantly check the *health* and *lives* variables:

```
if health<=0
{
    lives-=1;
    health=100;
}
if lives<=-1
{
    room_goto(room_game_over);
}
```

You can draw the value *health* just as you would any other real variable.

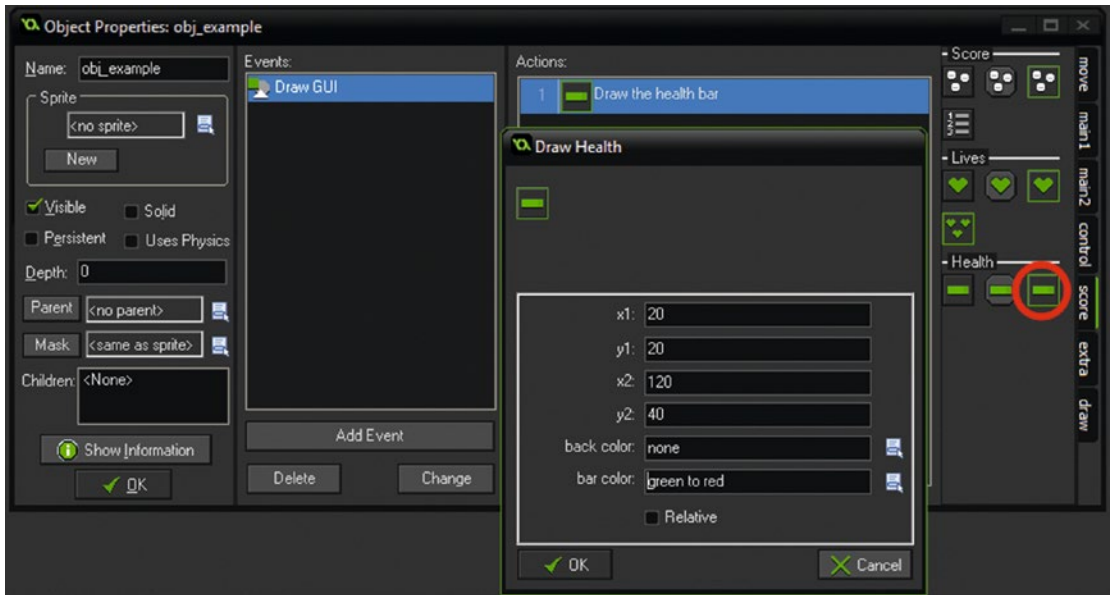
There is also a built-in Drag & Drop action for drawing a health bar, **Draw the health bar**. Figure 8-1 shows an example setup.

x1 is the X location at the top left of the health bar

y1 is the Y location at the top left of the health bar

x2 is the X location at the bottom right of the health bar

y2 is the Y location bottom right of the health bar



**Figure 8-1.** Drawing health bar as an image

■ **Note** The example in Figure 8-1 uses the **Draw GUI Event**, not the **Draw Event**. As mentioned in a previous chapter, this event draws independently of the view and is generally used for HUD elements such as the score, health, lives, etc.

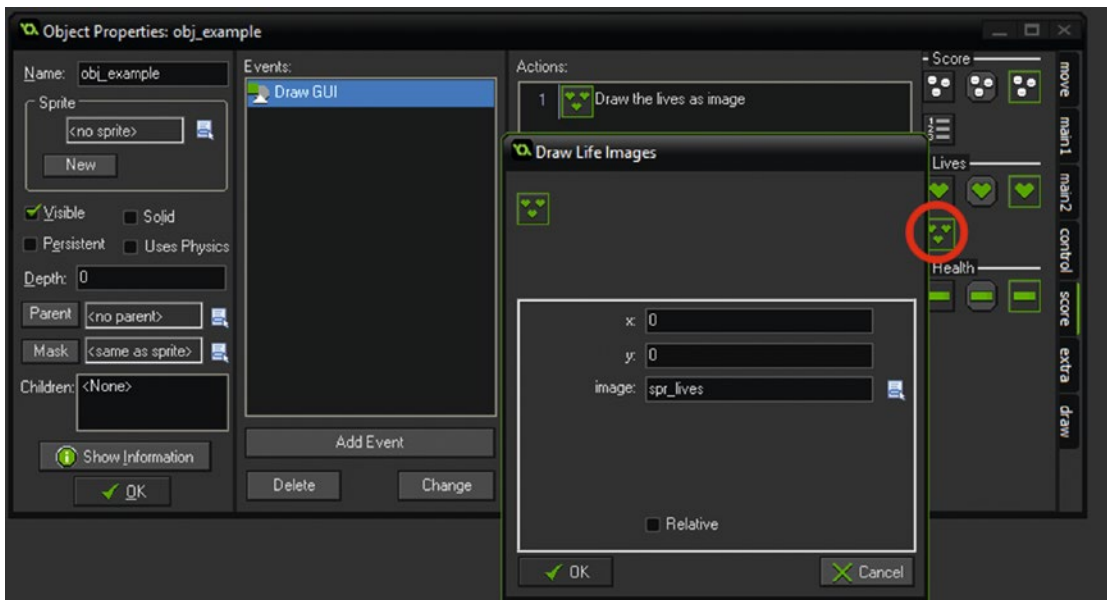
### Lives

As with other variables, you can test, change, and draw the **lives** variable. Draw score:

```
draw_text(50,50,lives);
```

You can also draw **lives** graphically using the in-built Drag & Drop action. There is no single function to do this in GML. This example assumes sprite **spr\_lives** has been loaded in, and this is available at **Project Assets & GMZ Files > Assets Used In Main Chapters > Chapter 8**.

It's OK to use D&D if you need to, as shown in Figure 8-2.

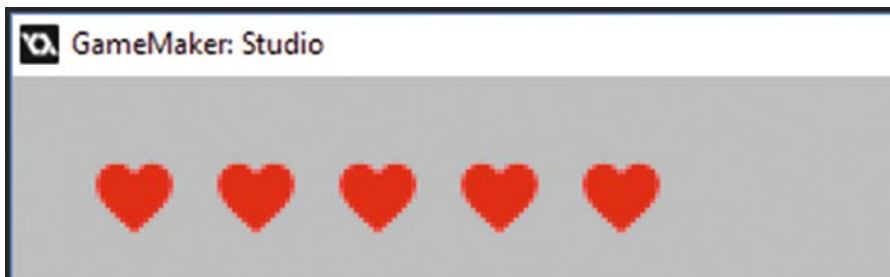


**Figure 8-2.** Drawing lives as separate images

If you wanted to draw lives as images manually you could use the following, where **spr\_lives** is the image you’ve created. This will draw the sprite at 50 pixel spacing. This assumes that the variable lives has already been set, though not essential as it is a built in variable:

```
for (var i = 0; i < lives; i += 1)
{
    draw_sprite(spr_lives,0,50+(50*i),50);
}
```

This will space out the lives at 50 pixel intervals; this will look like what is shown in Figure 8-3.



**Figure 8-3.** Showing lives drawn using code

**Score**

You can draw the **score** using GML, for example:

```
draw_text(30,30, "Score "+string(score));
```

You may also wish to display this graphically, for example, to indicate how many points are needed until you reach the next level. This will depend on the style of game you're creating.

For example, you have a game that levels up after every 1000 points. The following code would draw a bar at the top of the room, visually displaying how many points the player needs for the next level and their current level. This example assumes a room width of 800 and a font **font\_score** set up an Arial size 20.

**Step Event**

```
level=(score div 1000)+1; //calculate level
bar_width=(score mod 1000)*.8; //make bar fit room width of 800
```

**Draw Event or Draw GUI Event**

```
//draw background of bar
draw_set_colour(c_red);
draw_rectangle(1,1,800,40,false);
//Draw Current Level
draw_set_colour(c_green);
draw_rectangle(1,1,bar_width,40,false);
//Draw Over Hud
draw_set_colour(c_blue);
draw_rectangle(1,1,800,40,true);
//Draw Current Level in Text
draw_set_font(font_score);
draw_set_colour(c_white);
draw_set_halign(fa_center);
draw_set_valign(fa_middle);
draw_text(400,25, "Level="+string(level));
```

## Worksheet – Lives, Health, & Score

True or False?

- A) **lives** can be set to a maximum of 10. T / F
- B) When `health == 0`, **lives** will automatically decrement by 1. T / F
- C) Health bar can be drawn in any colour. T / F
- D) **lives** can be drawn as integer or sprites. T / F
- E) **health** can have a negative value. T / F
- F) **score** can be saved and loaded from an INI file. T / F

Correct any mistakes in the following:

- A) `lives=+1;`
- B) `if (global.score>1000)`
- C) `if score>1000 room_goto(room_level_2);`
- D) `draw_healthbar(50, 50, 550, 100, 50, c_blue, c_red, c_yellow, 1, yes, no);`

## Worksheet – Lives, Health, Lives, & Score – Answer Sheet

True or False?

- A) ***lives*** can set to a maximum of 10. **F**
- B) When health == 0, ***lives*** will automatically decrement by 1. **T / F - If you want something to happen when health runs out (reaches 0) you'll need to program it in**
- C) Health bar can be drawn in any colour. **T**
- D) ***lives*** can be drawn as integers or sprites. **T**
- E) ***health*** can have a negative value. **T**
- F) ***score*** can be saved and loaded from an INI file. **T**

Correct any mistakes in the following:

- A) `lives+=1`; Though technically correct, this will reset ***lives*** to a value of 1. To increment lives you would use `lives+=1`;
- B) `if (global.score>1000)` ***score*** is already a globally accessible variable, using the prefix `global` is not required and could cause confusion.
- C) `if (score>1000) room_goto(room_level_2)`; This will work fine, though it's good practice to use `{ code }` after a conditional for example:

```
if score>1000
{
    room_goto(room_level_2);
}
```

- D) `draw_healthbar(50, 50, 550, 100, 50, c_blue, c_red, c_yellow, 1, yes, no)`; Use ***true*** and ***false*** instead of yes and no

## Basic Projects

- A) Draw a health bar across the whole of the top of the game window, draw ***lives*** under this as images. Allows keys Q and W to change health value, and A and S to change lives.  
2 Points
- B) Use GML that draws lives as images, using an animated sprite. Use GML, not the built-in Draw Lives action.  
2 Points

## Advanced Projects

- C) Draw a bar at the top of the screen that draws the current score mod 1000. For each 1000 score increase the level by 1. Also draw score and level as text.  
3 Points
- D) Create 4 level buttons that each become clickable for every 1000 score points. Show they are clickable using different sub images. Also change sub image when mouse over and when clicked.  
3 Points



## End of Book Game Health, Lives, & Score

The main player object will make use of the built-in variables **health** and **lives**. A global variable `global.cash` will be used to assess the players progress, which will also allow the use of this variable in a shop where the player can buy weapons.

A control object will be used to monitor player stats and draw this at the top of the screen.

**health** and **lives** will be drawn graphically at the top of the game window.

Each asteroid and enemy ship will use instance variables for its health.

Think about what code will be needed to draw, monitor, and change these variables. What have you learned in previous sections that you can apply here?

## CHAPTER 9



# Mouse

Mouse interaction is valuable in quickly providing inputs into a game. Object movement and selection is more intuitive and doesn't require memorization that may be required with keyboard interaction. This section serves as an introduction to using the mouse.

Mouse input can be used for:

- Clicking on menu buttons
- Creating a location to move an object to
- Making an object move
- Using the middle button to change weapons
- Making an object point in the direction of the mouse
- Detecting screen presses (in iOS / Android)
- Displaying mouse cursor
- Display stats of a clicked object

About 99% of the time you'll want to constantly check the position of the mouse or mouse button interaction. This is achieved by placing your code at the **Step Event**.

As with keyboard interaction, you can test the mouse just being clicked, being held down, and being released, for example:

```
if mouse_check_button(mb_left) // check for being held down
{
    //do something
}
if mouse_check_button_pressed(mb_left) //activates one time only when button is pressed
{
    //do something
}
if mouse_check_button_released(mb_left) // activates one time only when button is released
{
    //do something
}
```

The same logic applies when using **Mouse Events**.

The position of the mouse can be found using:

`mouse_x` the x position of the mouse in the room.

`mouse_y` the y position of the mouse in the room.

For mouse buttons actions you can use, for example:

```
if (mouse_check_button(button))
```

Where *button* can be any of the following: `mb_left`, `mb_right`, `mb_middle`, `mb_none`, `mb_any`

`mb_middle` means middle button

`mb_none` means no button

`mb_any` means any mouse button

You can also use the **Mouse Events** instead of GML, though using GML will provide you with more flexibility.

Using GML you can make things happen when a mouse button is pressed. So the following, when placed in the **Step Event**, would make the object move slowly to the mouse's position when the left button is pressed:

```
if (mouse_check_button(mb_left))
{
    movement_speed=25; //Higher Number Makes Slower Speed
    target_x=mouse_x; //or other target position
    target_y=mouse_y; //or other target position
    x +=(target_x-x)/ movement_speed; //target position-current
    position
    y +=(target_y-y)/ movement_speed; //target position-current
    position
}
```

An example GMZ for the above is available in the resources at: **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 9**

You can also detect movement of the mouse wheel (if present):

```
if (mouse_wheel_up())
{
    weapon+=1;
}
```

You can detect scroll down in a similar manner:

```
if (mouse_wheel_down())
{
    weapon-=1;
}
```

You can also set the cursor to a sprite of your choice:

```
cursor_sprite=spr_name;
```

when **spr\_name** is a sprite that exists.

You can hide the default windows cursor using:

```
window_set_cursor(cr_none);
```

There are a number of built-in cursor types, see `window_set_cursor` in the manual for more info. This can be used (as with `cursor_sprite` above) with great effect in a game. Changing the cursor depending what the player is doing or what objects are at the cursor can give your player extra info, that is, making it clear an instance can be interacted with.

The following code will detect if the mouse is over an object and add to the score, which would increment by 1 every step if placed in a **Step Event**:

```
if (position_meeting(mouse_x, mouse_y, object_name))
{
    score+=1;
}
```

The following code would check that the mouse is over the object's sprite and the left mouse button is released. The code will then play a sound **snd\_bounce**, with a priority of 1 with looping set as **false**.

```
if position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left)
{
    audio_play_sound(snd_bounce,1,false);
}
```

There are also a number of **Mouse Events** that can be used instead of code, which are perfectly viable options; these are shown back in Figure 6-1. As with GML code, you can check for a button being held down, just pressed, or released (and you should understand the difference by now):

## Worksheet – Mouse Movement

Explain what each of the following code blocks do, when placed in the **Step Event**:

```
1)
if (mouse_check_button_pressed(mb_left))
{
    audio_play_sound(snd_beep, 10, false);
}
2)
if position_meeting(mouse_x, mouse_y, obj_button)
{
    show="mouse over";
}
else
{
    show="not over";
}
3) x = mouse_x
   y = mouse_y
   repeat (10)
   {
       instance_create(x, y, obj_smoke);
   }
repeat (3)
{
    instance_create(x, y, obj_fire);
}
```

## Worksheet – Mouse Movement – Answer Sheet

Explain what each of the following code blocks do, when placed in the Step Event:

- 1) This will check for the left mouse button; when pressed it will play the sound **snd\_beep**
- 2) This will set the variable `show` to `mouse` over when mouse cursor is over object, or `not` over otherwise
- 3) This would create 10 instances of **obj\_smoke** and 3 instances of **obj\_fire** at the mouse location

## Basic Projects

- A) Create an object that follows only the mouse's x position.  
2 Points
- B) Make the mouse cursor change when it's over an object.  
2 Points
- C) Draw the mouse's x and y positions in the bottom left of the screen. Remember to use the font and drawing colour and formatting.  
2 Points

## Advanced Projects

- D) Create a sound board (lots of buttons each of which plays a sound when clicked with the mouse). Draw text over each button, explaining what sound it plays.  
2 Points
- E) Create an object that can be moved around the room with the mouse.  
2 Points

## End of Book Game Mouse Movement

Although there's minimal mouse interaction with the game made in this book, other games that you make may be mouse intensive. Mouse interaction in this game is only used for button presses.

However, games made for the Android platform will mostly likely be wholly mouse based (a screen press on a tablet device is the equivalent of a mouse click). GameMaker: Studio allows you to detect taps, double taps, and screen swipes. It's also possible to create virtual buttons for different parts of the screen, which can, with a bit of coding, allow the taps to be recorded as **Keyboard Pressed Events**. Virtual keys map to regular **Keyboard** and **Keyboard Released Events** as well. This is an awesome method, especially if you're porting a game that was originally designed to run on a desktop platform.



## CHAPTER 10



# Alarms

Alarms are useful for many timed activities or abilities that are temporarily available. Timers can enhance gameplay by creating urgency in completing an action before an ability disappears. This section covers the basics of alarms. Alarms can be used to make something happen (or stop happening) when an alarm triggers. Alarms are set to a starting value when the alarm is created.

Alarms can be used for the following:

- Displaying a splash screen for a while
- Making a bomb explode an amount of time after firing
- Changing the player to invincible and then back again
- Display a bonus object for a set amount of time
- Create or move objects after an amount of time
- Create basic AI systems
- Limiting how quickly a player can shoot

An alarm can be set, for example, using, using the default `room_speed` of 30:

```
alarm[0]=60; //set alarm at 2 seconds
```

As you may wish to change the default room speed, a better code would be to use the following approach in all your alarms:

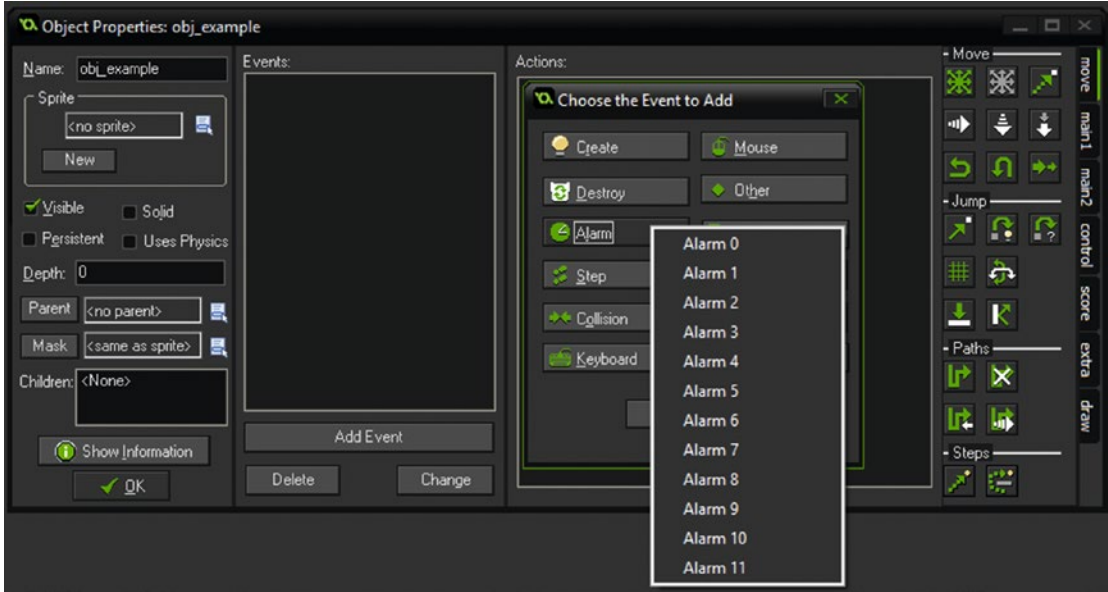
```
alarm[0]=2*room_speed; //set alarm at 2 seconds
```

By using the above method you can change the `room_speed` as needed without messing up your timings.

In total you can have 12 alarms for each object:

```
alarm[0]  
alarm[1]  
....  
alarm[10]  
alarm[11]
```

When an alarm activates (alarm goes off / runs out of time) you can detect this using an **Alarm Event**, which can be selected as shown in Figure 10-1:



**Figure 10-1.** Showing alarm events

You can add to an existing alarm, for example, adding 1 second:

```
alarm[0]+=1*room_speed;
```

Alarms run until they reach 0, and then count one more to -1, so you can stop an alarm by setting it to -1. Here are some examples:

Moving to another room, **room\_menu** after 5 seconds. You could use this in a room set up as a splash screen. For example, this could be used to display your company's logo or a graphic of some description.

**Create Event:**

```
alarm[0]=5*room_speed
```

**Alarm[0] Event:**

```
room_goto(room_menu);
```

Delaying how quickly a player can shoot, the code below would limit the rate of fire to once every 2 seconds:

**Create Event:**

```
can_shoot=true;
```

When player shoots, by pressing the Z key:

```
if can_shoot && keyboard_check(ord('Z'))
{
    //bullet creation code here
    can_fire=false;
    alarm[1]=2*room_speed;
}
```

**Alarm [1] Event:**

```
can_shoot=true;
```

As you can see, Alarm Events are a very useful commodity allowing us to set how often something can happen or how long something lasts.

---

■ **Note** Be careful about holding alarms open by constantly setting them. Use a flag to prevent this, as was done with `can_shoot` in the example above.

---

## Worksheet – Alarms

1. What are alarms?
2. How do you start an alarm?
3. What event would you use to execute code when alarm[4] goes off?
4. How many steps for an alarm set to 7 seconds with a room speed of 30?
5. How many alarms can a single object have?
6. How would you decrease an alarm by 45 steps?
7. How would you make a player invincible for 10 seconds?
8. How would you spawn an enemy every 15 seconds, then every 14 seconds, then every 13 seconds...? You should set a minimum for this so that the time doesn't go to 0.

## Worksheet – Alarms – Answer Sheet

1. What are alarms?

Alarms are timers that can be programmed to allow something to happen.

2. How do you start an alarm?

In GML, `alarm[number]=time;`

3. What event would you use to execute code **alarm[4]** goes off?

**The Alarm4 Event.**

4. How many steps for an **alarm** set to 7 seconds with a room speed of 30?

210 steps

5. How many alarms can a single object have? Up to 12. They go from 0 through to 11.

6. How would you decrease an alarm by 45 steps?

`alarm[0]-=45;`

7. How would you make a player invincible for 10 seconds?

**Set invincible to true:** `invincibility=true`, **create an alarm:**

`alarm[0]=10*room_speed`, **reset invincible back to false when alarm triggers:**  
`invincibility=false;`

8. How would you spawn an enemy every 15 seconds, then every 14 seconds, then every 13 seconds...?

Create Event:

```
time=14;
alarm[0]=time*room_speed;
```

Alarm 0 Event:

```
instance_create(50,50,obj_enemy);
if time>1 alarm[0]=time*room_speed else alarm[0]=room_speed;
```

## Basic Projects

- A) Create a program that changes the drawn text every 5 seconds, and use a list of 10 strings.  
2 Points
- B) Create a program with an object that moves with a speed of 1 and increases the speed of the object every 5 seconds. Also make the object wrap around the screen.  
2 Points
- C) Create a program that plays a random sound every 4 seconds.  
2 Points

## Advanced Projects

- D) Create a simple system that allows the player to shoot bullets at a maximum rate of 1 bullet every 2 seconds.  
2 Points
- E) Create an enemy AI that changes direction randomly every 5 seconds. Make the enemy object's sprite point in the direction that it is traveling. Also make the enemy object wrap around the screen. Set the object to shoot a bullet in the direction it is traveling every 8 seconds.

Note: You can assign an instance's value to a variable when you create it, and then set other variables for that instance. For example:

```
ball=instance_create(x,y,obj_ball);  
ball.speed=2;  
ball.health=50;
```

## End of Book Game – Ten Alarms

The game will use a few alarms. They'll be used for:

- Limiting how quickly a player can shoot bullets
- Showing the splash screen
- Showing message on screen
- Start fading bullets after a set amount of time

Alarms are great for changing a value from true to false. By combining two alarms you can change from true to false and then back again. This type of system will be used to display important messages on the screen. There will be an object for monitoring these messages. When the object receives a message, it will display it onscreen. If the object receives more than one message at a time, it will cue the messages and display them with a gap in between. This is achieved by using two different alarms in the one object.

# CHAPTER 11



# Collisions

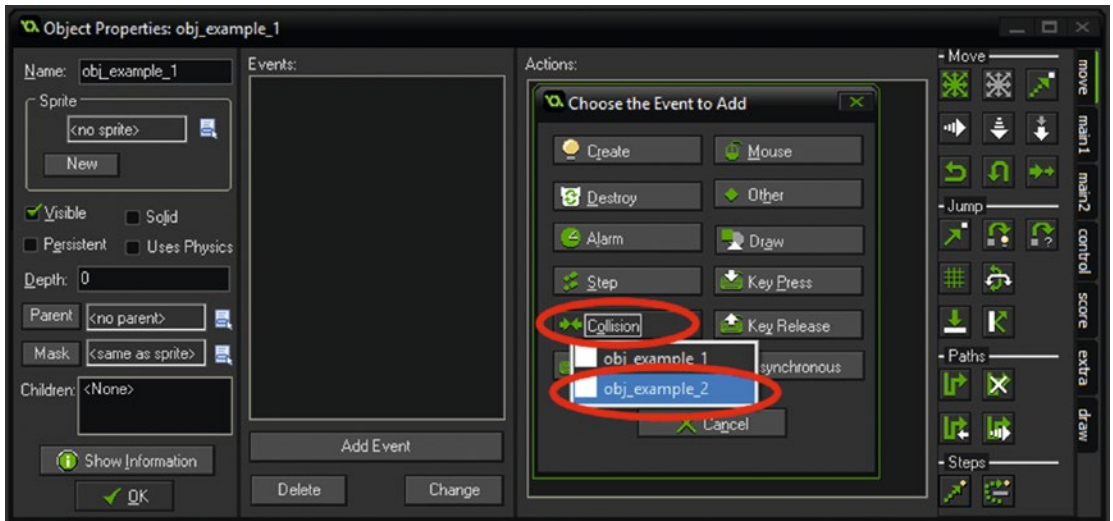
When planning on motions or deciding actions, it is often critical to see if collisions occurred with other instances within the game world. Put simply, collisions are what happens when two instances (with sprites or masks assigned) collide (their sprite or mask overlaps) with each other.

You use this to check if an instance is in collision with another, for example, a character walking on a platform in a platformer game or an enemy being hit by a player's bullet.

For example, here are things that can be made to happen when a collision occurs:

- destroy an object
- create an effect
- play a sound
- change **score**, **health**, or **lives**
- make something start or stop moving
- create a new object

Most of the time it's sufficient to use the **Collision Event**. You can create a **Collision Event** as shown in Figure 11-1; just select the object you want to check a collision with.



**Figure 11-1.** Setting up a collision event



There are a number of different ways to use GML to check for a collision.

For example, check whether it does not collide with another object, returning true or false:

```
place_empty(x, y);
```

As above, but checks for solid objects only:

```
place_free(x, y);
```

You can check for a specific object, which uses the sprite as base to check for overlap:

```
place_meeting(x, y, object);
```

You can check a single pixel location to see if an object is in that position, for example to check for a specific instance:

```
position_meeting(mouse_x,mouse_y,id);
```

You can destroy all objects at a location:

```
position_destroy(x, y);
```

Or the following, which finds the instance ID:

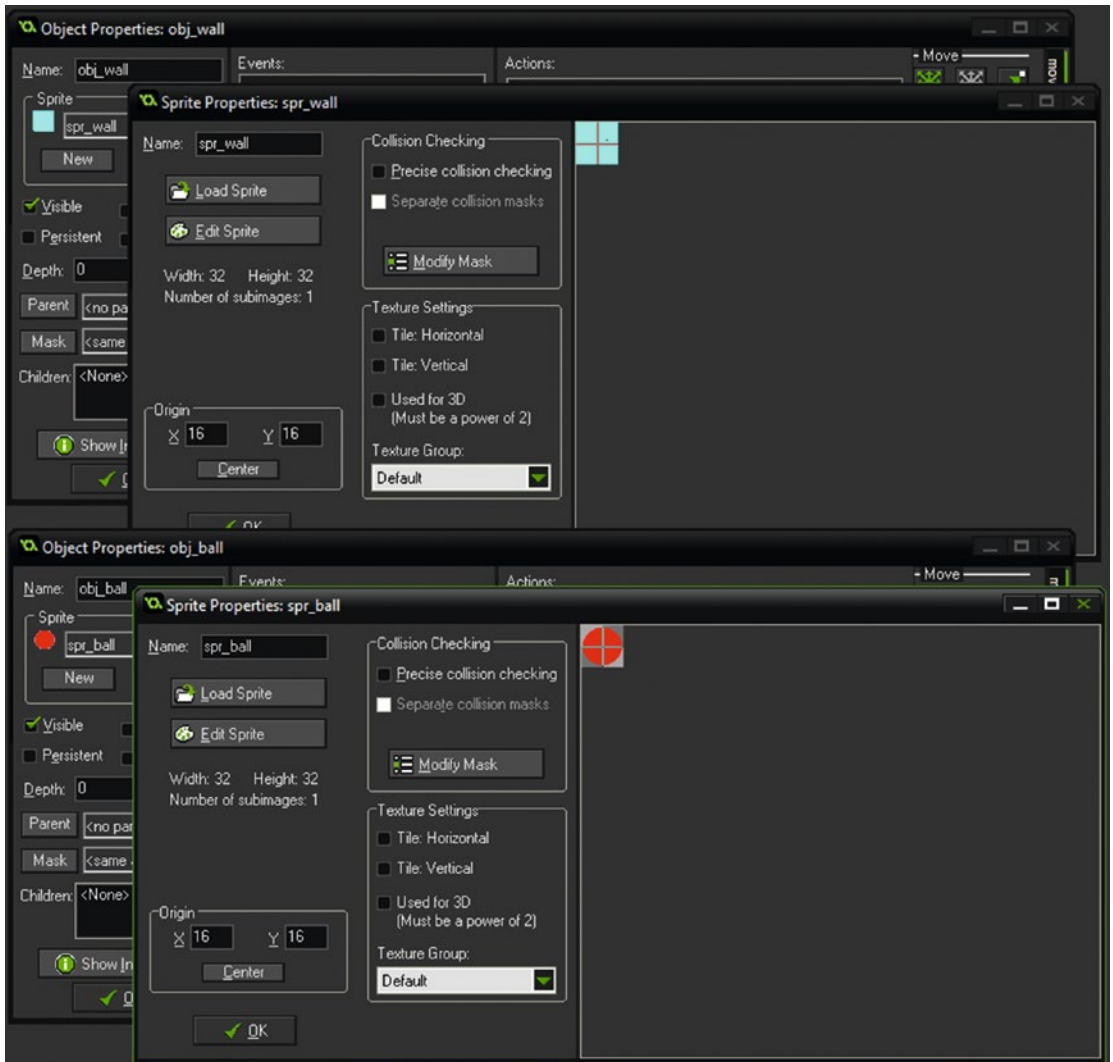
```
instance_position(x, y, obj );
```

For the purpose of this level 1 book we'll mainly be using the in-built **Collision Event** for all object collisions (except for buttons that detect a mouse click), but feel free to experiment with GML. Project files and resources are available in the download resources folder.

**Coding examples: Example 1: Bouncing ball**

Create a solid wall **obj\_wall**, as a 32x32 pixels sprite-coloured solid blue, and a player object **obj\_ball** as a smiley face also 32x32:

Set the origin at the center for each sprite shown in Figure 11-2:



**Figure 11-2.** Creating two objects, assigning sprite, and setting as solid – step 1

In **Create Event** of **obj\_ball** object put:

```
direction=(irandom(360));
speed=5;
image_speed=0;
```

This will make the object start moving in a random direction at a speed of 5.

Open up object **obj\_ball**, make a **Collision Event** with **obj\_wall** and put:

```
move_bounce_all(true);
```

This step is shown in Figure 11-3:

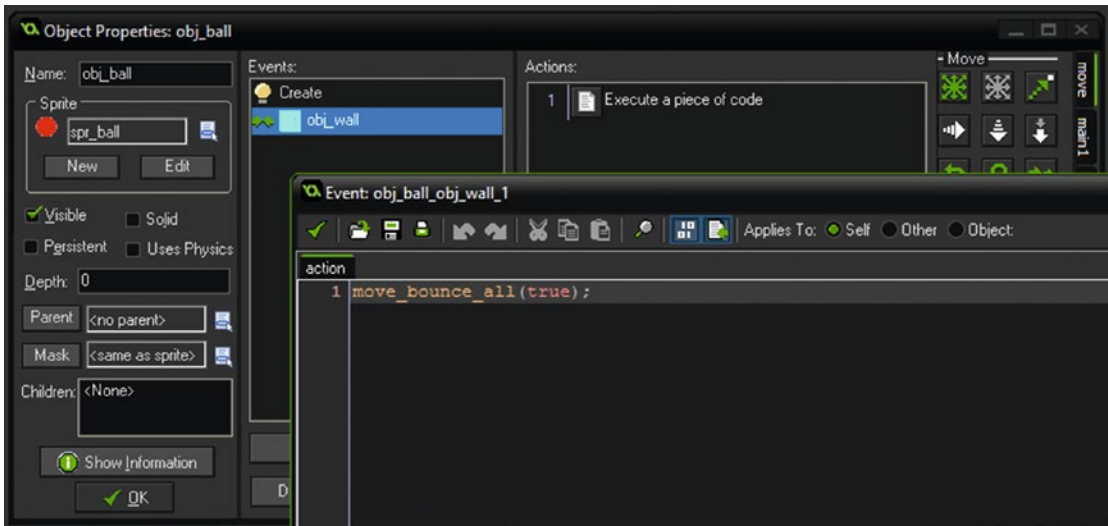


Figure 11-3. Adding GML to a collision event – step 2

Create a room and line the border with **obj\_wall** and place one of **obj\_ball** in the middle, as demonstrated in Figure 11-4.

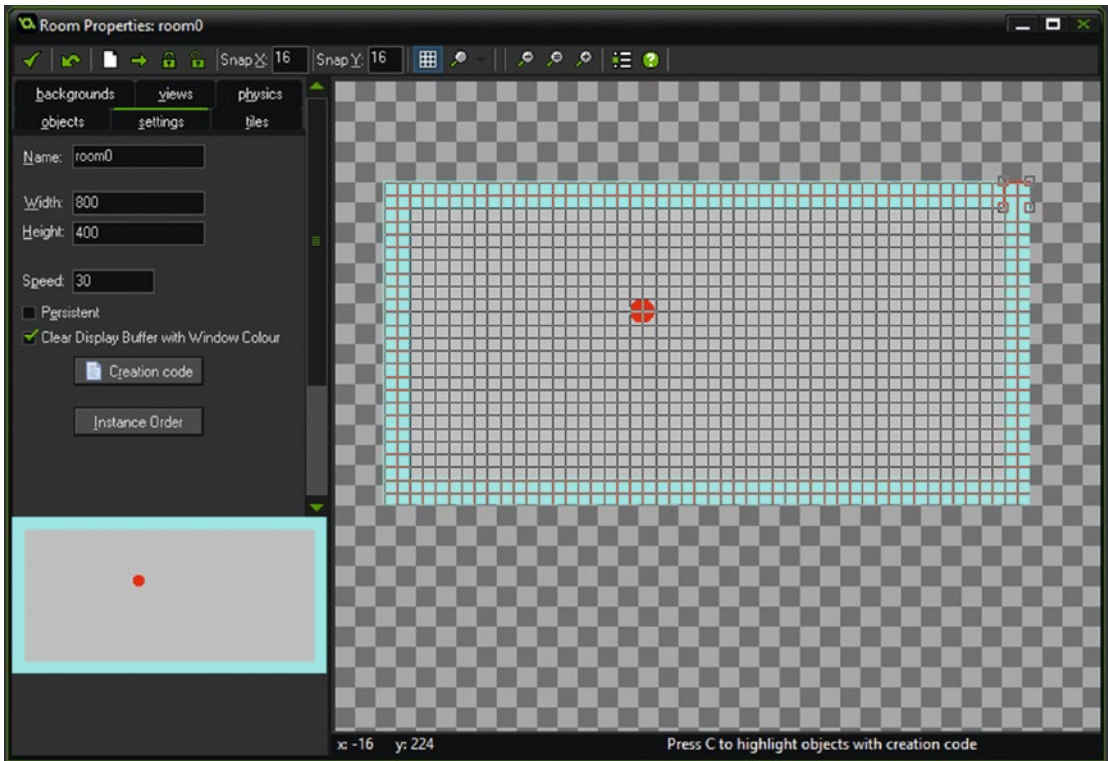


Figure 11-4. Adding objects to a room – step 3

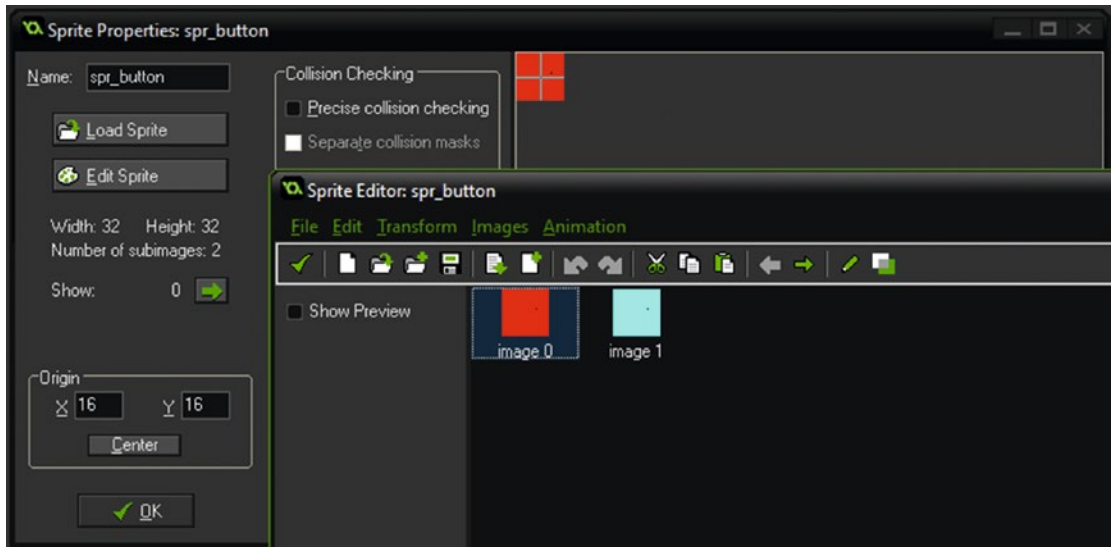
If you now play this game, the ball will bounce around the room.

An example GMZ for the above is available at: **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 11**

### Example 2 Mouse Over

There will be lots of occasions that you may want to check that the mouse cursor is over an object before performing any additional code. This is a simple example.

Create a new sprite, **spr\_button**. Add two sub images, one blue and one red, with a size of 32x32, as shown in Figure 11-5:



*Figure 11-5. Showing sprite with two sub images set*

Create an object **obj\_button** and assign this sprite you've just made. For the **Create Event** use this code:

```
image_speed=0;
image_index=0;
```

For the **Step Event** use:

```
if (position_meeting(mouse_x, mouse_y, id))
{
    image_index=0;
}
else
{
    image_index=1;
}
```

The above code will set a different image index depending whether the mouse cursor is over its sprite or not.

A GMZ example for the above is available in the resources at: **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 11.**

**Example 3 Collision Line**

For this example you'll need three sprites, **spr\_yellow\_face** and **spr\_red\_face** and **spr\_wall**. A size of 32x32 with origin set as center will be fine.

Create a wall object **obj\_wall** with a solid blue sprite, set as solid.

Create an object, **obj\_player**, set the sprite **spr\_yellow\_face**.

In the **Step Event** of **obj\_player** put the following; this will allow you to move the player object using the arrow keys:

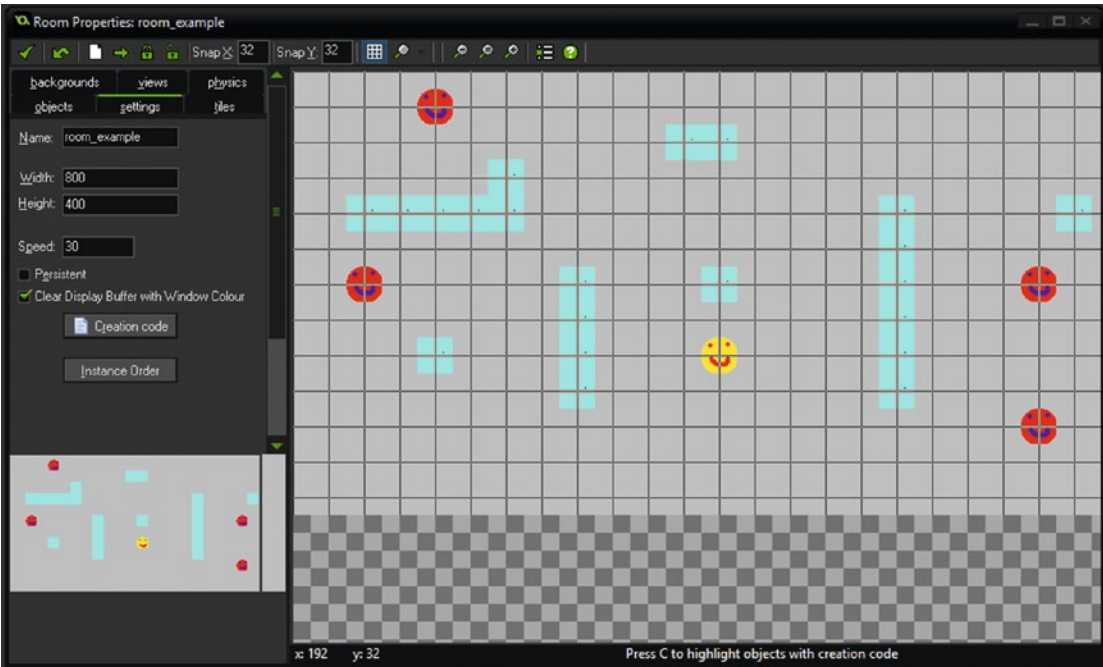
```
x+=4*(keyboard_check(vk_right)-keyboard_check(vk_left));
y+=4*(keyboard_check(vk_down)-keyboard_check(vk_up));
```

Create an object, **obj\_target**. Set the sprite **spr\_red\_face**.

In the **Draw Event** of **obj\_target** put the following code, which will draw a line between **obj\_target** and **obj\_player** if there is a direct line of sight (i.e., no walls in the way):

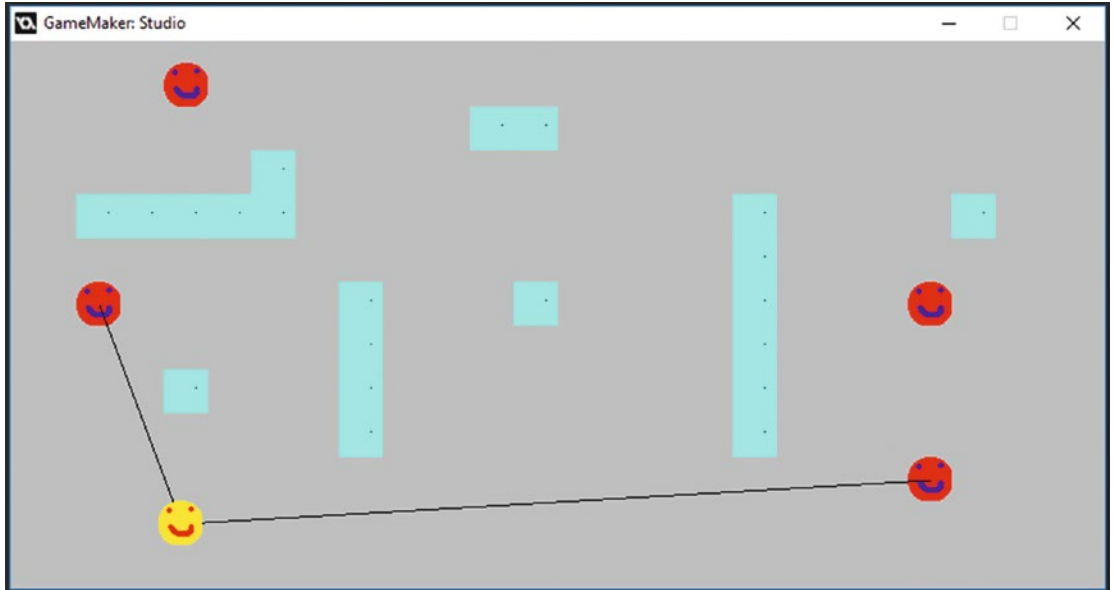
```
draw_self();
if (collision_line(x,y,obj_player.x,obj_player.y,obj_wall, false,true)) == noone
{
    draw_line(x,y,obj_player.x,obj_player.y);
}
```

Place one instance of **obj\_player**, and a few each of **obj\_target** and **obj\_wall** into a new room. An example layout is shown in Figure 11-6.



**Figure 11-6.** Showing example room with objects added

When the **obj\_target** can 'see' **obj\_player** it will draw a line between the two, as shown in Figure 11-7:



*Figure 11-7. Showing lines being draw to player if enemy can see it*

A GMZ for the above is available at: **Project Assets & GMZ Files** ► **Assets Used In Main Chapters** ► **Chapter 11**.

You could use something like this as the basis for an AI system.

## Worksheet – Collision Events

Correct the following code, if there is an error:

```
if place_meeting(obj_car, 65, 80)
if (position_meeting(mouse_x, mouse_y, obj_tree))
if (!place_free(x+5, y))
if (position_meeting(x, y, spr_bonus))
if (collision_line(x+100, y, obj_player.x, obj_player.y, obj_wall, true, true))
```

Explain what the following code does:

```
if (position_meeting(x, y, obj_bomb)){audio_play_sound(snd_bomb, 10 ,false); health-=1;}
if (place_free(x+5, y)) {x+=5;}
if (place_empty(x, y)){instance_create(x, y, obj_enemy);}
```

How would you do the following using events / GML:

- A) Set player variable `can_see` to true if it can be seen by an instance of **obj\_enemy\_1** and **obj\_enemy\_2**? For this question, assume that there is only one of each instance.
- B) Move an object to the right unless it collides with **obj\_wall**?

## Worksheet – Collision Events – Answer Sheet

Correct the following code, if there is an error:

```
if place_meeting(obj_car, 65, 80) if place_meeting(65,80,obj_car)
if (position_meeting(mouse_x, mouse_y, obj_tree)) correct
if (!place_free(x+5, y)) correct
if (position_meeting(x, y, spr_bonus)) Should be obj_bonus
if (collision_line(x+100, y, obj_player.x, obj_player.y, obj_wall, true, true))Should be
(collision_line(x+100, y, obj_player.x, obj_player.y, obj_wall, true, true)!=noone
```

Explain what the following code does:

- A) Will play sound **snd\_bomb** and reduce **health** by 1 if **obj\_bomb** is at a position.
- B) Will move right 5 pixels if no solid object present 5 pixels to the right.
- C) Checks if any instance exists at position; if not create **obj\_enemy** at position.

How would you do the following using events / GML:

- A) Set player variable can\_see to true if it can be seen by an instance of **obj\_enemy\_1** and **obj\_enemy\_2**?
- B) if (collision\_line(x,y,obj\_target.x,obj\_target.y,obj\_wall,false,true)) == noone or (collision\_line(x,y,obj\_target\_2.x,obj\_target\_2.y,obj\_wall,false,true)) == noone

```
{
    can_see=true;
}
else
{
    can_see=false;
}
```

- C) Move an object to the right unless it collides with **obj\_wall**?

```
if !place_meeting(x+(sprite_width/2), y, obj_wall) x+=4;
```



## Basic Projects

- A) Make the player change colour when it can see one or more of `obj_enemy`.  
2 Points
- B) Create a new object, `obj_target`, and assign a pink sprite to it. If player collides with it, play a sound and make it destroy itself.  
2 Points
- C) Create a clickable object with four sub images. When the mouse button is released when over the object change the sub image. On the forth click destroy the object.  
2 Points

## Advanced Projects

- D) Make the player change direction at random if the mouse gets within 100 pixels in any direction, but only check this once every 5 seconds. Also make the object wrap around the screen. See `distance_to_point(x, y)`; in the manual.  
2 Points
- E) Surround the outside of a room with walls. Create a ball that bounces around the room. Have some objects in the room that require four hits of the ball to be destroyed, changing the sub image each time it's hit. Note: You can use the D&D Bounce in the jump section of the Move tab.

## End of Book Game Collisions

- Player with asteroid
- Player with enemy bullet
- Player with enemy ship
- Asteroid with bullet
- Enemy with bullet
- Player with bonus item

Try and work out what you want to happen when these objects collide. Choose from: Collisions

- Change a variable
- Play a sound effect
- Play a voice
- Create a graphical effect
- Create / destroy an object
- Make something move or stop moving

Player with asteroid - When this happens we'll want to reduce the player's **health** and also create a visual effect and a sound effect or voice.

Asteroid with bullet - When this happens we'll want to reduce the asteroid's **health**, play a sound effect and create a visual effect, then destroy the bullet.

Player with bonus item - Add the value of the bonus to the player, and destroy the bonus object.

Player with enemy ship or bullet - Reduce player's **health** and create a sound effect and graphical effect. If collision with an enemy's bullet occurs, destroy the enemy bullet upon collision with the player.

## CHAPTER 12



# Rooms

Rooms are where you place your instances and where the action happens. A simple room may have an enemy instance and player instance – and maybe a few platforms to jump on – where the player and enemy try to shoot each other.

Instances are placed in the room, which then interact with other instances, keypresses, and mouse events.

The room editor is a very powerful visual layout tool, and has many useful features like views, creation codes, and tile settings.

It allows you to place objects in the room where you want them. You can also do the following:

- Set room creation code (GML that runs on room start, which happens after the **Create Event** of instances already in the room)
- Set backgrounds
- Set views
- Turn physics on or off
- Set the room size and name
- Set and add tiles

Most games have one than one room. You can use different rooms, for example:

- Splash Screen – Defining variables and showing your company’s logo
- Menu – Where a player selects a level to play
- Shop – Where weapons and upgrades can be purchased
- Game Levels – Where the main game action takes place
- Boss Levels – Special level between levels – usually harder than standard levels
- Game Over – When a player completes the game or loses all lives

Most rooms will have some form of background, from the basic static background to moving parallax backgrounds.

Create a new GameMaker: Studio project, and load in a new background. This can be done by clicking the Create Background icon, as shown in Figure 12-1 below:

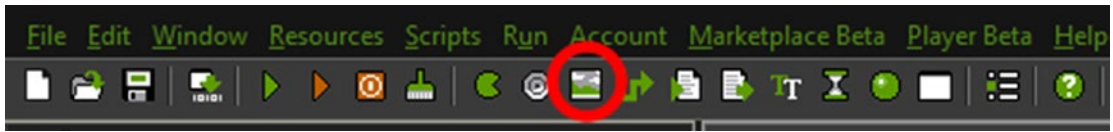


Figure 12-1. Creating a new background

Click the **Create a background** button on the top menu bar, and load in a background from the resources; this step is shown in Figure 12-2. The background to load is supplied in the resources for Chapter 12.

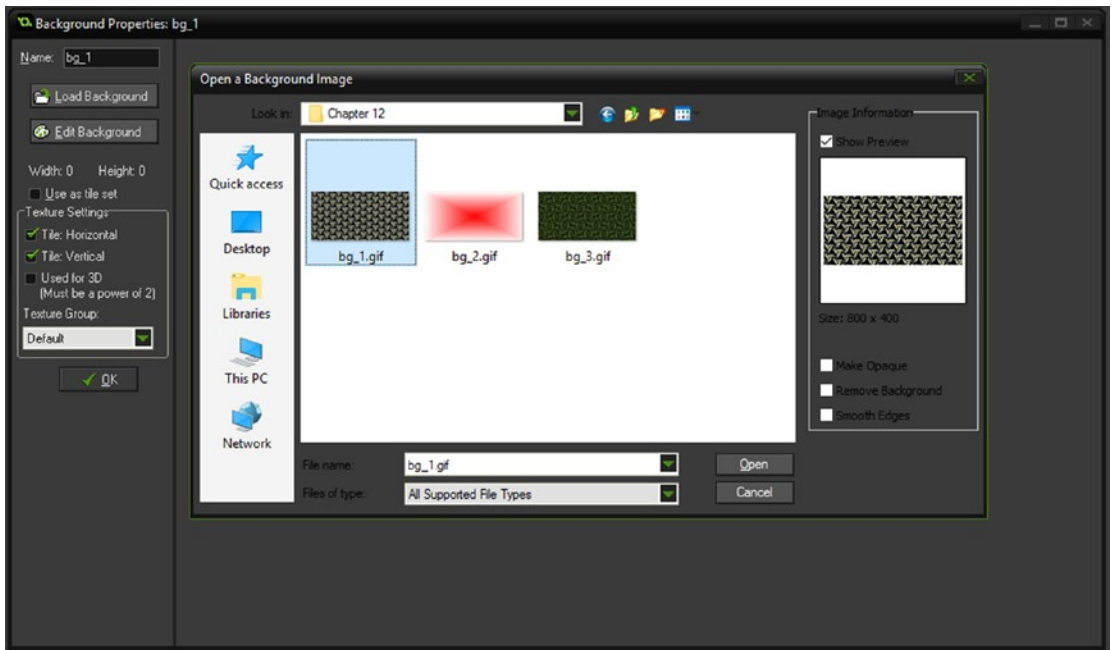


Figure 12-2. Loading in a background

You can set this background in a room. Create a new room, **room\_example**, and set the room dimensions as 800 by 400; you can do this from the **settings** tab, as shown in Figure 12-3:

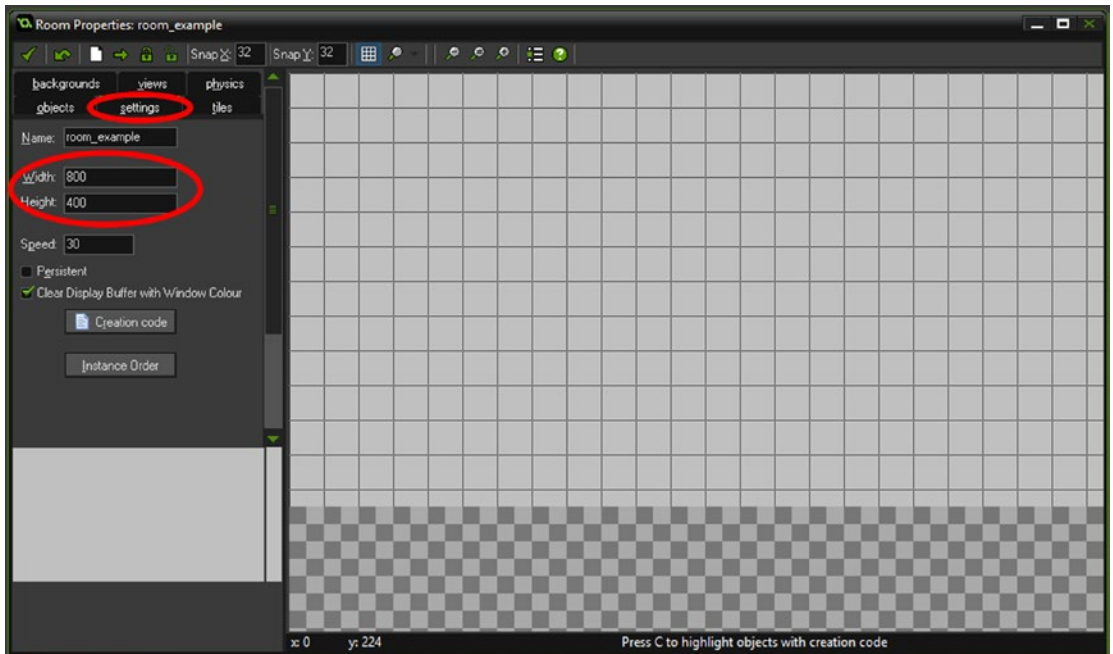


Figure 12-3. Setting room dimensions

Set the background by clicking on the **background** tab, unchecking **draw background color**, checking **visible** when it starts, and choosing **bg\_1** that you just loaded. This process is shown in Figure 12-4.

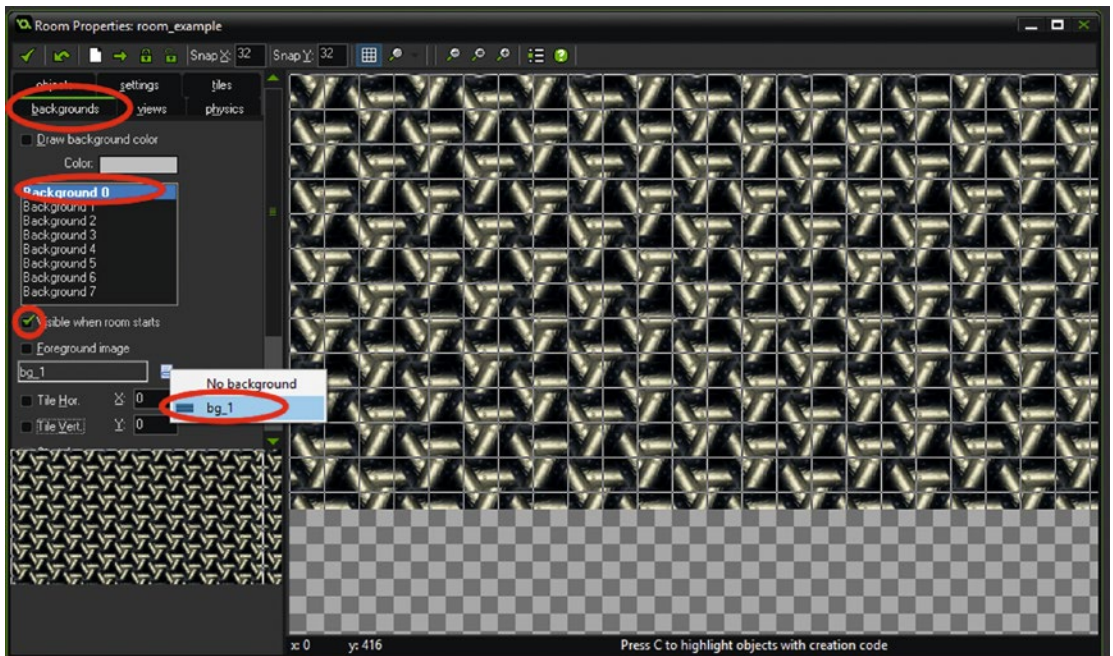


Figure 12-4. Setting a background

Rooms are dealt with in more detail in Chapter 13.

Next we'll look at the **views** tab. You set a view so it only shows part of the room at any one time. This is useful if you have a large room and only want to show the part where the player is, for example.

**Create a new project.**

Create new object, **obj\_player** and create and assign a red sprite (32x32 is fine) for it. In a **Step Event** put:

```
x+=4*(keyboard_check(vk_right)-keyboard_check(vk_left));  
y+=4*(keyboard_check(vk_down)-keyboard_check(vk_up));
```

Create another object, **obj\_wall**, and create and assign a blue sprite for it. No code is needed for this. Create a room, set the name as **room\_example** and the room width and height to 2000 each. Place one instance of **obj\_player** and multiple of **obj\_wall**. It doesn't matter too much where you place them. An example is shown in Figure 12-5.

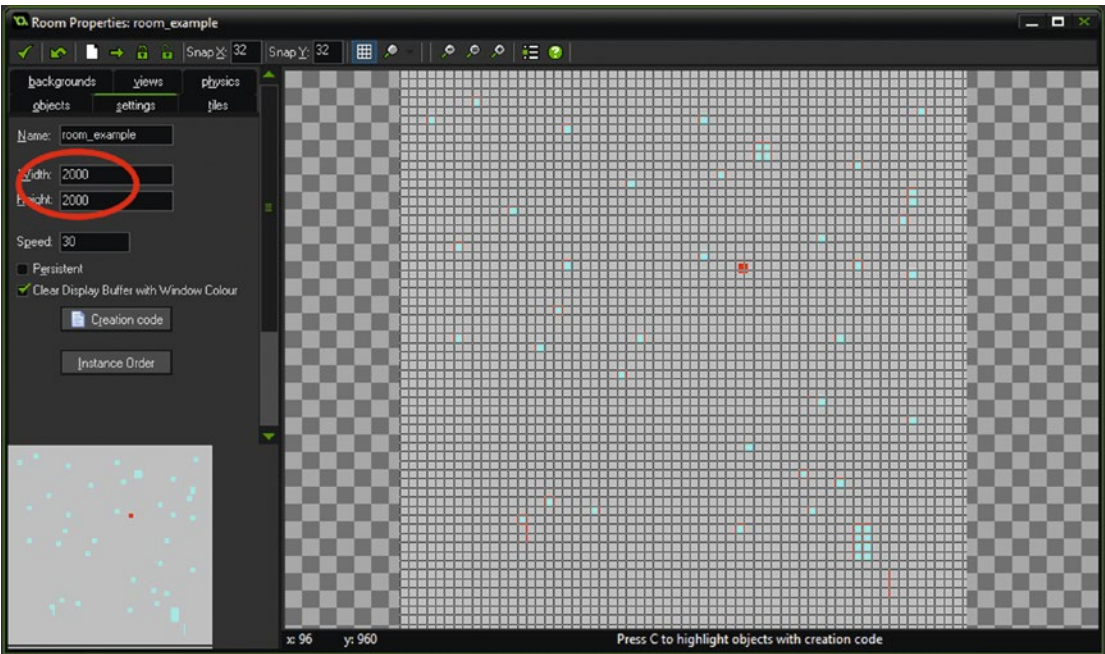
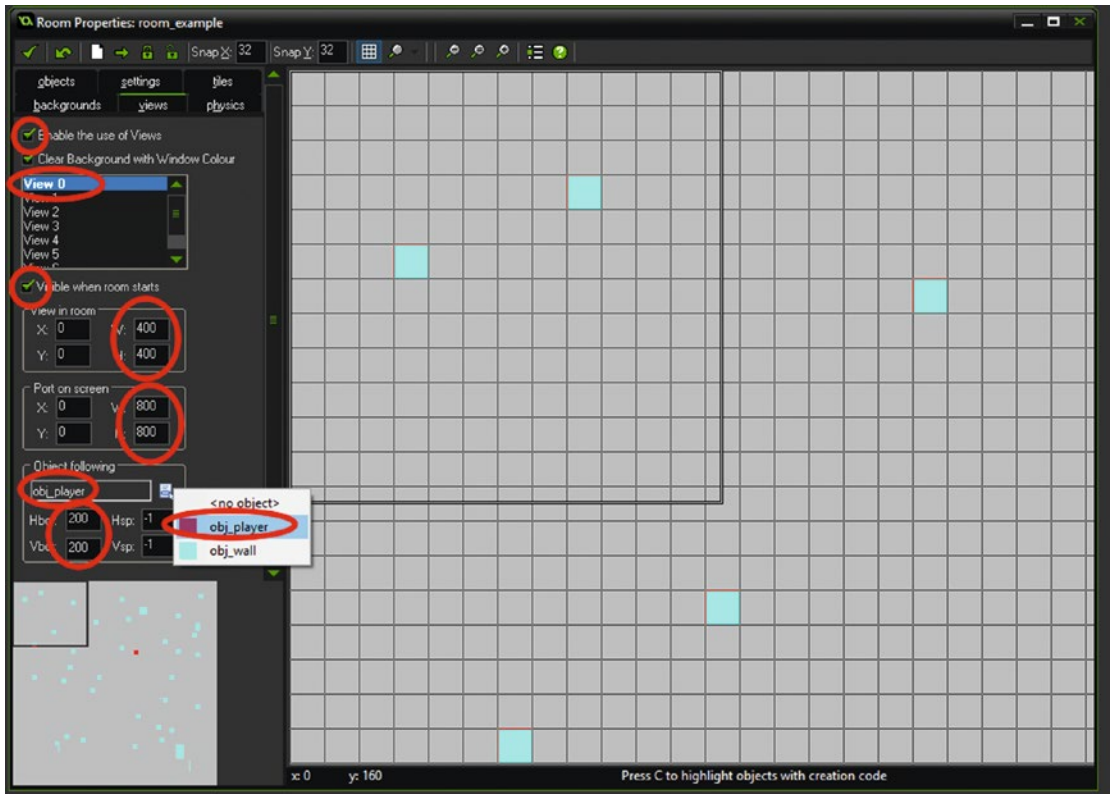


Figure 12-5. Showing objects added to room – step 3

Set up a view, as shown in Figure 12-6. This will create a view that keeps the player visible on the screen. Note: If the view and the viewport have different aspect ratios, this will cause unwanted stretching.



**Figure 12-6.** Setting up view to follow an object

An example for this is in the resources folder at: **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 12**

A view can be used to show part of the room. For example you may have a huge level that's 2000 by 2000 pixels – a view can be used just show a part, maybe 800 by 400 at any one time.

There is a number of built-in functions for backgrounds and views. If you're feeling adventurous, look these up in the manual, by pressed F1.

There's a ton of GML for using with rooms. The main ones are these:

This one will take you to a named room: `room_goto(room_name);`

This GML will take you to the next room (as shown in the resource tree): `room_goto_next();`

This GML will take you to the previous room (as shown in the resource tree): `room_goto_previous();`

You can restart the room using: `room_restart();`

## Worksheet – Rooms

1. Check your understanding
  - A) What are rooms?
  - B) How do you add objects to an existing room?
  - C) Can you draw sprites to a room using the room editor?
  - D) How do you add backgrounds to existing rooms?
  - E) How do you set room size?
  - F) How do you create a view of 200 by 200?

Correct the code in the following:

```
goto_room(room_level_2);  
goto_room_next;  
restart_room(room_level_1);  
if (room_exists(room_boss)) room_boss;
```

True or False

- A) There is a limit of 20 rooms. T / F
- B) `room_goto_next()`; will take you to the next room in the resource tree. T / F
- C) You can change views without restarting the room. T / F
- D) Rooms can have more than one background. T / F



## Worksheet – Rooms – Answer Sheet

1.

A) What are rooms?

Rooms are places where you show backgrounds, place objects, place tiles, and set views.

B) How do you add objects to an existing room?

Go to the object tab, select object, and place in the room by clicking.

C) Can you draw sprites to a room using the room editor?

Not directly, but you can place an object that can draw a sprite.

D) How do you add backgrounds to existing rooms?

Load in and name the background, go to the room editor, and apply it under backgrounds tab.

E) How do you set room size?

You can set this in the settings tab.

F) How do you create a view of 200 by 200?

Go to view tabs and set width and height of view.

Correct the code in the following:

A) `goto_room(room_level_2); room_goto(room_level_2);`

B) `goto_room_next; room_goto_next();`

C) `restart_room(room_level_1); room_restart();`

D) `if (room_exists(room_boss)) room_boss;`

`if (room_exists(room_boss)) room_goto(room_boss);`

True or False

A) There is a limit of 20 rooms T / F

B) `room_goto_next();` will take you to the next room in the resource tree. T / F

C) You can change views without restarting the room. T / F

D) Rooms can have more than one background. T / F

## Basic Projects

- A) Make a splash screen with a background that shows for 5 seconds, plays a sound, and then goes to a new room.  
3 Points
- B) Create a level select screen that has 5 buttons that each go to a different room. Make the buttons change colour when a mouse is over them. Draw the level as text in the middle of each button. Remember to set up text drawing correctly.  
3 Points

## Advanced Project

- C) Create 2 rooms, A B. Visualize them as:



Make the player wrap up and down in each room.

Make it so a player object can move from one room to the next. So if the player moves off the right of room A, the player will appear at the same Y location in room B, but on the left of the room; and, if the player moves off the right of room B, the player will appear at the same Y location in room A, but on the left of the room. Do this for moving left also.

4 Points

## End of Book Game Rooms

This game consists of a number of rooms. All rooms will have the same dimensions, 800 by 400.

They are:

- Splash screen - displays logo and initiates global game variables
- Menu - shows unlocked level and buttons to restart / quit
- Shop - allows player to buy weapons
- Game Level - this the game level the player controls
- Game over - player goes here if they lose all their lives
- Game complete - player goes here if they successfully complete the game

Think about what objects would need to go in each room and what they do.

## CHAPTER 13



# Backgrounds

Backgrounds are one of the basic resources that are used for rooms in which the game takes place. Backgrounds can be made of one large image or tiles of a smaller image. Additionally, you can have static and moving backgrounds in GameMaker: Studio. These backgrounds can be set using the room editor, or in GML. You can also set and change backgrounds using GML.

Backgrounds are images that show in the room but do not have any direct interaction with objects.

You can combine two or more backgrounds to create a parallax scrolling effect.

Some uses of backgrounds include:

- Splash screens
- Backgrounds for levels
- Moving backgrounds for infinite runner type games

Let's make an example by creating a new project. We are going to create a moving background. This is useful as it creates a more professional look to the game, and it is useful for scrolling type games. Load in a background from the resources, and name it as **bg\_wire**. Set it as shown in Figure 13-1.



Figure 13-1. Setting up a background

Now create a new room, **room\_example**, and set the dimensions as 800 by 400. Set the background as shown below in Figure 13-2:

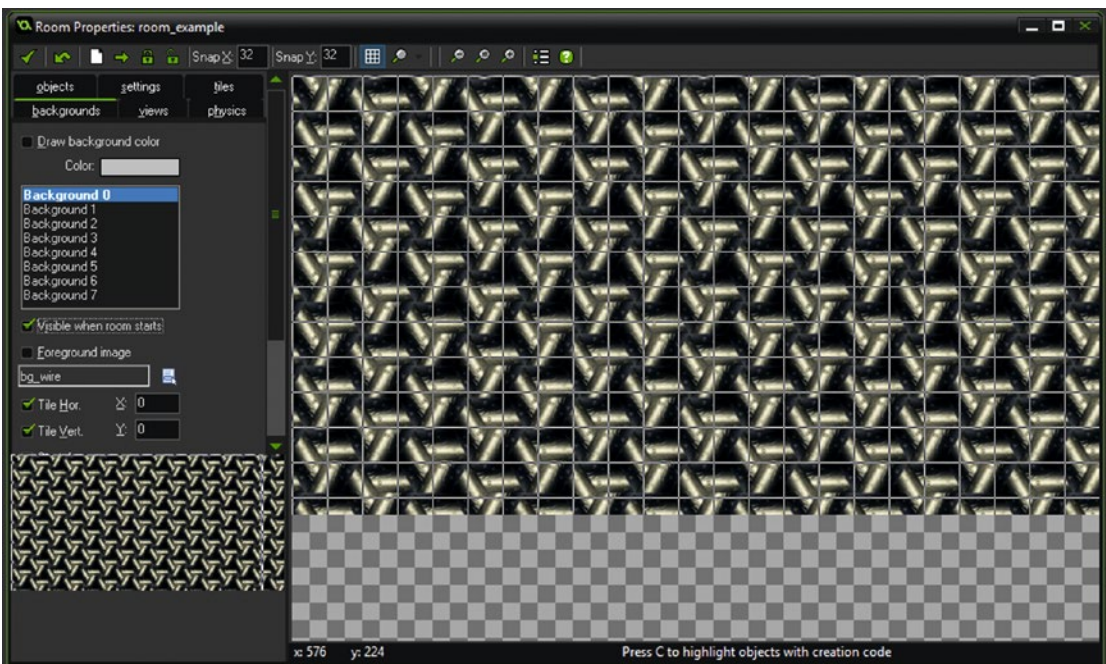


Figure 13-2. Setting background for the room

Now test your game; you should now see a room with background showing.

Now create an object **obj\_view\_control**.

In a **Step Event** put in the following code; this will move the background horizontally 5 pixels each step:

```
background_x[0]+=5;
```

Place one instance of **obj\_view\_control** in the room. Now test your game again, and the background will now move. Similarly you can set the vertical movement:

```
background_y[0]+=5;
```

You can set horizontal and vertical movement in the **background tab** of an open room. Also you set a constant speed for a background, for example, as shown below:

---

■ **Note** These don't have to be in the **Step Event**.

---

```
background_hspeed[4] = 2;
background_vspeed[4] = 1;
```

There are a heap of other background functions; see the manual if you're interested. A room's background colour can be set:

```
background_colour= c_blue;
```

You can make a background move vertically using:

```
background_vspeed[0] = speed;
```

and horizontally using:

```
background_hspeed[0] = speed;
```

You can have up to eight backgrounds, for example **background\_index[0]** or **background\_index[7]**.

You can set a loaded background using:

```
background_index[5] = bg_trees;
```

You can set a background to visible, for example:

```
background_visible[2]=true;
```

If you have another background visible, you can set it to invisible using:

```
background_visible[1]=false;
```

You can also check if a background is visible:

```
if background_visible[0]
{
    //do something
}
```

## Worksheet - Backgrounds

1. True or False:

Backgrounds can be animated. T / F

Backgrounds can be scrolled using GML code. T / F

Backgrounds can be tiled a limited number of times. T / F

2. Use the manual ([docs.yoyogames.com](http://docs.yoyogames.com)) to look up the following:

A) `background_vtiled[1]`

B) `background_yscale[4]`

Explain what each of the above does:

A)

B)

## Worksheet – Backgrounds – Answer Sheet

1. True or False:

Backgrounds can be animated. **False** Note: It can if you change it every frame, but not on its own. But this is correct to some point at the beginner level.

Backgrounds can be scrolled using GML code. **True**

Backgrounds can be tiled a limited number of times. **False** Note: It can be tiled up to eight times by setting eight backgrounds, each with their horizontal and vertical repeating disabled. But this is correct to some point at the beginner level.

Comment

2. Use the manual to look up the following:

A) `background_vtiled[1]`

B) `background_yscale[4]`

Explain what each of the above does:

A) Sets vertical tiling of background

B) Can be used to stretch the background



## Basic Projects

- A) Make a program sets whether backgrounds are visible when keys 1 to 5 are pressed.  
2 Points
- B) Make a background that scrolls left after 5 seconds.  
2 Points

## Advanced Projects

- C) Create two horizontal views, one which follows `obj_player` and one which follows `obj_enemy` (make both objects movable using key presses).  
2 Points
- D) Create a parallax system using at least four backgrounds. Have the backgrounds move to the left, with the top layer moving the fastest. Change the Y location of the background proportionate to player objects y location.  
4 Points
- Background assets are available for this in the downloadable resources, in the folder Assets Used In Book ► 13.

## End of Book Game Backgrounds

This game consists of a number of rooms, and each room will need some form of background to make the game look better.

The splash, menu, shop, game level, game over, and game complete rooms.

For the level, we'll change the background that's being show by changing the `background_index[0]` to a different background that has been loaded in.

## CHAPTER 14



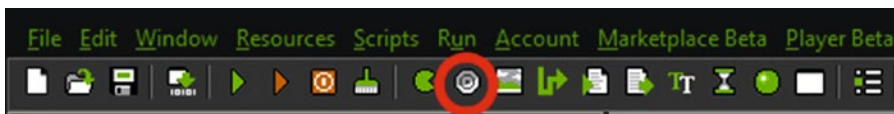
# Sounds and Music

Sounds and music are very important in games. The correct style of music can set the scene for the game: a horror-themed game would require different music than an RPG. Sound effects and voices can provide feedback too. For example when you buy an item in a shop, you want audio confirmation the purchase has gone through. When you fire a weapon or swipe a sword you want to hear a reassuring sound. This section serves as an introduction to playing sounds and music in GameMaker.

A few sound resources are included in the resources download for you to play with. Sounds can be used for the following:

- Explosions effects
- Button clicks
- Giving player feedback using voices
- Playing background music
- Jumping and landing effects
- Collision sounds

You can create a new sound by clicking the **Create a sound** button on the title bar as shown in Figure 14-1.



*Figure 14-1. Creating a new sound*

Load in the sound effect, **snd\_tank**, and name the sound **snd\_tank**; this step is shown in Figure 14-2.

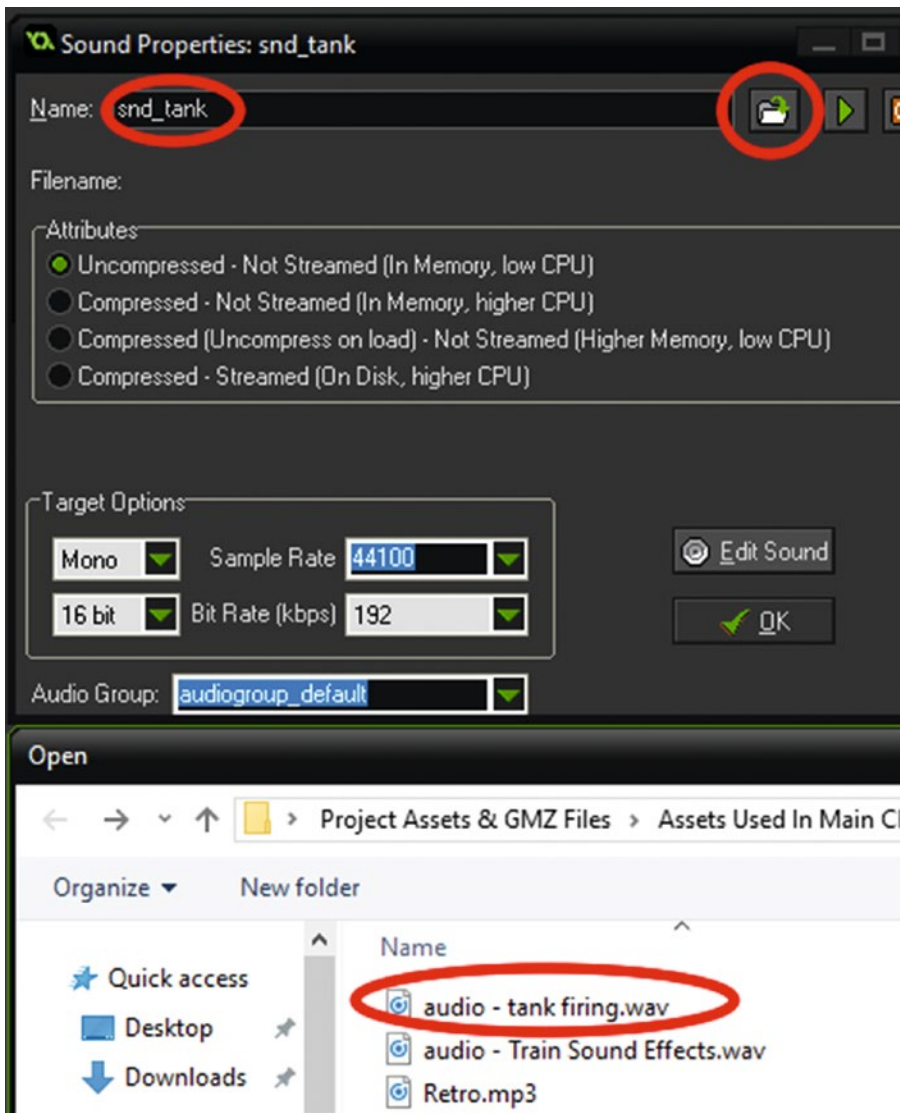


Figure 14-2. Loading a sound file using default settings

Create an object, **obj\_example**, and put the following code in the **Step Event**:

```
if keyboard_check(ord('X'))
{
    audio_play_sound(snd_tank,1,false);
}
```

The **snd\_tank** is the sound file to play, **1** is the channel priority, and **false** stops the sound from looping. Then create a room, **room\_example**, and place one instance of **obj\_example** in the room.

Test this game, when you press the X key, the sound effect will play.

Add a new sound, **snd\_music\_1** and load **Retro.mp3** from the resources. Put the following code in to a **Create Event** of object **obj\_example**.

```
audio_play_sound(snd_music_1,1,true);
```

Test again, and a background track will play. Check that you can hear the tank sound over the background music.

You can lower the volume of the background music, and make the sound effect easier to hear:

```
bg_music=audio_play_sound(snd_music_1,1,true);
audio_sound_gain(bg_music, 0.2, 0);
```

You can check if a sound is playing, and stop it if it is:

```
if (audio_is_playing(snd_music_1))
{
    audio_stop_sound(snd_music_1);
}
```

An example for this, including pause and resume, is in the resources: **Project Assets & GMZ Files** ► **Assets Used In Main Chapters** ► **Chapter 14**

Some additional functions include:

```
audio_pause_sound
audio_resume_sound
```

An example using the pause and resume functions could look like the following, which would pause the sound on keypress P and resume on keypress R:

**Create Event:**

```
audio_play_sound(snd_music_1,1,true);
```

**Step Event:**

```

if keyboard_check_pressed(ord('P'))
{
    if audio_is_playing(snd_music_1)
    {
        audio_pause_sound(snd_music_1);
    }
}
if keyboard_check_pressed(ord('R'))
{
    if audio_is_playing(snd_music_1)
    {
        audio_resume_sound(snd_music_1);
    }
}

```

You can also stop all sounds. Stopped sounds cannot be resumed.

`audio_stop_all`

Here is an example of this being used, which would stop any and all audio when X is pressed:

**Create Event:**

```
audio_play_sound(snd_music,1,true);
```

**Step Event:**

```

if (keyboard_check_pressed(ord('X')))
{
    audio_stop_all();
}

```

## Worksheet – Sounds & Music

1. Cross out the audio function that doesn't exist in GameMaker: Studio:

- audio\_exists
- audio\_play\_sound
- audio\_play\_all
- audio\_pause\_sound
- audio\_pause\_all
- audio\_resume\_sound
- audio\_resume\_all
- audio\_stop\_sound
- audio\_stop\_all
- audio\_is\_playing
- audio\_is\_paused

2. How would you do each of the following in GML:
  - a. Play a sound effect when a ball bounces off of a wall.
  - b. Play a sound when X is pressed, for example shooting a bullet.
  - c. Go to a room when a music track stops playing, for example, when splash screen music has finished.

## Worksheet – Sounds & Music – Answer Sheet

1. Cross out the audio function that doesn't exist in GameMaker: Studio:

-audio\_play\_all – **This doesn't exist**

2. How would you do each of the following in GML:

There are few possible ways to achieve any tidy code that does what's required or would be suitable.

- a. Play a sound effect when a ball bounces off of a wall.

In a Collision Event put:

```
audio_play_sound(snd_bounce,1,false);
```

- b. Play a sound when X is pressed, for example, shooting a bullet.

This can be placed in the **Step Event**:

```
if (keyboard_check_pressed(ord('X')))
{
    audio_play_sound(snd_effect,1,false);
    instance_create(x,y,obj_bullet);
}
```

- c. Go to a room when a music track stops playing.

Placed in a Step Event:

```
if !audio_is_playing(snd_music);
{
    room_goto(room_menu);
}
```



## Basic Projects

- A) Make a program that can play, pause, and stop a song.  
3 Points
- B) Play one of four sound effects at random when a ball collides with a wall.  
3 Points

## Advanced Projects

- C) Create a Juke Box program with five of your favorite songs.  
2 Points
- D) Create a movable player object, and a target object. If the player object is less than 200 pixels from the target object play a sound effect. Increase the sound volume the closer the mouse gets to it.  
(see `audio_sound_gain`)  
2 Points

## End of Book Game Sounds & Music

The game will include sounds and music.

The music will be single track that plays repeatedly in the background.

Sounds will include sound effects and voices, which will be triggered to play upon various conditions.

For example, when a bullet is fired, an appropriate sound effect will be played.

When a bullet collides with an asteroid an explosion sound effect will be played.

The game will also use a number of voices, which will provide information to the player, such as:

- A successful or unsuccessful purchase in the shop
- That the player has changed weapon
- Alert the player to no ammo
- Warn the player when **health** is getting low
- Notify the player that level has started
- Play a selection of different voices if player collides with an asteroid

To make it easier to call these sound effects, a couple of simple scripts will be written so you can call a sound effect using, for example:

```
scr_play_effect(snd_explosion_double_gun);
```

and similar for calling voices:

```
scr_voice(snd_voice_low_ammo);
```

Think about what sounds should be available, and write a list of voices that could provide informative feedback to the player.

## CHAPTER 15

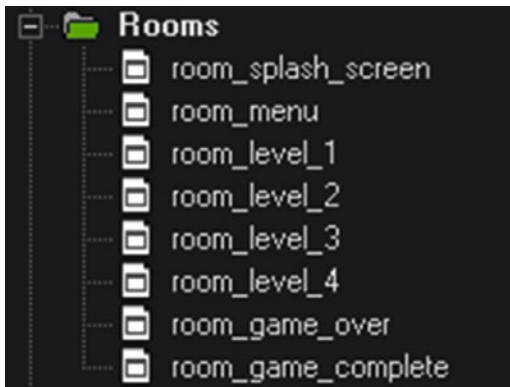


# Splash Screens and Menu

A splash screen is usually shown when a game starts up. It's an ideal time to initialize variables and load external resources. It's also a great way to show your company logo or a sponsor's message. You can create a splash screen in the **Global Game Settings**; however, normally GameMaker: Studio loads the game very fast and so the user will hardly have time to see it, which is another good reason to create your own. A good way to show a splash screen is to create a new room, assign a background, and then move it to the top of the room's resources tree, for example, as shown in Figure 15-1.

Menus are great ways to allow the player to select a difficulty, turn sound on or off, or visit an unlocked level.

A completed game rooms tree may look something like this:



**Figure 15-1.** Showing an example of room order in resource tree

As you will see above, in Figure 15-1, each room has a distinct name. The room at the top of the tree will be loaded first.

### **Start a new project within GameMaker: Studio.**

Create a new object, **obj\_splash\_screen**.

### **Create Event code:**

```
global.level=1;
lives=5; //Set initial lives to 5
score=0; //Start with a score of 0
global.bonus_score=0; //Set with a value of 0
health=100; //Start with full health
alarm[0]=7*room_speed; //Set alarm for 7 seconds
```

**Alarm[0] Event code:**

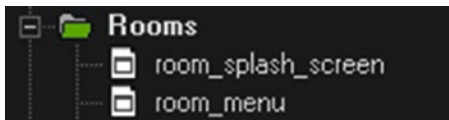
```
room_goto_next();
```

Load in the background from the resources folder, **Project Assets & GMZ Files > Assets Used In Main Chapters > Chapter 14**.

Create a new room, **room\_splash\_screen**, and set this new background to the room, making it visible on room start.

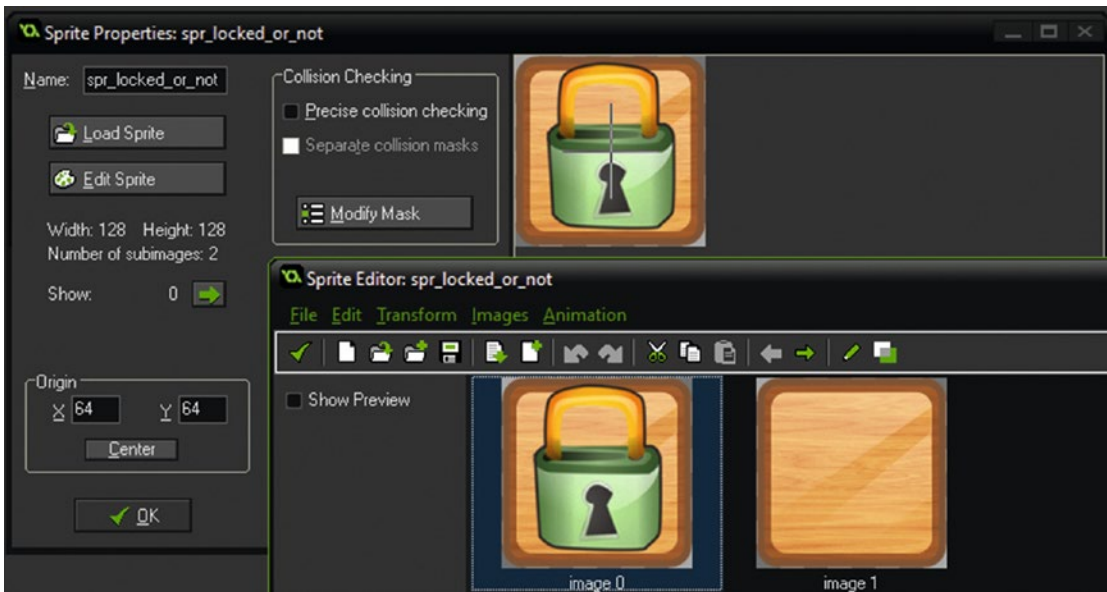
Then place one instance of **obj\_splash\_screen** in **room\_splash\_screen**.

Next we'll look at the menu room. Create a new room, **room\_menu**, and place it under the room **room\_splash\_screen** as shown in Figure 15-2:



*Figure 15-2. Showing room order*

Next create a new sprite, **spr\_locked\_or\_not**, and set up as shown in Figure 15-3, with two sub images loaded in:



*Figure 15-3. Loading in sub images and assigning to a sprite*

Create an object, **obj\_button\_parent**, and assign the sprite you have just created. In a **Step Event** put:

```
if global.level>=my_id locked=false else locked=true
```

```

if locked image_index=0 else image_index=1
if (!locked && position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_
left))
{
    if my_id==1 room_goto(room_level_1);
    if my_id==2 room_goto(room_level_2);
    if my_id==3 room_goto(room_level_3);
    if my_id==4 room_goto(room_level_4);
}

```

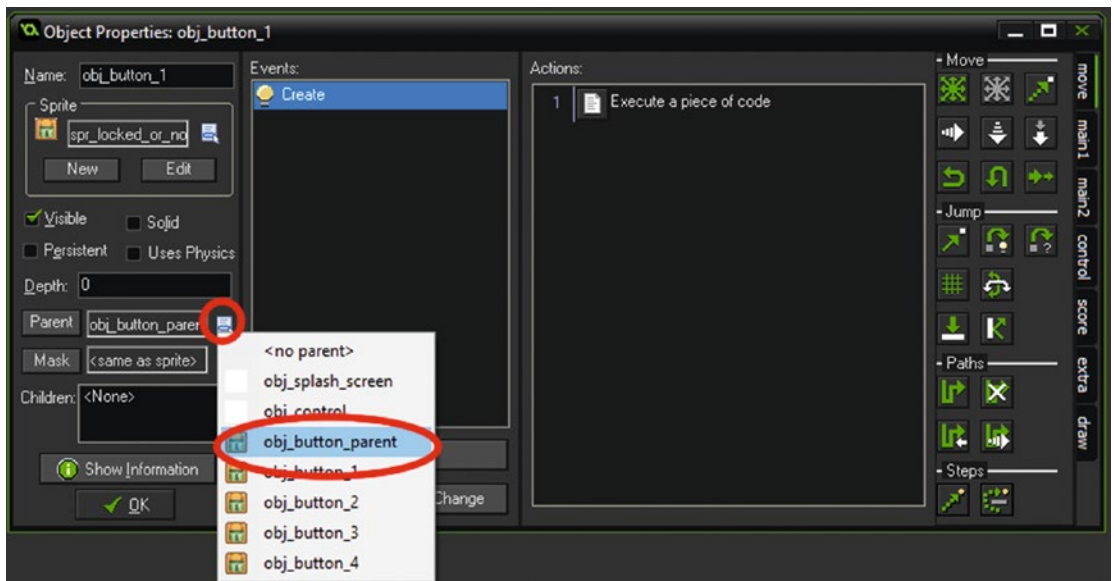
and in **Draw Event** put:

```

draw_self();
draw_set_font(font_lock);
draw_set_valign(fa_middle);
draw_set_halign(fa_center);
draw_set_colour(c_black); // These two lines create
draw_text(x-1,y-1,my_id); // a shadow effect
draw_set_colour(c_white);
draw_text(x,y,my_id);

```

That is all for this object. Create a new object, **obj\_button\_1**, and set the parent object as **obj\_button\_**  
**parent**, as shown in Figure 15-4:



**Figure 15-4.** Setting a parent object

Now add a **Create Event** to this object, **obj\_button\_1**, and put the following code

```
my_id=1;
```

Repeat this for **obj\_button\_2**, **obj\_button\_3**, and **obj\_button\_4**, setting the parent object, but changing the **Create Event** code as:

For **obj\_button\_2**:

```
my_id=2;
```

and **obj\_button\_3**:

```
my_id=3;
```

and **obj\_button\_4**:

```
my_id=4;
```

accordingly.

Now create four new rooms:

**room\_level\_1, room\_level\_2, room\_level\_3, room\_level\_4**

Next create a new object, **obj\_control**, but no sprite is needed. This is used so we can check it all works as planned.

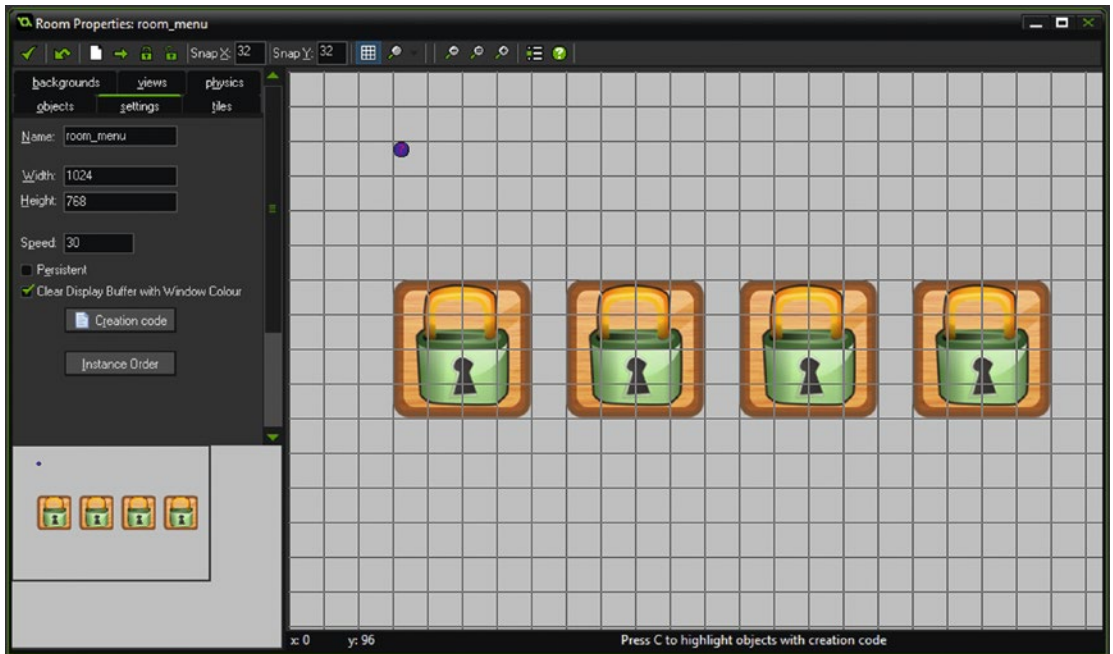
In a **Step Event** place:

```
if keyboard_check_released(ord('Q'))
{
    global.level+=1;
}
if keyboard_check_released(ord('A'))
{
    global.level-=1;
}
```

And in a **Draw Event**:

```
draw_text(10,10,global.level); //for testing only
```

Open room **room\_menu**, place one instance of **obj\_control** in the top left, and one of each of the button objects, in order, from left to right: **obj\_button\_1**, **obj\_button\_2**, **obj\_button\_3**, and **obj\_button\_4**. So it looks Figure 15-5:



*Figure 15-5. Showing room setup with objects placed*

You can now save and test this.

An example project in GMZ is available at: **Project Assets & GMZ Files** ► **Assets Used In Main Chapters** ► **Chapter 15**

## Worksheet – Splash Screens & Menu

Correct the following if there is a mistake:

```
if (room_exist(room_level_20))  
{goto_room, "room_level_20";}
```

True or False:

- A. All rooms must be the same size. T / F
- B. You can restart a room using GML. T / F
- C. The room at the bottom in the resource tree is the first room when a game starts. T / F
- D. You can't have two rooms with the same name. T / F
- E. If a room is set as persistent it will 'remember' where instances are. T / F

How would you code the following using GML?

- A. A button that changes its image index on mouse over? (both sub images are same size and shape)
- B. A bonus level button that becomes clickable when **score** exceeds 5000?
- C. A new button for a boss level becomes visible when levels 1 to 4 are completed?
- D. A button that rotates left and right slowly to 45°.



## Worksheet – Splash Screens & Menu – Answer Sheet

Correct the following if there is a mistake:

```
if (room_exists(room_level_20))
{
    room_goto(room_level_20);
}
```

True or False:

- A. All rooms must be the same size. **F**
- B. You can restart a room using GML. **T**
- C. The room at the bottom in the resource tree is the first room when a game starts. **F**
- D. You can't have two rooms with the same name. **F**
- E. If a room is set as persistent it will 'remember' where instances are. **T**

How would you code the following using GML?

- A. A button that changes its image index on mouse over? (both sub images are same size and shape).
- B. One approach is to have an object with a sprite that has two images and uses the following code:

**Create Event:**

```
image_speed=0;
```

**Step Event:**

```
if position_meeting(mouse_x, mouse_y, id)
{
    image_index=0;
}
else
{
    image_index=1;
}
```

- A. A bonus level button that becomes clickable when **score** exceeds 5000?

```
if position_meeting(mouse_x, mouse_y, id) && mouse_check_button_pressed(mb_left)
&& score>5000
{
    //do something
}
```

- B. A new button for a boss level becomes visible when levels 1 to 4 are completed?  
In **Draw Event**:

```
if global.level==5
{
    draw_self();
}
```

- C. A button that rotates left and right slowly to 45°.

**Create Event:**

```
clockwise=true;
```

**Step Event:**

```
if clockwise
{
    image_angle-=1;
    if image_angle<=-45 clockwise=false;
}
else
{
    image_angle+=1;
    if image_angle>=45 clockwise=true;
}
```

## Basic Projects

- A) Create a splash screen and place an object with a sprite animation. Set it to go to the next room 5 seconds after the animation has ended.

2 Points

- B) Create a menu room with background music with a button for it to go to an instructions room (with button to return to menu room), and a game room. Change the music for instruction screen and playing the game.

4 Points

## Advanced Project

- D) Create an object that becomes unlocked only if user types in a password on the keyboard. Make the password “xbacon.” Display visibly whether the object is locked or unlocked.

4 Points

## End of Book Game Splash Screens & Menu

The game will make use of a splash screen and menu.

The main purpose of the splash screen will be to define the initial data and load any saved INI file (if it exists).

The splash screen will use an alarm to take the player to the menu after a few seconds.

The game will need to store a number of variables so the player can return to the game and continue on their current level with the same stats as when they left.

The game will need to know the player's current level, **health**, amount of cash, and total bullets fired. It also needs to know weapon stats such as how many bullets they have left and how many shots have been taken of each weapon.

The level select screen will allow the player to play any unlocked level, quit the game, or restart it (deleting all data). Upon loading, the menu select screen will store all player data to the INI file. This is so when the player returns, they can continue with the next level.

## CHAPTER 16



# Random

A static game with the exact same pattern of movement doesn't have much replay value. This type of game can be memorized. Using randomness in your games to create variety in game play greatly increases the game replay value.

Strictly speaking there is no such thing as a true random number on a regular PC, but it can be simulated and approximated for game play.

For the purposes of making games easier to create and test, GameMaker: Studio will always generate the same random numbers each time a game is played. For example, if in your game you rolled a dice 5 times and got 5 3 1 6 3 2, the next time you played from the Studio IDE you'd also get 5 3 1 6 3 2. Depending on your game you may want different numbers each time when testing, so remember that a final executable will not display this behavior. To do this, use `randomize()`, which only needs to be called once at the start of the game.

For example, in the Needs event formatting of an object in your splash screen you may have:

```
randomize();
```

Then in your game you may have:

```
starting_x=irandom(800);  
starting_y=irandom(400);
```

Now each time the game is run, `starting_x` and `starting_y` will have different random values. Randomness can be used for such things as the following:

- Selecting an effect to use
- Making AI move / change direction
- Choose an attack from a selection
- Choose a random sound effect / backing music
- Move an object to a new location

An example of a random function:

```
attack=choose(1, 1, 1, 2, 2, 3);
```

This will choose a number at random. There will be a 50%(3/6) chance of getting a '1,' a 33%(2/6) chance of getting a '2,' and a 17%(1/6) chance of getting a '3.'

You can also use other things as well as numbers: for example, objects, sounds, colours:

```
music_track=choose(snd_track_1, snd_track_2, snd_track_3);
text_colour=choose(c_green, c_red, c_blue);
spawn_enemy=choose(obj_enemy_1, obj_enemy_2);
```

You can create a real random number:

```
number=random(50);
```

This will choose a random number between 0 and 50 exclusive, with decimal places.

An example of a value returned by the `random()` function: 44.7768221937 (which may contain more digits than this).

If you are generating real numbers with decimals, the following is useful; however `floor` is generally used for randomized values:

```
floor(4.2) would return 4
```

```
ceil(4.2) would return 5
```

```
round (4.2) would return 4
```

A method more suited to most applications, as it creates whole integers is:

```
number=irandom(50);
```

This will choose a whole number integer between 0 and 50 inclusive.

An example of this value is 38.

You can also choose a whole random integer number between values, for example, between 40 and 50 inclusive:

```
irandom_range(40, 50);
```

There may be times that you want the randomization to be the same each time, even when the game has been compiled to the final executable for distribution: for example, when you have a randomly generated level. To achieve this, you can use the following function:

```
random_set_seed(number);
```

or

```
randomize();
```

If you want a random outcome each time.

---

■ **Note** This should be placed just before any deterministic results are generated.

---

## Worksheet – Random

- A. With this code:

```
my_colour=choose(c_blue, c_blue, c_olive, c_orange);
```

What is the chance of each colour being chosen?

- B. How would you choose an integer number between 0 and 75?
- C. How would you move the object to a random x position between 150 and 450?
- D. What value would `floor(19.8)` return?
- E. What value would `floor(11.2)` return?
- F. What value would `ceil(0.06)` return?
- G. What value would `round(53.487)` return?
- H. What value would `round(1076.882)` return?
- You have five objects, **obj\_bonus\_1**, through to **obj\_bonus\_5**.
- I. How would you program it so each has an equal chance of being selected?
- J. How would you program it so that object **obj\_bonus\_1** has a 50% chance of being selected and the rest an equal chance?

## Worksheet – Random – Answer Sheet

- A. With this code:

```
my_colour=choose(c_blue, c_blue, c_olive, c_orange);
```

What is the chance of each colour being chosen? **Blue 50%, 25% for both olive and orange**

- B. How would you choose an integer number between 0 and 75? `irandom_range(0,75);`

Or equally:

```
irandom(75);
```

- C. How would you move the object to a random x position between 150 and 450?  
`x=irandom_range(150,450);`

- D. What value would `floor(19.8)` return? **19**

- E. What value would `floor(11.2)` return? **11**

- F. What value would `ceil(0.06)` return? **1**

- G. What value would `round(53.487)` return? **53**

- H. What value would `round(1076.882)` return? **1077**

You have five objects, **obj\_bonus\_1**, through to **obj\_bonus\_5**.

- I. How would you program it so each has an equal chance of being selected?

```
selected=choose(obj_bonus_1, obj_bonus_2, obj_bonus_3, obj_bonus_4, obj_bonus_5);
```

- J. How would you program it so that object **obj\_bonus\_1** has a 50% chance of being selected and the rest an equal chance?

```
selected=choose(obj_bonus_1, obj_bonus_1, obj_bonus_1, obj_bonus_1, obj_bonus_2, obj_bonus_3, obj_bonus_4, obj_bonus_5);
```



## Basic Projects

- A) Play a random sound every time the player presses the spacebar.  
2 Points
- B) Make an object jump to a random position, no closer than 50 pixels near the edge of the window, when clicked with the mouse. Use a sprite with multiple sub images. Stop the animation on the last sub image.  
2 Points
- C) Make an object move randomly around the room without going off of the edges. Make it change direction every 5 seconds. If it gets within 10 pixels of the edge of the screen, make it change direction away from the edge.  
2 Points

## Advanced Project

- D) Create a lottery game that chooses six different numbers between 1 and 49. Display them in ascending order, inside a circle. If the number is between 1 and 9 make the circle white, 10-19 blue, 20-29 green, 30-39 red, 40-49 yellow.  
4 Points

## End of Book Game Random

Randomization will only be used a few times in the game.

It will be used to randomly set the direction and rotational speed of asteroids, within a given range.

It will also be used to play random voices, again from a selection, at various points in the game.

## CHAPTER 17



# More Movement (Basic AI)

Most casual games involve a player and some form of computer AI (Artificial Intelligence). Basically an AI object will interact with what's onscreen and what the player does. For example, in a top-down game the AI object may move toward the player avoiding other objects and shoot a bullet toward the player if they're within a certain range. Usually in the first few levels of a game the AI object will be more forgiving to the player, but higher levels may make the AI more aggressive.

Some of the functions the AI may have:

- Move toward player
- Fire a bullet / missile toward player
- Decide which card to play next
- Punch the player if they're not blocking

In most cases you'll be checking whether a variable equals some number, or if a function is **true** or **false** – and use this to create the illusion of AI.

This starts with a very basic AI and progressively makes it more intelligent.

We'll create a basic AI system that makes an enemy move toward the player, if there is a direct line of sight.

**Start a new project.**

Create three new objects, **obj\_wall**, **obj\_player**, and **obj\_enemy**.

Load in the appropriate sprites from the resources at: **Project Assets & GMZ Files > Assets Used In Main Chapters > Chapter 17**

Place the following code in the **Step Event** of **obj\_player**. This code will allow the player to move so long as it does not collide in the direction it is traveling with an **obj\_wall** instance:

```
hor=4*(keyboard_check(vk_right)-keyboard_check(vk_left));
vert=4*(keyboard_check(vk_down)-keyboard_check(vk_up));
if place_free(x+hor,y)
{
    x+=hor;
}
if place_free(x,y+vert)
{
    y+=vert;
}
```

That is all for this object.

Open up object **obj\_enemy** and put the following code in a **Create Event**. This will create a flag that can be changed and tested upon, with true if it has a direct line of sight to the player, or false otherwise:

```
can_see=false;
```

In the **Step Event** of this object put:

```
//if there is a direct line of sight between the player and the enemy then set can_see to
true, otherwise false
if (collision_line(x, y, obj_player.x, obj_player.y,
obj_wall, true,false))!=noone
{
    can_see=false;
}
else
{
    can_see=true;
}

if can_see
{
    mp_potential_step( obj_player.x, obj_player.y, 2, true);
}
```

`collision_line` returns whether an imaginary line between it and the player instance collides with an instance of **obj\_wall**. It returns false if it collides with **obj\_wall**.

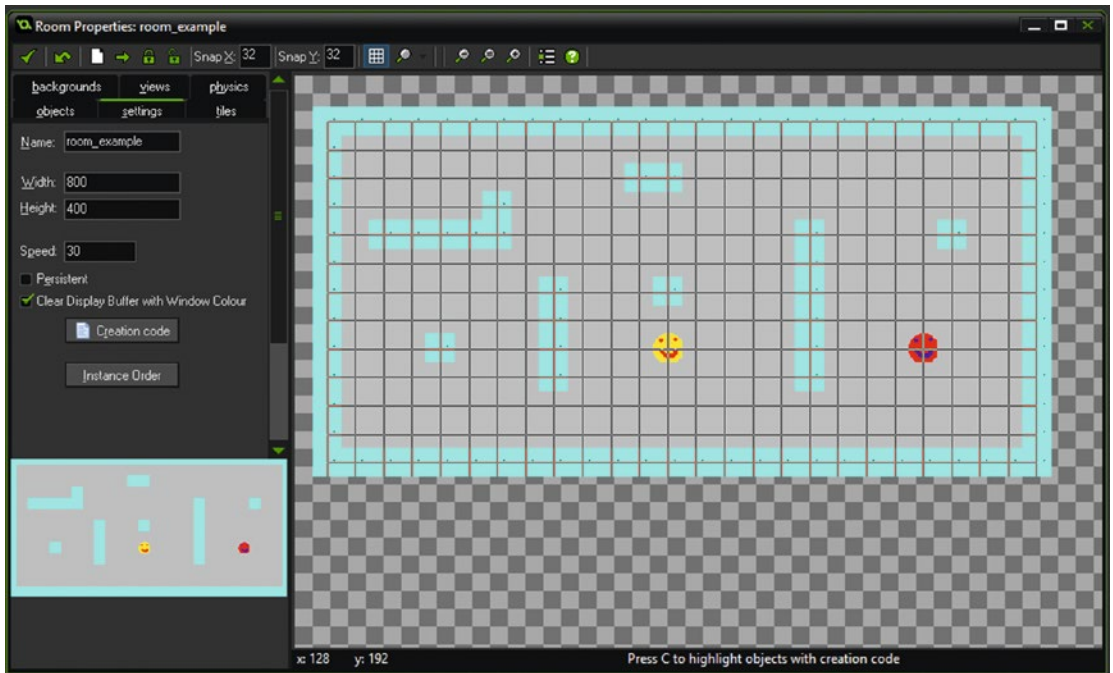
`mp_potential_step( obj_player.x, obj_player.y, 2, true);` moves the object 2 pixels towards the player for each frame.

Finally create a **Draw Event** for this object with the following code:

```
draw_self();
if can_see
{
    draw_line(x, y, obj_player.x, obj_player.y);
}
```

This will draw its own sprite, and a line between itself and player if `can_see` is true.

Set the room up as shown in Figure 17-1:



*Figure 17-1. Showing example room layout*

This project is available in the resources at: **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 17 ► example17a**

This next part continues adding to the example created above. Do not start a new project.

First load in and assign the sound file, **snd\_ouch**, from the resources.

Next create a new object, **obj\_bullet** and load in assign the **spr\_bullet** from the resource.

In the **Create Event** of **obj\_bullet** put:

```
direction=point_direction(x,y,obj_player.x, obj_player.y);
speed=3;
```

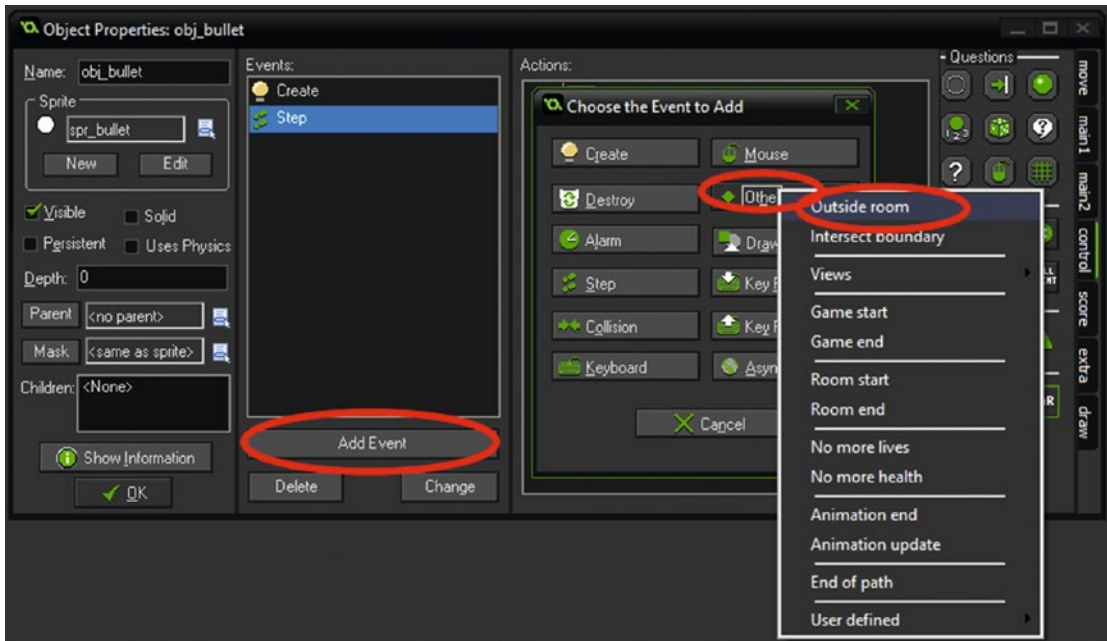
In a **Collision Event** with **obj\_player** put:

```
audio_play_sound(snd_ouch,1,false);
instance_destroy();
```

The above code will play the sound on collision with the player instance, then destroy itself.

Instead of playing a sound you could do things such as removing a life or making a graphical effect.

Finally for this part, we'll set the **obj\_bullet** to destroy itself when outside the room, using the **Outside Room Event**, as shown in Figure 17-2:



**Figure 17-2.** Setting an outside room event

Put this code in this event:

```
instance_destroy();
```

That is all for this object.

Open up object **obj\_enemy**, and change the **Create Event** code to:

```
can_see=false;
timer=0;
```

This will initialize the variable **timer** to 0.

And change the **Step Event** for **obj\_enemy** to:

```
//if there is a direct line of sight between the player and the enemy then set can_see to
true, otherwise false
if (collision_line(x, y, obj_player.x, obj_player.y,
obj_wall, true,false))!=noone
{
    can_see=false;
}
else
{
    can_see=true;
}
```

```

if can_see
{
    mp_potential_step( obj_player.x, obj_player.y, 2, true);
    timer++;
}
else
{
    timer=0;
}

if timer==10
{
    instance_create(x,y,obj_bullet);
    timer=0;
}

```

This change will make the variable `timer` count up if it can see the player instance. If it reaches 10 it will create a bullet instance and reset the timer. If the player is hidden, the timer will be set to 0.

You can now save and test the game. An example project is available at:

**Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 17 ► example17b**

This next part continues on from the previous example; do not start a new project.

Create a new room, **room\_splash**. Ensure this sits above the room **room\_example** in the resources tree.

You can change its position by clicking and holding the left mouse button, then move it by dragging it.

Set the room size to 800 by 400.

Create a new object **obj\_splash** and place the following in its **Create Event**:

```

global.enemy=0;
global.player=0;
room_goto_next();

```

Place one instance of this object in the room **room\_splash.room\_splash**

That is all for this object.

You can delete the object **obj\_bullet** and sprite **spr\_bullet** as they are not required.

To delete a resource from the resource tree you can right-click on it and select delete.

Create a new object, **obj\_star** and load in and set the appropriate sprite from the resources folder.

In a **Create Event** put:

```

move=false;

```

In the **Step Event** of this object, put the following code:

```

if move
{
    while(!place_free(x,y))
    {
        x=irandom(800);
        y=irandom(400);
    }
    move=false;
}

```

In a **Collision Event** with **obj\_enemy** put:

```
global.enemy+=1;
move=true;
```

In a **Collision Event** with **obj\_player** put:

```
global.player+=1;
move=true;
```

This will find a random empty position in the room.

If it collides with the player or enemy it will award a point and then seek a new empty position.

Place one instance of this object into **room\_example**.

Next change the **Create Event** code of object **obj\_enemy** to:

```
can_see=false;
```

And its **Step Event** code to:

```
if instance_exists(obj_star)
{
    mp_potential_step( obj_star.x, obj_star.y, 2, false);
}
```

This code above will move towards the **obj\_star** instance, avoiding objects.

The **Draw Event** is no longer required, as a default drawing will be used. You can remove it by right-clicking and choosing delete.

Finally create an object **obj\_score**, and place the following in the **Draw Event** (or **Draw GUI Event**):

```
draw_text(100,50,"Player:"+string(global.player));
draw_text(200,50,"Computer:"+string(global.enemy));
```

Place one instance of this object in room **room\_example**.

You can now save and test.

An example project is available at:

**Project Assets & GMZ Files** ► **Assets Used In Main Chapters** ► **Chapter 17** ► **example17c**



## Worksheet – More Movement

Using the game created previously:

- A. How would you change the game to make the bullets move faster when the player reaches a **score** of 10 or more?
- B. How would you add a sound effect when the bullet is fired?
- C. How would you make an enemy only follow the player's X position?
- D. How would you add a sound effect when a bullet is created?

Correct the mistakes in the following code:

- A. `alarm[7]=roomspeed*10;` (roomspeed is not a declared variable)
- B. `instance_create(obj_star, x, 10);`
- C. `draw_self;`
- D. `draw_text(lives, 50, 100);`
- E. `if (keyboard_check(ord('d'))){x+=5;}`

## Worksheet – More Movement – Answer Sheet

Using the game created previously:

- A. How would you change the game to make the bullets move faster when the player reaches a **score** of 10 or more? One method is:

**Create Event**

```
global.faster=false;
```

**Step Event**

```
if (score>=10)
{
global.faster=true
}
```

- B. Bullet object **Create Event**:  
 C. `direction=point_direction(x,y,obj_player.x,obj_player.y);`

```
speed=4;
if (score >= 10) { speed *= 2; }
```

- D. How would you make an enemy only follow the player's X position?

```
x=obj_player.x;
```

- E. How would you add a sound effect when a bullet is created?

In **Create Event** of bullet object:

```
audio_play_sound(snd_fire,0,false);
```

- F. How would you make an enemy only follow the player's x or y position?

In enemy's **Step Event**:

```
x=obj_player.x
```

Correct the mistakes in the following code:

- A. `alarm[7]=roomspeed*10; alarm[7]=room_speed*10;`  
 B. `instance_create(obj_star, x, 10); instance_create(x,10,obj_star);`  
 C. `draw_self; draw_self();`  
 D. `draw_text(lives, 50, 100); draw_text(50,100,lives);`  
 E. `if (keyboard_check(ord('d'))) {x+=5;} if (keyboard_check(ord('D')))  
{x+=5;}`

## Basic Projects

- A) Create a control object that creates a star every 5 seconds at a random position at the top of the screen that then falls toward the bottom. Player gets a point for every one collected. Player can only move left and right at the bottom of the screen, without being able to leave the window. Use the control object to draw the score in the top left of the window, remembering to set a font, drawing style, and colour.

2 Points

- B) Make a movable player object (using arrow keys) and a static enemy object. Make the enemy shoot at the player, getting faster as the player gets closer. Prevent the player from leaving the window. Ensure the bullet gets destroyed when leaving the room; use `room_width` & `room_height` for this.
- (See `distance_to_object` in the help file).

4 Points

## Advanced Projects

- C) Create an enemy that changes direction every 5 seconds to move away from the player, and wraps around the room if it goes off of the edge. Also add the bullet function from project 17 B, but make the firing speed no more than two seconds between shots.

4 Points

## End of Book Game

This game will only have one AI object, a spaceship that will hunt down the player object.

It will be set up to spawn when a large asteroid is created. It will slowly move toward the player and fire bullets at the player as it does so.

This is only very basic AI, but it adds an extra dimension to the game.

## CHAPTER 18



# INI Files

With casual games that people may only play for a few minutes at a time, the ability to store the player's progress is a required feature. Initialization or INI files provide an easy way to save and load data.

With INI files you can pretty much save any data, but the main ones you're most likely to save are these:

- Player's name
- Health
- Score
- Lives
- High Score
- Distance traveled
- Enemies killed

INI files involve two main elements:

```
[section]
```

```
and
```

```
key=data
```

A simple ini file would look like this:

```
[player]
top_score=100
name="Bob"
```

You can save this as an INI file (use a text editor and save as **savedata.ini**) and add it to the included files on the project tree. You can do this by right-clicking on **Included Files** and selecting **Create Included File**. Note: On most exports it is not required to include an INI as an included file. This is not the case in HTML5, in which you must include one.

To load data from this file you can use:

```
ini_open("savedata.ini");
global.top_score=ini_read_real("player","top_score",0);
global.player_name=ini_read_string("player","name","");
ini_close();
```

What the above code does:

Looks for a file named ***savedata.ini***

Sets the variable `global.top_score` to the value stored in the INI file. If the file or section and key do not exist it sets `global.top_score` to a default value, in this case 0 (the value after the last ,).

Sets the variable `global.player_name` to the value stored in the INI file. If the file or section and key do not exist it sets `global.player_name` to a default value, in this case "" (the value after the last ,).

It then closes the open INI file. When you are done reading / writing with INI files, it's a good practice to close it immediately after you're done.

You can also write variables to an INI file. For example:

```
ini_open("savedata.ini");
ini_write_real("level_section", "level", global.level);
ini_write_real("distance_section", "distance", global.distance);
ini_close();
```

This will store the current values of `global.level` and `global.distance`. If the values were 3 and 287, then the created INI file would look like this:

```
[level_section]
level=3
[distance_section]
distance=287
```

`ini_read_real()` and `ini_write_real()` work with real numbers.

`ini_read_string()` and `ini_write_string()` work with strings.

## Worksheet – INI Files

Given the Following INI file, *game\_data.ini*:

```
[level_info]
current_level=5
enemies_destroyed=2056
[player_info]
player_name=Bob
player_score=12768
player_best_score=17456
player_health=45
player_lives=2
[game_info]
total_games_played=11
total_enemies_destroyed=167328
```

What code would you use (including opening and closing the INI file) to:

Load `player_name`, `player_score`, `player_health`, `player_lives` on game start and store each as a global variable?

At the end of the level add the value of the enemies destroyed in that level to value of `total_enemies_destroyed`?

At the end of game, increment the value of `total_games_played`?

At end of game, if the player's **score** is greater than `player_best_score`, update the value?

## Worksheet – INI Files – Answer Sheet

Load `player_name`, `player_score`, `player_health`, `player_lives` on game start and store each as a global variable?

```
ini_open("game_data.ini");
global.player_name = ini_read_string("player_info", "player_name", "");
global.player_health = ini_read_real("player_info", "player_health", 100);
global.player_lives = ini_read_real("player_info", "player_lives", 5);
ini_close();
```

At the end of the level add the value of the enemies destroyed in that level to value of `total_enemies_destroyed`?

```
ini_open("game_data.ini");
current=ini_read_real("game_info", "total_enemies_destroyed", 0);
to_save=current+total_destroyed;
ini_write_real( "game_info", "total_enemies_destroyed", to_save );
ini_close();
```

At the end of game, increment the value of `total_games_played`?

```
ini_open("game_data.ini");
total_played=ini_read_real("game_info", "total_games_played", 0);
total_played+=1;
ini_write_real( "game_info", "total_games_played", total_played );
ini_close();
```

At end of game, if the player's **score** is greater than `player_best_score`, update the value?

```
ini_open("game_data.ini");
best_score=ini_read_real("player_info", "player_best_score", 0);
if score>best_score
{
    ini_write_real( "player_info", "player_best_score", score);
}
ini_close();
```



## Basic Projects

- A) Create two rooms, `room_splash` and `room_game`. Create an object for `room_splash` that loads any data from an INI file to two global variables. If no INI file is present, set the starting location the value of each to 100. Make this object clickable to go to `room_game`. Create a movable object for `room_game`, which starts in the position stored in the INI file. Pressing X saves the location to the INI file and restarts the game.

3 Points

- B) Create a counter that keeps track of how many keypresses the player makes in total, and save/load the value of this counter to keep track of presses over multiple games. Use a splash screen with an object to load from the INI file. Use a separate object for detecting space presses.

3 Points

## Advanced Project

- C) Create an object that takes in five people's names, ages, and favorite food. Display this data on screen. When the game is restarted, and an INI file exists, give the option to display the current INI file or enter new data. Create a splash and main room as required. This example should use arrays, which haven't been taught yet; this project is for the more adventurous.

4 Points

## End of Book Game INI files

The game at the end of the book will use INI files to store various game and player stats.

Upon loading, the game will load any saved data (if it exists) or use default values otherwise.

When that player enters the menu screen, all player data will be saved to the INI file.

The INI file will store the variables needed for **health**, **lives**, and cash. It will also store the current unlocked level, total shots taken in game, and how many bullets are left for each weapon.

Think about what would be suitable sections and keys for the INI file to keep the data organized nicely.

## CHAPTER 19



# Effects

Eye candy is quite important in modern graphical games. Your backgrounds, sprites, and buttons need to be of high quality in a finished game. You can also create an assortment of graphical effects using the built-in effects function. The advantage of using effects is they're easy to program and update if required.

In its most basic form you can create an effect using ("ef" designates effect):

```
effect_create_above(ef_cloud, 100, 150, 2, c_green);
```

This would create an effect above the current object. The effect will be a cloud at position x 100 and y 150 in the colour green. The 2 denotes the size of the effect, which is large.

You can also create an effect below the current object using

```
effect_create_below(type, x, y, size, colour);
```

You can try out these effects. Create an object, **obj\_example**, and put this in the **Step Event**:

```
if (mouse_check_button(mb_left))
{
    effect_create_above(ef_cloud,mouse_x, mouse_y, 2, c_green);
}
```

Create a room and place one instance of this of object in it. You can test the effect using the left mouse button.

You can use the following types of effects:

**ef\_cloud, ef\_ellipse, ef\_explosion, ef\_firework, ef\_flare, ef\_rain,  
ef\_ring, ef\_smoke, ef\_smokeup, ef\_snow, ef\_spark, ef\_star.**

You can introduce some randomness into it, for example, change the code to:

```
if (mouse_check_button_pressed(mb_left))
{
    effect_create_above(ef_firework,mouse_x, mouse_y, 2, choose(c_green,c_red,c_white,c_
yellow));
}
```

Now test again.

Try the following code:

```
if (mouse_check_button_pressed(mb_left))
{
    effect_create_above(choose(ef_smoke,ef_ring, ef_explosion, ef_firework),mouse_x,
mouse_y, 2, choose(c_green,c_red,c_white,c_yellow));
}
```

You can also create a mouse trail using effects. Create a new object **obj\_trail**, and put in the following code in the **Step Event**:

```
repeat 2
{
    effect_create_above(ef_star ,mouse_x, mouse_y, 2, choose(c_green,c_red,c_white,c_
yellow));
}
```

Now test the game again. You will see an effect like that shown in Figure 19-1:



**Figure 19-1.** Showing effect trail

## Worksheet – Effects

- A. What would the following code do?

```
effect_create_above(ef_ring,x, y+40, 2, c_white);
```

- B. Which effect type would you use to make snow?

- C. Correct the mistakes in the following code:

```
effect_create_above(ef_smoke, 92, 80, c_blue);  
effect_create_below(ef_flare, 80,56, 2, c_yellow, c_green);
```

- D. If you wanted to create an effect 50 pixels above the current mouse position, what would be the Y value to create the effect at? Y-50
- E. Why would you use effects instead of a sprite animation? Quicker and easier to update.
- F. What does the GML `effect_clear()`; do?
- G. How many effects can be on the screen at once?
- H. Can you create custom colours to use in effects?

## Worksheet – Effects – Answer Sheet

- A. What would the following code do?

```
effect_create_above(ef_ring,x, y+40, 2, c_white);
```

Create a ring 40 pixels below object in white

- B. Which effect type would you use to make snow? `ef_snow`  
 C. Correct the mistakes in the following code:

```
effect_create_above(ef_smoke, 92, 80, c_blue);
```

Needs a size defined, like: `effect_create_above(ef_smoke, 92, 80, 2, c_blue);`

```
effect_create_below(ef_flare, 80,56, 2, c_yellow, c_green);
effect_create_below(ef_flare, 80,56, 2, choose(c_yellow, c_green));
```

- D. If you wanted to create an effect 50 pixels above the current mouse position, what would be the Y value to create the effect at?

`mouse_y-50`

- E. Why would you use effects instead of a sprite animation?  
 Easy to use, takes up less space.
- F. What does the GML `effect_clear()`; do? **Clears / removes any active effects**
- G. How many effects can be on the screen at once?  
 No real limit, though too many will cause the game to slow down.
- H. Can you create custom colours to use in effects?  
 Yes, just as you'd create any other custom colour.

## Basic Projects

- A) Allow the user to change the weather by pressing W. Change between a snow and rain effect.  
2 Points
- B) Create a menu button that creates firework effects of different colours in the middle when pressed with the mouse button.  
2 Points

## Advanced Projects

- C) Create a line of effects, 20 pixels apart, which start at the top of the screen and fall down to the bottom, then start from the top again.  
3 Points
- D) Create an effect that spreads out from a location when the mouse is clicked. Make the effect move out every 10 degrees from the starting point. Destroy any effects after 3 seconds.  
3 Points

## End of Book Game Effects

The game will use a combination of effects to visually show weapons exploding and collisions.

Some effects will happen straight off of a **Collision Event**, while some will utilize an alarm system to limit the time between effects.



## CHAPTER 20



# Loops

A common need in programming is to repeat the same code multiple times. This functionality is available through using loops. There are four main loop functions in the GameMaker Language: **do**, **while**, **for**, and **repeat**. Each of these has its own strengths and weaknesses.

A loop can result in many similar actions being executed in a very short notice.

Note: It is important to make certain that loops run the correct number of times and then stop. It is possible through a coding mistake to create an infinite loop that runs forever. This will cause the game to stop working. Care needs to be taken to avoid this from happening.

### **repeat Loop:**

**repeat** is a very simple control structure, which will repeat an assigned action for a specified number of times. The following code creates five enemies in a single step, all at random positions in the room.

```
repeat(5) //Repeat the following code block 5 times
{
    instance_create(irandom(room_width),
        irandom(room_height), obj_enemy);
}
```

### **while Loop**

The **while** loop will repeat an action as long as the expression assigned to it is true or until you call a **break**. For example the following checks for an empty space randomly, and when it finds a free space the loop stops.

```
while (!place_free(x, y))
{
    x = random(room_width);
    y = random(room_height);
}
```

### **for Loop:**

The **for** loop will execute different actions, and after each one it will check if its expression is true. If the expression is not true, the loop will end, and none of the following functions will be executed. Like before, we'll use it to create a 50-point array in a matter of milliseconds. In almost all cases, you'd want to use `var i` here, as the loop counter generally has no need to be used outside of the block scope.

```
for(var i=0; i<50; i++)
{
    array[i]=i;
}
```

The first statement (`i=0`) initializes the loop and sets a starting value. After initialization, the loop will check if `i` is smaller than 50, and if it is, it will increase the value of `i` using the `i+=1;` statement; this can also be written as `i++;`. After each step, the array's length will increase by 1 and add `i` to that array's position.

---

■ **Note** Alternatively you can reduce a value by 1 using `i--;`.

---

To iterate `n` times with a for loop:

- If you start with `i=0`, check `i<n`.
- If you start with `i=1`, check `i<=n`.

**do Loop:**

This will repeat until an expression is true. For example, the following will repeat until it finds an empty position:

```
do
{
    x = random(room_width);
    y = random(room_height);
}
until (place_free(x, y));
```

Not too hard to understand, right? It can be used in many ways and it's one of the most practical functions in the GameMaker Language, so don't forget about it.

Notes: Some of these functions can loop forever, so be careful how you use them.

The **do** loop will always execute at least one time, but in **for** or **while**, it can be skipped.

## Worksheet – Loops

**A.** True or False

Loops are used to perform many tasks in a very short time. T / F

It's impossible for them to loop forever. T / F

The DO function will loop until its expression is true. T / F

**B.** How do we use the DO, FOR, and WHILE loops? Write an example for each:

**C.** Write a For loop that initializes a 100-index array, with values 1 through to 100.

**D.** You have a 2D array 5 by 5 in size. Write the appropriate code to draw all data onscreen:

## Worksheet – Loops – Answer Sheet

**A.** True or False

-Loops are used to perform many tasks in a very short time. **T**

-It's impossible for them to loop forever. **F**

-The DO function will loop until its expression is true. **T**

**B.** How do we use the DO, FOR, and WHILE loops? Write an example for each:

Any suitable answer is OK here.

**C.** Write a WHILE loop that initializes a 100-index array, with values 1 through to 100.

```
for(var i=0; i<100; i++)
{
    array[i]=i+1;
}
```

**D.** You have a 2D array 5 by 5 in size. Write the appropriate code to draw all data onscreen. One suitable approach is this:

```
var loop1,loop2;
for (loop1 = 0; loop1 < 5; loop1 += 1)
{
    for (loop2 = 0; loop2 < 5; loop2 += 1)
        draw_text(loop1*100,loop2*50,array[loop1,loop2]);
}
```

## Basic Projects

- A) Place an object in a room at a random position. Create another object that finds a random location within 100 pixels of the first object. Use a while loop for this.  
2 Points
- B) Add 100 random numbers between 1 and 1000 to a `ds_list`, then sort them into order, highest value first. Display onscreen in 4 columns of 25. Use for loops for this.  
2 Points

## Advanced Projects

- C) Create four random points in the room. Get an object to visit each point in order. Display a message when all the have been visited.  
3 Points
- D) Store the names of students in your class in an appropriate way. Display names in alphabetical in order, one at a time on the screen for 5 seconds, with a gap with no name for 2 seconds.  
3 Points

## End of Book Game Loops

The game includes two examples of a loop.

A *repeat* loop used for creating multiple explosions for the nuke when it collides with an asteroid or enemy ship.

And a *for* loop for displaying the total number of shots for each weapon and the total of all weapons combined.

## CHAPTER 21



# Arrays

Arrays are a useful way of storing data in an organized format. They allow similar information to be stored together. This allows for easy access, manipulation, and displaying of data.

You can store real numbers, integers, strings; and the indexes of sounds, sprites, and objects in a single array.

Some use applications for arrays include the following:

- Storing weapon information for multiweapon games
- Keeping lists of data
- Storing data in order created
- Storing facts and information

There are both one-dimensional arrays and two-dimensional arrays in GameMaker.

An example of setting a 1D array:

```
name[0]= "Bob";  
name[1]= "Claire";  
name[2]= "Steve";  
name[3]= "Nigel";  
name[4]= "Sue";
```

A visualization of this would look something like this:

---

Location

0	Bob
1	Claire
2	Steve
3	Nigel
4	Sue

---

The following would draw the string stored in the array "name" at index 2, "Steve," at position 100, 100.

```
draw_text(100, 100, name[2]);
```

An example of a 2D array, storing name, age, and country of residence.

```
//names
info[0, 0]= "Bob";
info[0, 1]= "Claire";
info[0, 2]= "Steve";
info[0, 3]= "Nigel";
info[0, 4]= "Sue";
//ages
info[1, 0]=27;
info[1, 1]=19;
info[1, 2]=52;
info[1, 3]=40;
info[1, 4]=102;
//country
info[2, 0]= "America";
info[2, 1]= "Spain";
info[2, 2]= "Brazil";
info[2, 3]= "Canada";
info[2, 4]= "France";
```

A visualization of this in table form would look like this:

---

	0	1	2
0	Bob	27	America
1	Claire	19	Spain
2	Steve	52	Brazil
3	Nigel	40	Canada
4	Sue	102	France

---

So the value at cell **[1, 3]** would be **40**.

You can use and change the values of an array as you would with any variable.

Arrays come into their own when processing data, especially with 2D arrays, as each array entry can be different lengths. Put the code shown previously into the **Create Event** of an object.

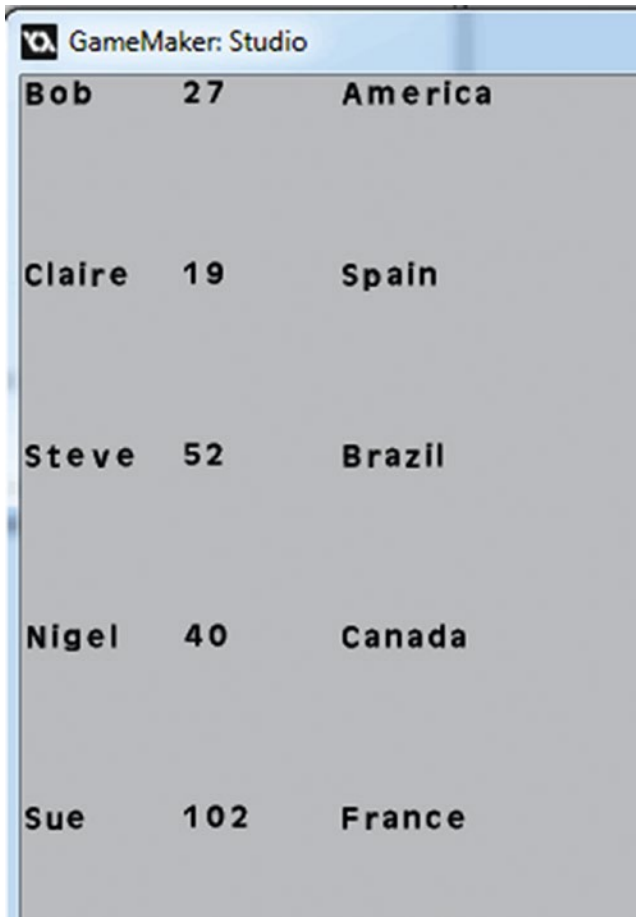
For loops are great ways to sequentially process data. The example below will repeat *i* three times and each time repeat *j* five times each time for *i*; this is known as a nested loop. It will then draw the value of the array cell at that position.

In a **Draw Event** put:

```
for (i=0; i < 3; i++)
{
  for (j= 0; j < 5; j++)
  {
    draw_text(i*70, j*80, info[i, j]);
  }
}
```

This code will draw the values of information onscreen in a table format, as shown in Figure 21-1.





*Figure 21-1. Showing output of data*

An example GMZ is available in the resources at: **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 21 ► examplea**

You can add to an array variable just like a regular variable. For example:

```
info[1, 3]+=1;
```

Which would increase Nigel's age by 1.

You can compare array values:

```
if (info[1 ,0]>info[1, 1])
{
    text_to_show=(info[0, 0]+ " is older than "+info[0, 1]);
}
```

Which sets text\_to\_show to **Bob is older than Claire**.

**That is all for this example.**

An application of this in GameMaker: Studio as an example; arrays are very useful for holding data for multiple weapons, as shown:

Weapon Name	Strength	Cost	Current Ammo	Sound Effect	Image	Bullet Object
Gun	1	1	200	snd_gun	spr_gun	obj_bullet_gun
Machine Gun	5	10	400	snd_mach_gun	spr_mach_gun	obj_bullet_mach_gun
Rocket Grenade	250	300	8	snd_rocket	spr_rocket	obj_bullet_rocket
Nuke	1000	5000	2	snd_nuke	spr_nuke	obj_bullet_nuke

You can create a data array for the previous code using:

```
//declare other variables needed
global.cash=100000;
global.selected_weapon=0;
//declare array
//weapon name
weapon_info[0,0]= "Gun";
weapon_info[0,1]= "Machine Gun";
weapon_info[0,2]= "Rocket Grenade";
weapon_info[0,3]= "Nuke";

//weapon strength
weapon_info[1,0]=1;
weapon_info[1,1]=5;
weapon_info[1,2]=250;
weapon_info[1,3]=1000;

//weapon cost
weapon_info[2,0]=1;
weapon_info[2,1]=10;
weapon_info[2,2]=300;
weapon_info[2,3]=5000;

//weapon ammo
weapon_info[3,0]=200;
weapon_info[3,1]=400;
weapon_info[3,2]=8;
weapon_info[3,3]=2;
```

```

//weapon sound effect
weapon_info[4,0]=snd_gun;
weapon_info[4,1]=snd_mach_gun;
weapon_info[4,2]=snd_rocket;
weapon_info[4,3]=snd_nuke;

//weapon sprite
weapon_info[5,0]=spr_gun;
weapon_info[5,1]=spr_mach_gun;
weapon_info[5,2]=spr_rocket;
weapon_info[5,3]=spr_nuke;

//weapon bullet object
weapon_info[6,0]=obj_bullet_gun;
weapon_info[6,1]=obj_bullet_mach_gun;
weapon_info[6,2]=obj_bullet_rocket;
weapon_info[6,3]=obj_bullet_nuke;

```

Create an object, **obj\_example**, and place the above code in a **Create Event**.

The resources needed for this example are in the downloadable resources.

You can set up an easy weapon select system using the following in a **Step Event**.

```

if (keyboard_check_pressed (ord('0')))
{
    global.selected_weapon=0;
}
if (keyboard_check_pressed(ord('1')))
{
    global.selected_weapon=1;
}
if (keyboard_check_pressed(ord('2')))
{
    global.selected_weapon=2;
}
if (keyboard_check_pressed(ord('3')))
{
    global.selected_weapon=3;
}

```

Then you can do something like the following to fire the gun, also in the **Step Event**:

```
if (mouse_check_button_pressed(mb_left)) && weapon_info[3, global.selected_weapon]>=1
{
    audio_play_sound(weapon_info[4, global.selected_weapon], 10, false); //play firing sound
    weapon_info[3, global.selected_weapon]-=1; //reduce ammo
    instance_create(mouse_x, mouse_y, weapon_info[6, global.selected_weapon=0]); //create
bullet
}
x=mouse_x;
y=mouse_y;
```

In a **Draw Event** put:

```
draw_sprite(weapon_info[5,global.selected_weapon],0,x,y);
//draw info
draw_text(10,20, "Strength: " +string(weapon_info[1,global.selected_weapon]));
draw_text(10,40, "Cost: "+ string(weapon_info[2,global.selected_weapon]));
draw_text(10,60, "Current Ammo: " +string(weapon_info[3,global.selected_weapon]));
```

If you wanted to take this further, then you could add sprites to the bullets and use this as a basis for a game. For instance if you had a constant, `w_strength` set to a value of 1, you could replace this code:

```
draw_text(10,20, "Strength: " +string(weapon_info[1,global.selected_weapon]));
```

with:

```
draw_text(10,20, "Strength: " +string(weapon_info[w_strength,global.selected_weapon]));
```

Which is easier to understand.

Using a macros constant this way keeps the code VERY tidy and makes changing the value later if required easy, as it only need changed in one place.

An example is available at **Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 21 ► exampleb**

## Worksheet – Array

Put the following data into an array, called ***car\_array***:

Car Type	Miles	Seats	Colour	Image	Cost
Rover	52000	5	Red	spr_rover	2800
Ford	28000	5	Yellow	spr_ford	4200
Bugatti	800	2	Black	spr_bugatti	156000
Nissan	14500	4	Blue	spr_nissan	1400
Kia	0	6	Silver	spr_kia	18995
Mini	126500	4	Green	spr_mini	1100
Limousine	3500	8	White	spr_limo	36000

Create code to:

- A. Draw all of the values and legends on screen in a grid format, including the sprite image.
- B. Reduce all costs by 10%.
- C. Add an additional element for damage, and set a value of 1 (low damage) to 10 for each car.

## Worksheet – Array – Answer Sheet

Put the following data into an array, called **car\_array**:

The following will populate the array with initial data:

```
//Car Names
car_array[0,0]= "Rover";
car_array[0,1]= "Ford";
car_array[0,2]= "Bugatti";
car_array[0,3]= "Nissan";
car_array[0,4]= "Kia";
car_array[0,5]= "Mini";
car_array[0,6]= "Limousine";
//Car Mileage
car_array[1,0]=52000;
car_array[1,1]=28000;
car_array[1,2]=800;
car_array[1,3]=14500;
car_array[1,4]=0;
car_array[1,5]=126500;
car_array[1,6]=3500;
//Car Seats
car_array[2,0]=5;
car_array[2,1]=5;
car_array[2,2]=2;
car_array[2,3]=4;
car_array[2,4]=6;
car_array[2,5]=4;
car_array[2,6]=8;
//Car Colour
car_array[3,0]= "Red";
car_array[3,1]= "Yellow";
car_array[3,2]= "Black";
car_array[3,3]= "Blue";
car_array[3,4]= "Silver";
car_array[3,5]= "Green";
car_array[3,6]= "White";
//Car Image
car_array[4,0]=spr_rover
car_array[4,1]=spr_ford;
car_array[4,2]=spr_bugatti;
car_array[4,3]=spr_nissan;
car_array[4,4]=spr_kia;
car_array[4,5]=spr_mini;
car_array[4,6]=spr_limo;
//Car Cost
car_array[5,0]=2800;
car_array[5,1]=4200;
car_array[5,2]=156000;
car_array[5,3]=1400;
car_array[5,4]=18995;
```

```
car_array[5,5]=1100;
car_array[5,6]=36000;
```

**Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 21**

Create code to:

- A. This is one approach, there are several other valid ways to program this:

```
var loop;
for (loop = 0; loop < 7; loop += 1)
{
    draw_text(20,70+(loop*50),car_array[0,loop]); //Draw Car Type
    draw_text(120,70+(loop*50),car_array[1,loop]); //Draw Miles
    draw_text(220,70+(loop*50),car_array[2,loop]); //Draw Seats
    draw_text(320,70+(loop*50),car_array[3,loop]); //Draw Car Type
    draw_sprite(car_array[4,loop],0,420,70+(loop*50)); //Draw Sprite
    draw_text(520,70+(loop*50),car_array[5,loop]); //Draw Price
}

draw_text(20,20, "Car Type: "); //Draw Legend
draw_text(120,20, "Miles: "); //Draw Legend
draw_text(220,20, "Seats: "); //Draw Legend
draw_text(320,20, "Colour: "); //Draw Legend
draw_text(420,20, "Image: "); //Draw Legend
draw_text(520,20, "Price: "); //Draw Legend
```

- B. Reduce all costs by 10%. The following code will do this, when executed once:

```
//reduce costs by 10%
var loop;
for (loop = 0; loop < 7; loop += 1)
{
    car_array[5,loop]*=0.9;
}

```

- C. Add an additional element for damage, and set a random value of 1 (low damage) to 10 for each car.

```
//add damage
var loop;
for (loop = 0; loop < 7; loop += 1)
{
    car_array[6,loop]=irandom_range(1,10);
}

```

And update the **Draw Event** accordingly, by adding the following within the loop of the **Draw Event**:

```
draw_text(620,70+(loop*50),car_array[6,loop]); //Draw Damage
```

And the following below:

```
draw_text(620,20, "Damage: "); //Draw Legend
```

An example for this worksheet projects is in the downloadable assets folder:

**Project Assets & GMZ Files ► Assets Used In Main Chapters ► Chapter 21**

## Basic Projects

- A) Create a 2D array with data relating to four students in your group. Include variables name, age, height, eye colour, and favourite food. Display this data onscreen.  
2 Points
- B) Make a 1D array with the values of 10 types of food. Display one at random each time the spacebar is pressed.  
2 Points

## Advanced Projects

- C) Create and populate a two-dimensional array with the 12 times table. Draw the contents on the screen.  
3 Points
- D) Create an array to store the starting positions in a game of chess, use letters to represent each piece, that is, K for King, Q for Queen, N for knight, B for bishop, and C for castle; all other squares to have a value of 0. Use UPPER CASE for black and lowercase for white. Represent this board on the screen. Try to use one or more loops to populate the array. Draw a chessboard with the correct value in the middle of each square.  
3 Points



## End of Book Game Arrays

Arrays will be used in the game to store data for the weapons.

An example section for weapon with the index 1 will look like this:

```
//name of weapon
global.weapon_info[0,1]= "Gun";
//strength of weapon
global.weapon_info[2,1]=1;
//number of bullets
global.weapon_info[3,1]=200;
//sprite of ship when weapon selected
global.weapon_info[4,1]=spr_ship_gun;
//weapon firing sound
global.weapon_info[5,1]=snd_gun;
//weapon explosion sound
global.weapon_info[6,1]=snd_explosion_gun;
```

In total there will be four types of weapons. By placing the data in an array we can quickly and easily access any weapon data.

## CHAPTER 22



# ds\_lists

DS Lists (Data Structure) are effectively one-dimensional arrays that allow you to add data sequentially. However, they are much more flexible than a one-dimensional array, and you can perform lots of cool functions with them. They are great for the following:

- Sorting data alphabetically
- An inventory system
- Shuffling items
- Card games
- Music management
- Message management
- Sort the contents
- Delete contents, which will shift the remaining contents
- Add content at the start, anywhere in the middle, or at the end, which again can shift contents
- Find where something occurs in the list

---

■ **Note** *ds\_lists* start with an index of 0, so if you wanted to insert at position 5, the actual index would be 4.

---

**ds\_lists** are awesome for making an inventory system. You can easily add items and remove them. So in an RPG type game you may pick up a key, then remove it when you collide with a door.

To start you need to declare a **ds\_list**, for example:

```
example_list = ds_list_create();
```

Then add something to the list:

```
ds_list_add(example_list, "cheese");  
ds_list_add(example_list, "bacon");  
ds_list_add(example_list, "mushroom");  
ds_list_add(example_list, "ham");  
ds_list_add(example_list, "tomato");
```

■ **Note** It's also possible to add many values within one line of code. For example:

```
ds_list_add(example_list, "cheese", "bacon", "mushroom", "ham", "tomato");
```

This would give the following result:

Index	value
0	"cheese"
1	"bacon"
2	"mushroom"
3	"ham"
4	"tomato"

You can sort the list:

```
ds_list_sort(example_list, true);
```

Where **true** will be ascending or **false** for descending. Strings are sorted alphabetically, lowest to highest for values.

This would then look like:

Index	value
0	"bacon"
1	"cheese"
2	"ham"
3	"mushroom"
4	"tomato"

You can remove a value:

```
ds_list_delete(example_list, 2);
```

Would remove the value at position 3 (index 2).

The list will then look like:

Index	value
0	"bacon"
1	"cheese"
2	"mushroom"
3	"tomato"

You can insert a new value:

```
ds_list_insert(example_list, 1, "egg");
```

Would insert "egg" at position 2 (index 1).  
The list will then look like:

Index	value
0	"bacon"
1	"egg"
2	"cheese"
3	"mushroom"
4	"tomato"

After adding an element you may want to sort your list again.  
Other things you may want to do, such as finding where in the list something appears:

```
position = ds_list_find_index(example_list, "cheese");
```

You can return a value to a variable, for example, the following, which would set word to "mushroom":

```
word=ds_list_find_value( example_list, 3);
```

You may need to find the size of a list, for example:

```
list_size=ds_list_size(example_list);
```

---

■ **Note** The first position is at 0. So if the size is 5, its indexes will be 0,1,2,3,4

---

Which is great to use before you try and find a value at a position.  
You can shuffle (randomize) the values with:

```
ds_list_shuffle(example_list);
```

You can also use the accessor | that will allow you treat the **ds\_list** as an array, for example, the following code would draw the value in index 3 at 100,100 on the screen:

```
draw_text(100,100,example_list[| 3]);
```

You can look up how to use:

```
ds_list_clear()
ds_list_empty()
ds_list_replace()
ds_list_copy()
ds_list_read()
ds_list_write()
```

When you have finished using a **ds\_list** it's essential to destroy it. This helps prevent memory leaks.  
For example:

```
ds_list_destroy(example_list);
```

## Worksheet – ds\_lists

Write the GML code to perform the following:

- A. Add the following values to a **ds\_list**, list\_example.  
232, 6, 34, 989, 42, 56
- B. What does the list look like?

---

0

1

2

3

4

5

---

- C. Sort the values in ascending order; what does it look like now?
- D. Remove value at index 3...what does it look like now?
- E. Add the following values: 12, 388, 191, 17
- F. Sort descending; what does it look like?
- G. What code would you use to find out how many elements are in the **ds\_list**?
- H. What code would you use to remove one index at random?
- I. What code would you use to shuffle the list and find the value of index **0**?

## Worksheet – ds\_lists – Answer Sheet

Write the GML code to perform the following:

- A. Add the following values to a **ds\_list** called list\_example  
232, 6, 34, 989, 42, 56
- B. What does the list look like?

---

0	232
1	6
2	34
3	989
4	42
5	56

---

- C. Sort the values in ascending order; what does it look like now? 6,34,42,56,232,989
- D. Remove value at index 3...what does it look like now? 6,34,42,232,989
- E. Add the following values: 12, 388, 191, 17: Will look like:

6,34,42,232,989,12,388,191,17

- F. Sort descending; what does it look like? 989,388,232,191,42,34,17,12,6
- G. What code would you use to find out how many elements are in the **ds\_list**?

```
ds_list_size(list_example);
```

- H. What code would you use to remove one index at random?

```
ds_list_delete(list_example, irandom(ds_list_size(list_example) - 1));
```

- I. What code would you use to shuffle the list and find the value of index 0?

```
ds_list_shuffle(list_example);
example=ds_list_find_value(list_example, 0);
```

## Basic Projects

- A) Create an inventory system for five objects. If you add an additional item, add it to the end of the list, then remove the top item. Draw this onscreen.  
2 Points
- B) Create a **ds\_list** with five fruits. Player enters a fruit; if it matches a value in the **ds\_list**, remove it, and tell player they made a correct guess. Player wins when all five fruits are guessed.  
2 Points
- C) Create a list with the names of students in the class. Sort them in ascending order and draw on the screen.  
2 Points

## Advanced Project

- D) Add the names of all playing cards to a list. Shuffle them. Create four new **ds\_lists** to represent player's hands. Deal and remove the top card from the main list and deal to each player until each has five cards. Draw the values of each player's hand on the screen. Represent value and suit like: AS, 9H, 2D etc.  
4 Points

## End of Book Game ds\_list

This game will use a DS list to store and process messages. We can add a message to the list anytime we want to, then use a control object to process them in the order they arrived.

We can take the message from position 0 and assign it to another variable. We can then delete the message at position 0 and allow the next one to take its place (if there is another message).

Taking this new variables string we can do what we want with it. In this instance we'll display it as a message on the screen in a big font and give the player time to read it. We'll then remove it, wait a short amount, and then display the next message if there is one.



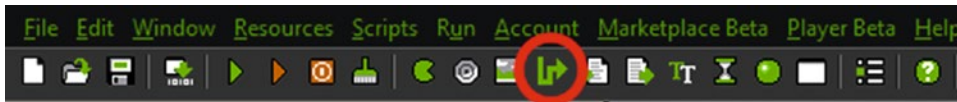
## CHAPTER 23



# Paths

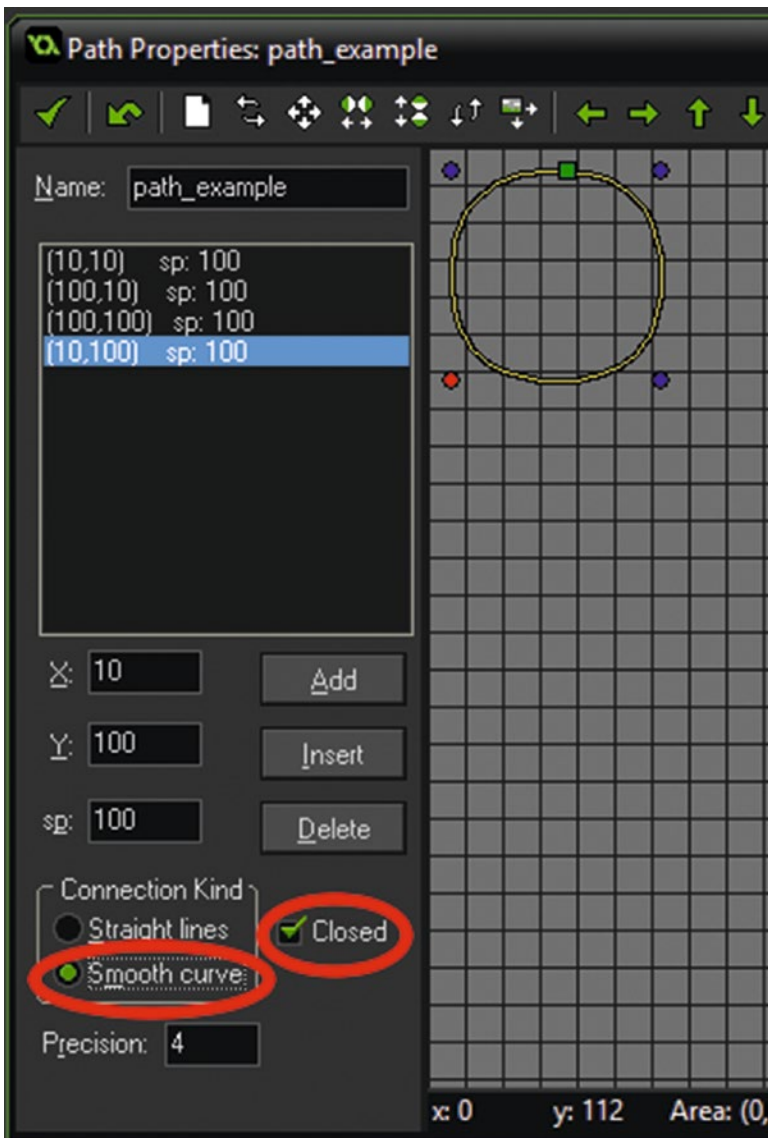
Sometimes you want to have repeated object motion similar to how code is repeated with scripts. Paths are a useful way to make objects move in a set way. You can create paths using the built-in path editor or using GML. This section covers both.

You can create a new path by clicking the **Create a path** button at the top as shown in Figure 23-1.



*Figure 23-1. Creating a new path*

You can then create the path, and add points by clicking. Figure 23-2 shows a path with four points added, with **Closed** and **Smooth curve** checked. sp means the speed.



**Figure 23-2.** Showing an example path closed and smooth curve

If you were to create the same path using GML, you would use the code below:

```
path_example=path_add();
path_add_point(path_example,10,10,5);
path_add_point(path_example,100,10,5);
path_add_point(path_example,100,100,5);
path_add_point(path_example,10,100,5);
```

This creates the path and stores it in instance scope, not global scope as the path editor would have. This would create a memory leak if the instance is destroyed later without calling `path_delete()`.

The 5 at the end relates to the objects speed, though this is usually superseded when starting the path. The above example could be used to make a sentry move around and protect a building.

You can then set whether the path is straight or curved: use **0** for straight, **1** for curved. So in this example we'll use:

```
path_set_kind(path_example, 1);
```

Then set that the path is closed with:

```
path_set_closed(path_example, true);
```

Then start the path with:

```
path_start(path_example, 50, path_action_restart, false);
```

`path_example` is the name of the path you created in the editor.

50 relates to the speed.

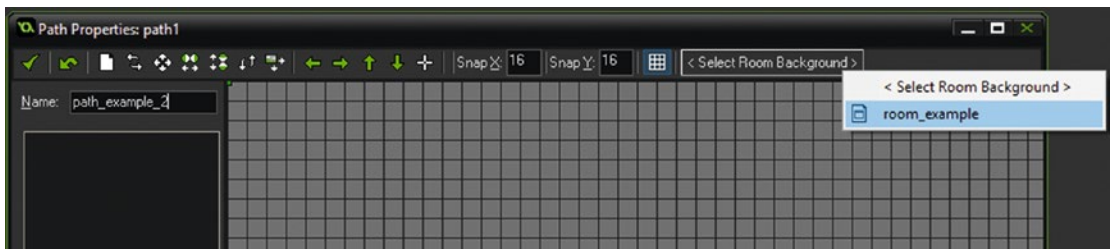
`path_action_restart` tells what should happen when the last path location (end) is reached.

The last part can be **true** or **false**. **false** places the path at the absolute position within the room (i.e., where you defined it in the path editor) and **true** positions it relative to the instance (i.e., the start position will be at the current instance x/y position).

There are several end path actions. They are:

Action name	Path Action Value	What It Does
<code>path_action_stop</code>	0	<b>Stops at the end of the path</b>
<code>path_action_restart</code>	1	<b>Restarts path from first position</b>
<code>path_action_continue</code>	2	<b>Continues the current path</b>
<code>path_action_reverse</code>	3	<b>Reverses along current path</b>

A neat feature that should be pointed out is that using the inbuilt path editor, you can show a room's object setup, enabling you to create a path avoiding objects you've already placed. To use this feature, click as shown in Figure 23-3.



**Figure 23-3.** Showing a room in the path editor

Some other useful codes for dealing with paths include the following:

---

■ **Note** These should not be used for paths currently being followed.

---

You can insert a new point in a path using the following, which would add a new point to the path **path\_example** at position 3 with the X and Y at with a speed of 5:

```
path_insert_point(path_example,3, 50,50, 5);
```

You can change a point's location, or speed using the following, which would change the path's point at position 4 to an X and Y of 25 with a speed of 5:

```
path_change_point(path_example, 4, 25, 25, 5);
```

You can get the X or Y point of a path's position using, for example, the following, which give the point of X location at position 2:

```
xpoint=path_get_point_x(path_example, 2);
```

You can set how precise a curved path is using **1**(low) to **8**(high):

```
path_set_precision(path_example, 4);
```

## Worksheet – Paths

Correct mistakes in the following:

```
path_get_point_x(3, path_example);  
path_point_insert(path_example,2, 100,45, 15);  
path_start(path_example, 5, path_action_restart, 2);  
path_example=path_add(new);
```

True or False:

1. Paths can be straight lines. T / F
2. At the end of the path it will automatically start from the beginning again. T / F
3. You can only add 5 new points to a path. T / F
4. You can reverse the direction of a path at any time. T / F
5. You can calculate the X and Y location of the object at any time. T / F
6. You can calculate the X and Y location of the next point in the path. T / F
7. You can delete or change any point in a path. T / F

## Worksheet – Paths – Answer Sheet

Correct mistakes in the following:

```
path_get_point_x(3, path_example);  
path_get_point_x(path_example,3);  
path_point_insert(path_example,2, 100,45, 15); path_insert_point(path_example,2,  
100,45, 15);  
path_start(path_example, 5, path_action_restart , 2);  
path_start(path_example, 5, path_action_restart, true);  
path_example=path_add(new);  
path_example=path_add();
```

True or False:

1. Paths can be straight lines. **T**
2. At the end of the path it will automatically start from the beginning again. **T**
3. You can only add 5 new points to a path. **F**
4. You can reverse the direction of a path at any time. **T**
5. You can calculate the X and Y location of the object at any time. **T**
6. You can calculate the X and Y location of the next point in the path. **T**
7. You can delete or change any Y point in a path. **T**

## Basic Projects

- A) Create a path for an object, and make it move in a circle.  
1 Point
- B) When the player left-clicks the mouse add the location as a new point on the path. Use project 23 A as a basis.  
2 Points

## Advanced Projects

- C) Make an object point in the direction of movement when following a path. Use project 23 B as a basis.  
3 Points
- D) Create a random path of 10 points, and save the points to an INI file.  
3 Points

## End of Book Game Paths

The game includes two examples of paths.

Paths are used for moving the bonus coin object and the cloud object.



## CHAPTER 24



# Scripts

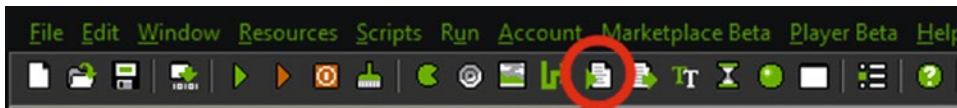
Reusable code makes it easy to update program objects at the same time. Additionally, it makes it easier to organize and understand how code works. GML scripts are useful in processing data, especially if you will be doing the same calculation again and again. This can include sending data to the script and then returning a value if required. If you are using the same code twice or more anywhere in your program, then you should consider using a script. This allows you to make just one change to update your code. Imagine a game that had over 100 enemy monsters with their own code; changing the code for each would take many hours, and be prone to errors. Using a script you could do this in a few minutes. It also allows for nice and tidy code.

Also, scripts allow the possibility of sharing code between multiple projects.

Some examples for scripts:

- Doing a math calculation and returning the answer, even if only used once; it means that your code is easier to read through and understand.
- Setting a drawing or font type, formatting, and colour – makes code easier to read.
- Playing sound effects and voices.
- Sending through an object and returning the closest instance.
- Drawing text that's used multiple times.
- Recording bullet hits against multiple different objects.
- Adding things to a DS list.
- Any other GML that's used more than once.

You can create a new script by clicking the **Create a script** button at the top of the window in the location shown in Figure 24-1.



**Figure 24-1.** Creating a new script

Name the script *scr\_example*. Put in the following code:

```
///scr_example(value1,value2,value3)
//this script adds 3 numbers
var total=argument0+argument1+argument2;
return total;
```

Save this by pressing the green tick.

Create an object **obj\_example**.

In its **Create Event** put (so we don't try and draw a nonexistent variable):

```
answer=scr_example(12,18,7);
```

In the **Draw Event** put:

```
draw_text(100,100,answer);
```

This will send the values **12**, **18**, and **7** to the script. The script will perform its GML and then return the value of total.

You can return strings, integers, real numbers, Boolean (true or false) and indexes of sounds, objects, rooms etc.

For example you can send through some objects:

Create a script and name it *scr\_pos*:

```
/// scr_pos(obj1, obj2)
//compares heights on y axis
if (instance_exists(argument0) && instance_exists(argument1))
{
    if argument0.y < argument1.y
    {
        return true;
    }
}
return false;
```

This will return **true** if **obj\_1** is higher up the room than **obj\_2**, by checking the Y location of each, or false otherwise.

Note that when you call return you are exiting the script, and no further code will be run from it.

You do not have to return a value, as shown in the following example. Another example, this script will use views to keep two objects in view, and set the border size. It takes the X and Y locations of two objects and adjusts the view so that both can be seen at the same time.

*scr\_view\_control(obj1,obj2,border)*

```
var o1,o2,x1,x2,y1,y2,vw,vh,vb,vscale;
o1=argument0; x1=o1.x; y1=o1.y;
o2=argument1; x2=o2.x; y2=o2.y;
vb=argument2; vw=view_wport[0]; vh=view_hport[0];
vscale=max(1,abs(x2-x1)/(vw-(vb*2)),abs(y2-y1)/(vh-(vb*2)));
view_wview[0]=vscale*vw;
view_hview[0]=vscale*vh;
view_xview[0]=(x1+x2-view_wview)/2;
view_yview[0]=(y1+y2-view_hview)/2;
```

For example `scr_view_control(obj_player1, obj_player2, 150);`

You can send through positions of an instance or of the mouse, for example, without returning. The following would draw a laser between two endpoints.

***scr\_laser(start\_x,start\_y,end\_x,end\_y)***

```
draw_set_color(make_color_rgb(irandom(255),irandom(255),irandom(255)));
draw_line_width(argument0, argument1, argument2, argument3, 5);
draw_set_color(c_lime);
draw_line(argument0+1,argument1+1,argument2,argument3);
draw_line(argument0+1,argument1-1,argument2,argument3);
draw_line(argument0-1,argument1+1,argument2,argument3);
draw_line(argument0-1,argument1-1,argument2,argument3);
draw_line(argument0,argument1,argument2,argument3);
effect_create_above(ef_spark, argument2, argument3, 1,
choose(c_red,c_orange));
```

An example of using this script, which would draw a laser effect from the object to the mouse position:

```
scr_laser(x, y, mouse_x, mouse_y);
```

Arguments can be accessed in two ways, for example:

```
name=argument0;
```

Or

```
name=argument[0];
```

---

■ **Note** You cannot mix the two types above in a script.

---

## Worksheet – Scripts

Describe what each of the following scripts would do:

1.

```
number=floor((argument0+argument1+argument2)/3);  
return number;
```

2.

```
if argument0 mod 2==0  
{  
    return true;  
}  
else  
{  
    return false;  
}
```

3.

```
if position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left)  
{  
    if global.total<global.max_total  
    {  
        global.total+=1;  
    }  
}
```

## Worksheet – Scripts – Answer Sheet

Describe what each of the following scripts would do:

**1. Will add three numbers and return the average**

```
number=floor((argument0+argument1+argument2)/3);  
return number;
```

**2. Will return true if number is even or false if it is odd**

```
if argument0 mod 2==0  
{  
    return true;  
}  
else  
{  
    return false;  
}
```

**3. If mouse is left, button released over object will add 1 to global.total if it is less than global.max\_total**

```
if position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left)  
{  
    if global.total<global.max_total  
    {  
        global.total+=1;  
    }  
}
```

## Basic Projects

Create a script to do each of following, and display any result onscreen visually, as required, remembering to set up any text drawing.

- A) Find the average of five numerical values and round to the nearest whole number.  
1 Point
- B) Work out if the player is within 200 pixels of an enemy object, and return true or false.  
1 Point
- C) Draw given text, in black with a red shadow, at given position.  
1 Point
- D) Create three different fonts. Create a script that allows you to quickly draw text, using font, alignment, colour, and position.  
1 Points

## Advanced Projects

- E) Create a script that finds an average point between two objects, and draws a star effect at that position.  
2 Points
- F) Create a script that takes and calculates an angle between two objects and draws that direction as text as the angle were on a compass needle, that is, North or South West. The direction is that from the first object to the second. North is up.  
4 points

## End of Book Game Scripts

The game makes use of a lot of scripts. They are used when code is used in more than one place; this makes code easier to understand and for performing some math calculations.

The scripts used are:

- ***scr\_angle\_rotate*** - for rotating player and enemy ships
- ***scr\_bullet\_hit*** - reducing asteroid or enemy health when hit by a bullet
- ***scr\_buy*** - buys a bullet in shop if player has enough cash
- ***scr\_create\_bullet*** - creating bullet(s) if player has enough
- ***scr\_cycle*** - performing a math calculation
- ***scr\_draw\_shop*** - draws data for a weapon in the shop
- ***scr\_fading*** - used for fading a bullet if alarm is reached
- ***scr\_locked\_or\_unlocked*** - works out if level is unlocked and displays appropriate sprite
- ***scr\_msg*** - a message to a DS list
- ***scr\_play\_effect*** - plays a given sound effect
- ***scr\_set\_menu\_text*** - sets font drawing properties for menu
- ***scr\_shop\_set\_text*** - sets font drawing properties for shop
- ***scr\_target*** - works out nearest enemy or asteroid for missile weapon, and for enemy to find player
- ***scr\_voice*** - plays a voice sound effect

## CHAPTER 25

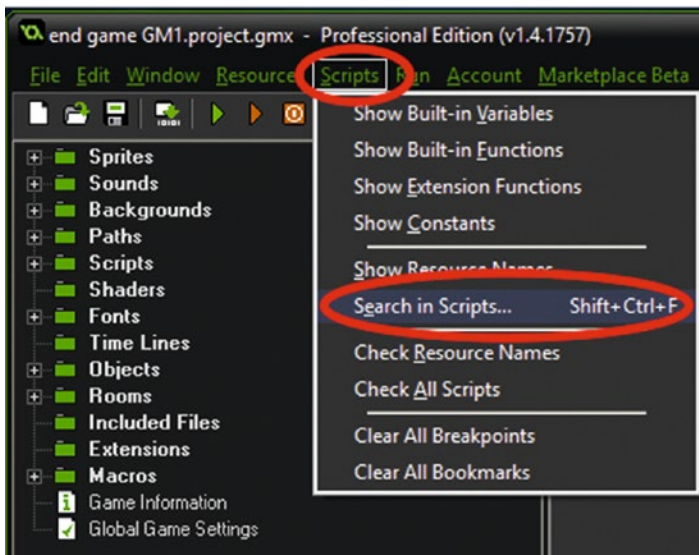


# Hints and Tips

## Scripts Tricks

When you've finished your game, or while making it, you may want to make changes to your code. Here are some tips for code handling.

You can search in scripts by clicking **Scripts** ► **Search in scripts**. This is shown below in Figure 25-1.



**Figure 25-1.** Starting a search in scripts

For example, you could search everything for `global.selected_weapon`; this search and result is shown in Figure 25-2 and the result in Figure 25-3.



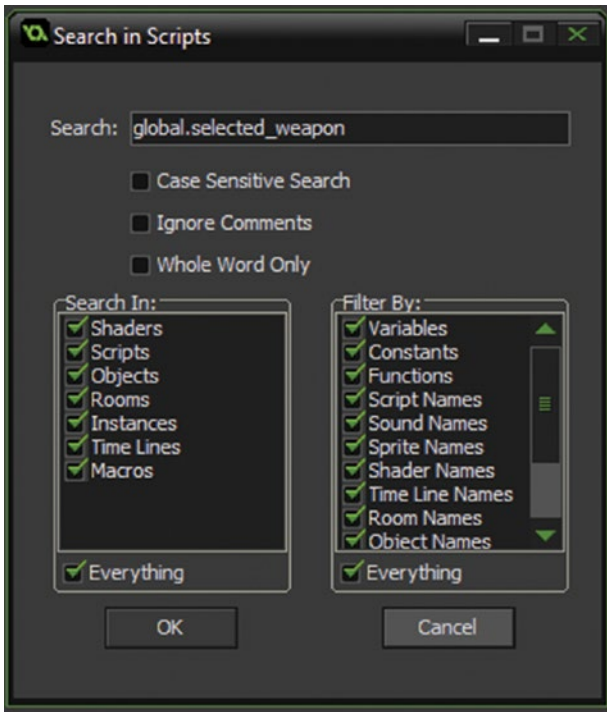


Figure 25-2. An example search

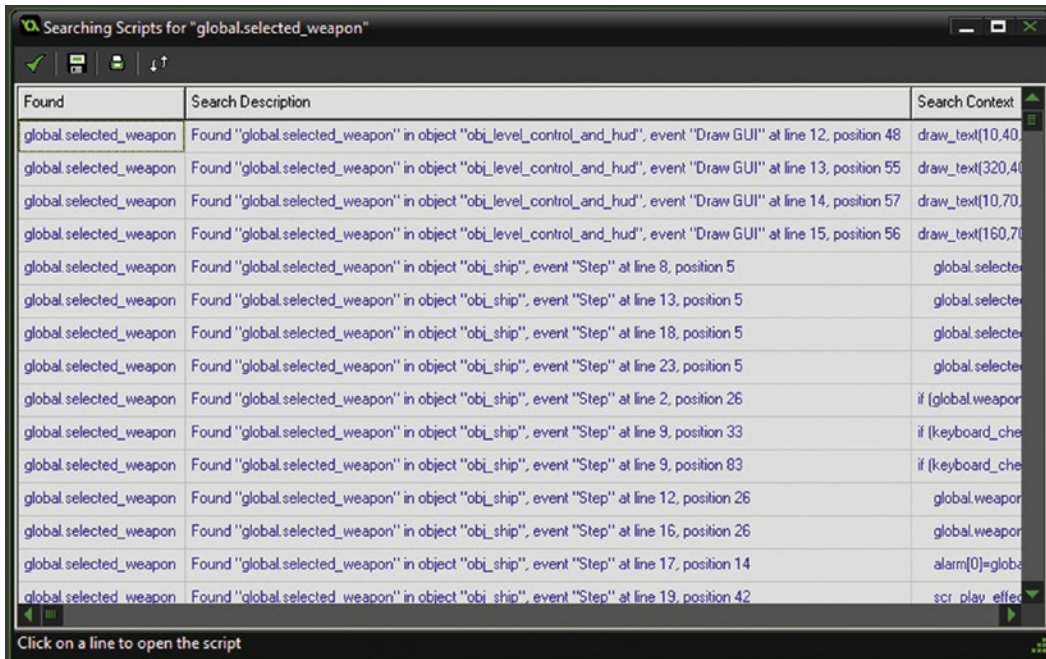


Figure 25-3. Showing search results

You can then click on each result to be taken directly to where that search is found.

## Testing

When testing, you don't want to spend hours repeating the same sections of your game to get to the bit you want to test. For example, if your game had four levels, you wouldn't want to have to spend time repeatedly going through levels 1 to 3 just to test the fourth level.

What you can do is create shortcuts in your game to allow you to do things like change the level: **health**, **lives**, etc. Sometimes programmers leave these in the final game – these are known as easter-eggs. You've probably played a console game where you enter a certain button combination used to unlock things; most of these were put in for the benefit of testers (and sometimes for the player).

An example would be:

```
if keyboard_check_released(ord('Q'))
{
    global.level+=1;
}
if keyboard_check_released(ord('A'))
{
    global.level-=1;
}
```

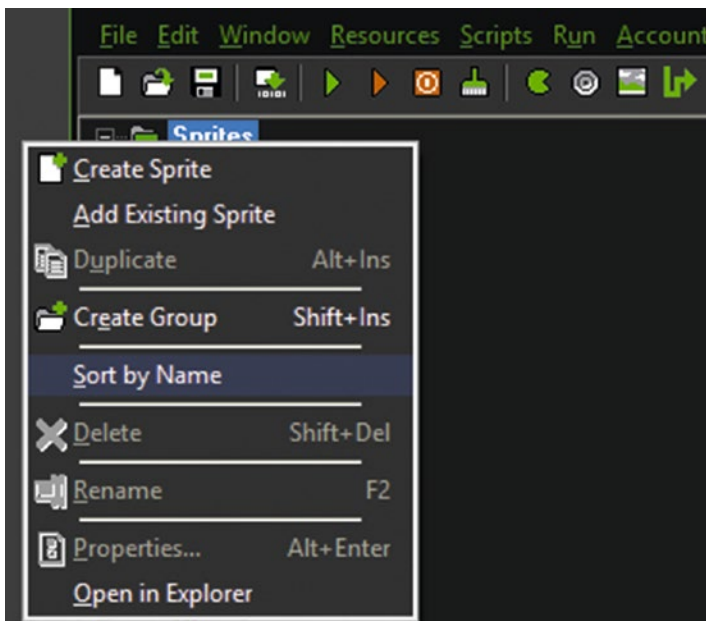
Placed in an object on a level select screen would allow you to change/unlock levels.

You can easily do other key combinations to change other variables as needed.

## Assets Handling

It's important to keep your assets well named and tidy.

You can right-click on a folder or section and select **Sort by Name**, as shown in Figure 25-4.



**Figure 25-4.** *Sorting assets by name*

Alternatively you can click on individual assets and drag them to a new location, as there may be times where you want assets in a non-alphabetical order.

Similarly, you can **Create a Group** (a folder) and divide assets into sections; this is a great way to organize your assets and find them quickly. This is shown in Figure 25-5.



**Figure 25-5.** *Sorting assets into groups*

After creating a group, you can click and drag an asset to that group as needed, and sort the contents of the group.

# Projects

The final project combines what you have learned previously. Each is 1 point.

- A) Create a background and control object that makes the background change direction every 5 seconds. Make it move up, down, left, right, and diagonally.
- B) Create a movable player object and an enemy object that moves towards the player every 5 seconds. Only an enemy can wrap around the screen. Make both point in the direction traveling.
- C) Make the enemy shoot in the direction every time it changes direction, in the direction it's moving. Player can also shoot in direction that it's moving, no quicker than once that every 4 seconds.
- D) Create a small health bar above player and enemy objects. Set it so being hit by a bullet reduces health.
- E) Allow player to place a bomb anywhere onscreen using the mouse. Bomb detonates after 3 seconds. If player or enemy are in range, they lose health.
- F) Make it rain if player has more health, snow if the enemy has more.
- G) Create visual and audio effects if a bullet hits player or enemy, or when bomb explodes.

## CHAPTER 26



# Creating a Game – Outline

First up, note that the process of making a game is never as shown in the following chapters. The purpose of these chapters is to demonstrate how you can apply what has been learned in the book in the context of a game. If you were creating a game from scratch, you would generally make some design notes prior to starting to program. You would then start off with a single object, that is, `obj_ship`, and build the game up from there, adding new elements and constantly testing.

We'll create an arcade style shooting game.

At this point you now should be able to understand what all the GML code in this example game does. Explanations are still given, so you can understand how the code is combined and actually applied to the making of the game.

This will also give you extra insight into programming and game implementation that you can apply in the final book projects.

As before, all assets for this game are in the downloadable resource at:

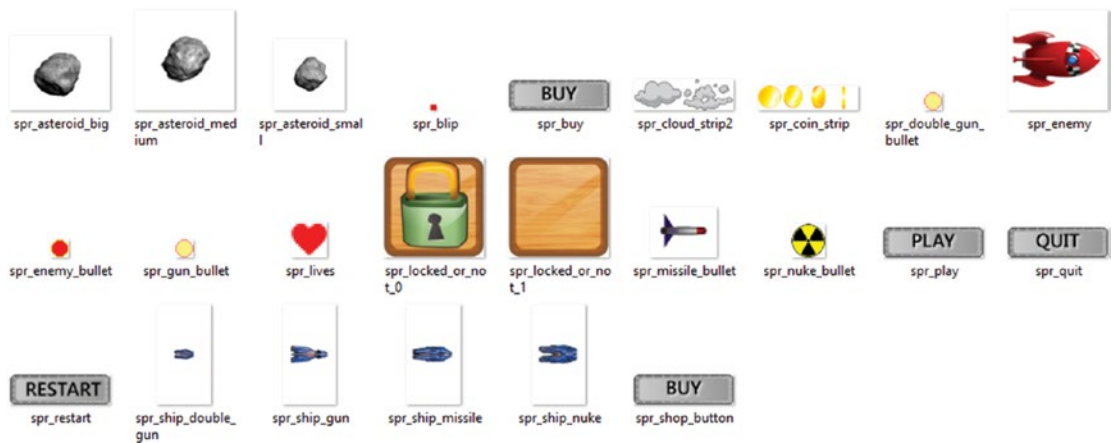
**Project Assets & GMZ Files > Final Game Assets**

## CHAPTER 27



# Creating a Game – Sprites

All assets for the game are included; you just need to load them in, and they are as shown in Figure 27-1:



*Figure 27-1. All sprites used in game*

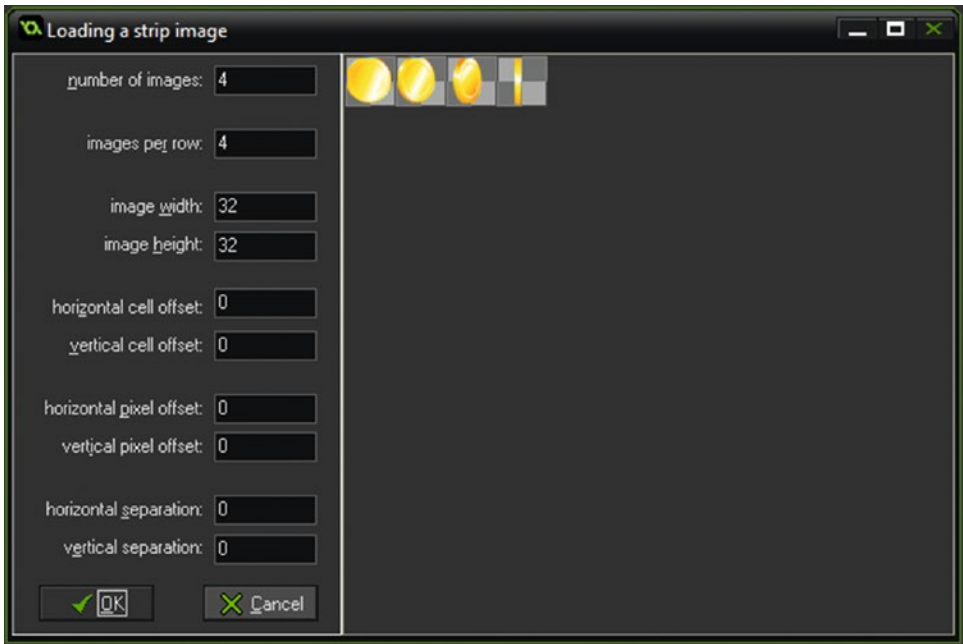
Sprites are named as the asset that you'll be naming them, except for **spr\_coin** and **spr\_cloud** which make use of an image strip. To create a strip create a new sprite, name it and click **edit**, followed by **File** then **Edit, File** then **Create From Strip**, then click the image to load and apply the settings needed.

---

■ **Note** You can drag these assets into the resource tree from the folder and rename as required.

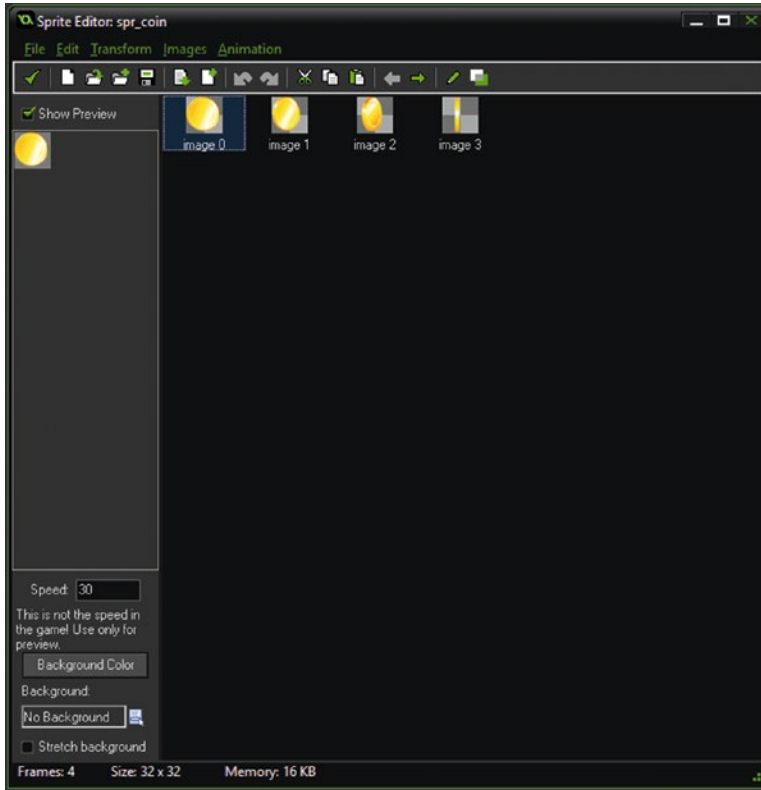
---

The settings for **spr\_coin** is shown in Figure 27-2:



*Figure 27-2. Showing strip settings for spr\_coin\_strip31*

When saved, by clicking OK, you can click and preview to check it all loaded OK: for example, as shown in Figure 27-3:



**Figure 27-3.** Showing sprite with sub images



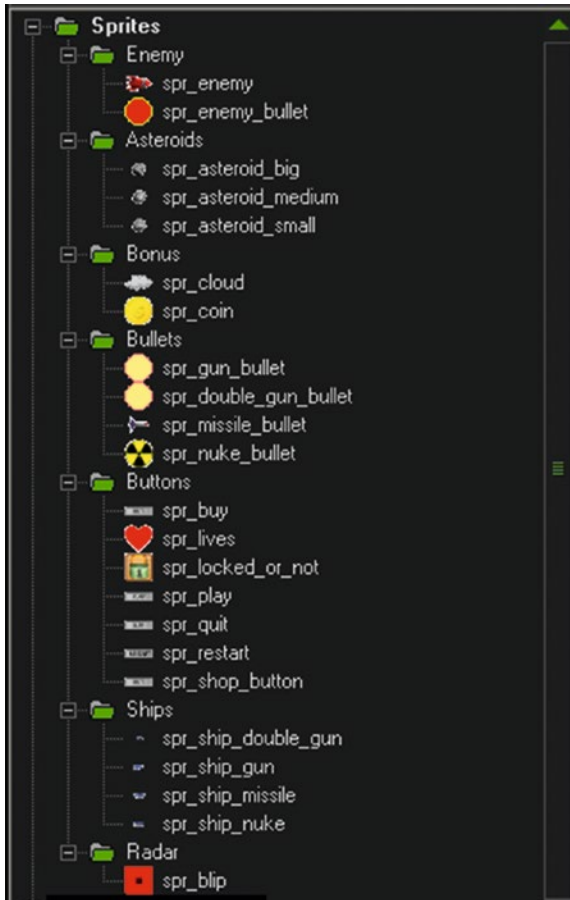
Settings for `spr_cloud` are shown in Figure 27-4:



*Figure 27-4. Showing strip settings for `spr_cloud`*

The sprite origin is center, unless otherwise indicated.

Once loaded and organized, your resource tree for the sprites will look like Figure 27-5.



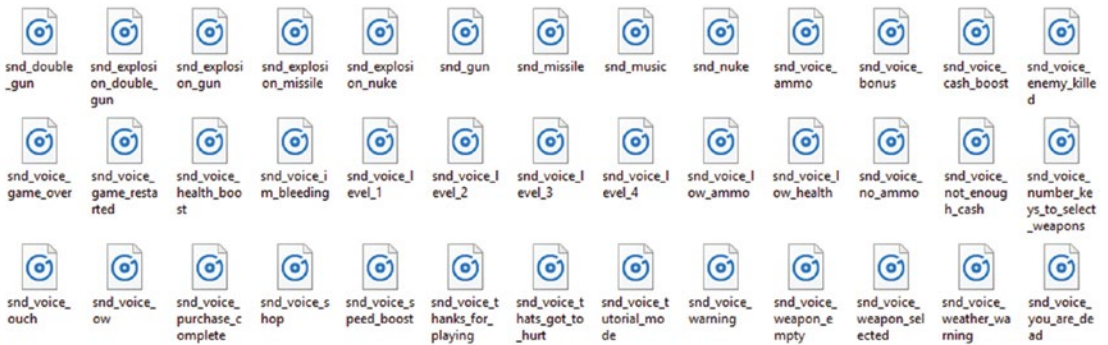
*Figure 27-5. Showing sprites loaded and organized*

## CHAPTER 28



# Creating a Game – Sounds

Next is to load in the sounds. They are all ready to use from the assets folder. Each file is named the same as the resource; you just need to load them in. It's fine to use the default sound settings for each of them. Your resource folder will appear as shown in Figure 28-1.



**Figure 28-1.** Showing sound assets folders

---

**Note** You can drag these assets into the resources tree and rename as needed.

---

Once loaded in and organized, your resource tree will look like Figure 28-2.



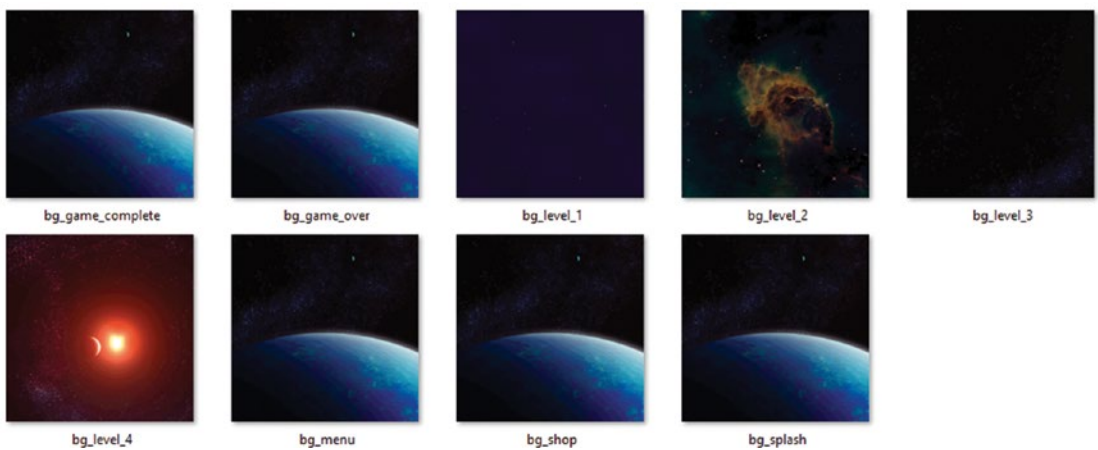
Figure 28-2. Showing sounds loaded and organized

## CHAPTER 29



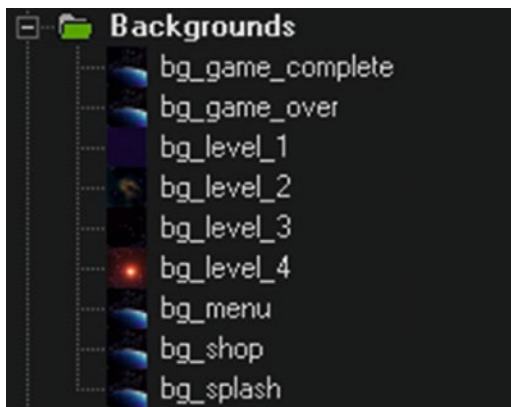
# Creating a Game – Backgrounds

The backgrounds are ready to load, and again have been the same as the assets. Your background folder will look like the one as shown in Figure 29-1:



**Figure 29-1.** Showing background files in folder

After loading into the game, the resource tree will be as shown in Figure 29-2.



**Figure 29-2.** Showing assets added to game

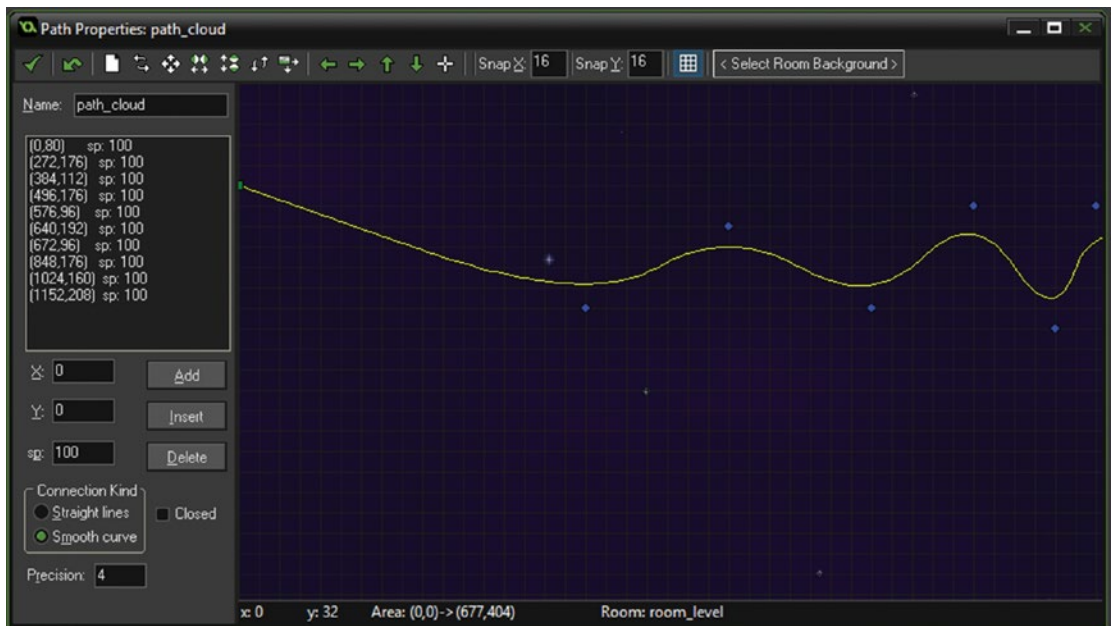
## CHAPTER 30



# Creating a Game – Paths

The game makes use of two paths: one for the cloud bonus and one for the coin bonus.

Set the paths up as shown in Figure 30-1 and Figure 30-2; you don't need to be 100% accurate.



**Figure 30-1.** Showing path `path_cloud`

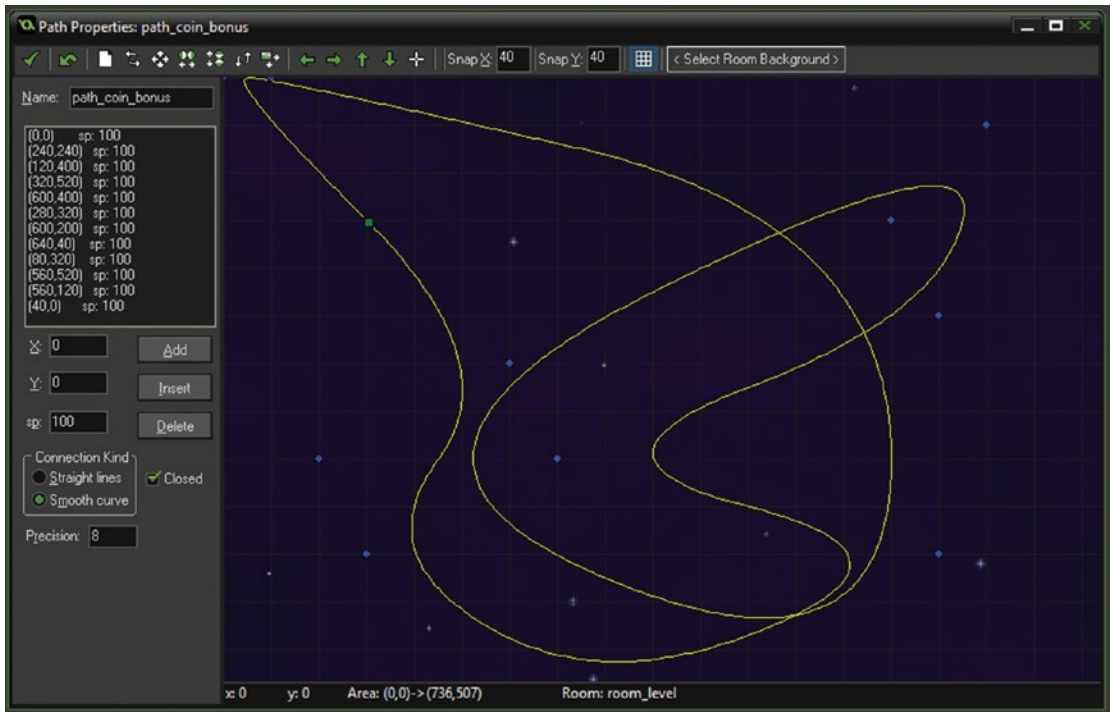


Figure 30-2. Showing path\_coin\_bonus

## CHAPTER 31



# Creating a Game – Fonts

This game makes use of seven fonts:

- font\_asteroid\_text** - Arial size 12
- font\_lock** - Arial size 25
- font\_menu\_stats** - Garamond size 22
- font\_message** - Arial size 50
- font\_mini\_message** - Arial size 15
- font\_restart** - Arial size 14
- font\_shop** - Arial size 18

Create the fonts as listed above.



## CHAPTER 32

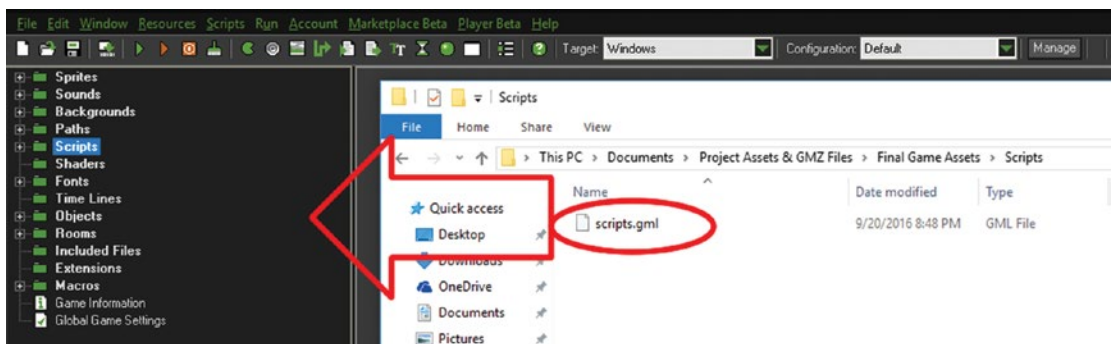


# Creating a Game – Scripts

This game uses a lot of scripts. The script file is available in the resources at:

**Project Assets & GMZ Files > Final Game Assets > Scripts**

To add these scripts to your project, just drag the file across to your resource tree, as shown in Figure 32-1 below:

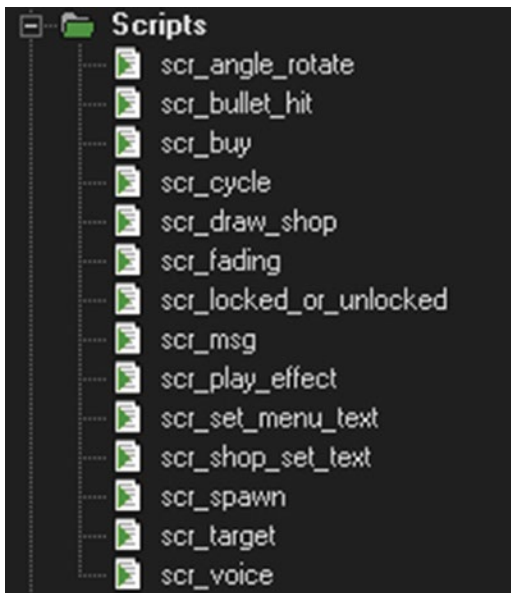


**Figure 32-1.** Importing script file

Once added, your resource tree will look like Figure 32-2 on next page.

***scr\_angle\_rotate***

```
//find rotation angle  
return argument0 + max(min(scr_cycle(argument1 - argument0, -180, 180), argument2), -argument2);
```



**Figure 32-2.** Showing all scripts added

This script finds the shortest way to rotate to a given angle: for example if it's at 350' and the target is at 10', it will work out that it needs to rotate 20' clockwise.

***scr\_bullet\_hit***

```
//do if hit by bullet
my_health-=other.strength;
global.hits+=1;
with (other) instance_destroy();
```

This script reduces the `my_health` value by the strength of the colliding bullet, makes note of this increasing **global.hits**, and then destroys the bullet.

***scr\_buy***

```
//check if player can buy
if position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left)
{
  if global.cash>=global.weapon_info[2,button_id] //check if enough cash
  {
    global.cash-=global.weapon_info[2,button_id]; //take cash away
    global.weapon_info[3,button_id]+=1; //add ammo
    scr_voice(snd_voice_purchase_complete);
  }
  else scr_voice(snd_voice_not_enough_cash);
}
```

The previous script checks for a mouse button released over itself. It then checks if the player has enough cash; if it does it takes the cash away, increases the ammo of that bullet, and plays the voice **purchase complete**. If there is not enough cash to purchase it, it plays the voice **not enough cash**.

#### *scr\_cycle*

```
// cycle - works with scr_angle_rotate
argument0 = (argument0 - argument1) mod (argument2 - argument1)
if (argument0 < 0) return argument0 + argument2
return argument0 + argument1;
```

#### *scr\_draw\_shop*

```
//draw button and text
draw_self();
scr_shop_set_text();
draw_text(50,y, global.weapon_info[0,button_id]); //draw name
draw_text(250,y, global.weapon_info[1,button_id]); //strength
draw_text(350,y, global.weapon_info[2,button_id]); //cost
draw_text(450,y, global.weapon_info[3,button_id]); //current ammo
```

This script draws a button for the object, sets text to the font for the shop, then draws the values of the weapon in a line, with the y relative to where the button has been placed.

#### *scr\_fading*

```
//reduce alpha when not live
if (!live) image_alpha-=0.05;
if image_alpha<0 instance_destroy();
```

If live has been set to false (after an alarm has triggered), it will fade the object by 0.05 per step and then destroy it when alpha is below 0.

#### *scr\_locked\_or\_unlocked*

```
///Check If Unlocked - Goto Shop If OK
if global.level>=my_id
{
    locked=false;
}
else
{
    locked=true;
}
if locked
{
    image_index=0;
}
else
{
    image_index=1;
}
```

```
if (!locked && position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left))
{
    global.my_level=my_id;
    room_goto(room_shop);
}
```

Displays locked or unlocked image depending on current level. If unlocked, player can click it to start level.

#### ***scr\_msg***

```
//add message to a ds list
ds_list_add(global.message,argument0);
```

Cues up a message to be used by the message object.

#### ***scr\_play\_effect***

```
//plays a sound effect
audio_play_sound(argument0,2,0);
```

This script plays a given sound effect.

#### ***scr\_set\_menu\_text***

```
//this script sets drawing font, alignment and colour
draw_set_font(font_shop);
draw_set_colour(c_white);
draw_set_halign(fa_left);
draw_set_valign(fa_middle);
```

The above script sets the drawing text style. Although a script isn't really required, it does make the code in the calling object a bit tidier.

#### ***scr\_shop\_set\_text***

```
//this script sets drawing font, alignment and colour
draw_set_font(font_shop);
draw_set_colour(c_red);
draw_set_halign(fa_left);
draw_set_valign(fa_middle);
```

Used to set drawing text style. Used to make calling object code tidier.

#### ***scr\_spawn***

```
var asteroid=instance_create(argument0,argument1,argument2);
asteroid.my_health=argument3;
asteroid.my_starting_health=argument4;
i=irandom_range(argument5,argument6);
asteroid.rotate=choose(i,-i);
asteroid.direction=argument7;
asteroid.speed=irandom_range(argument8,argument9);
```

The previous script spawns asteroids with the give attributes.

***scr\_target***

```
//Find a nearest object  
return instance_nearest(argument0,argument1,argument2);
```

Although only one line, it's used a few times, so it justifies having a script.

***scr\_voice***

```
//play a voice sound effect]  
audio_play_sound(argument0,1,0);
```

Used to play a voice sound effect. Voices are used a lot, and this script makes the calling object's code easier to read.

## CHAPTER 33

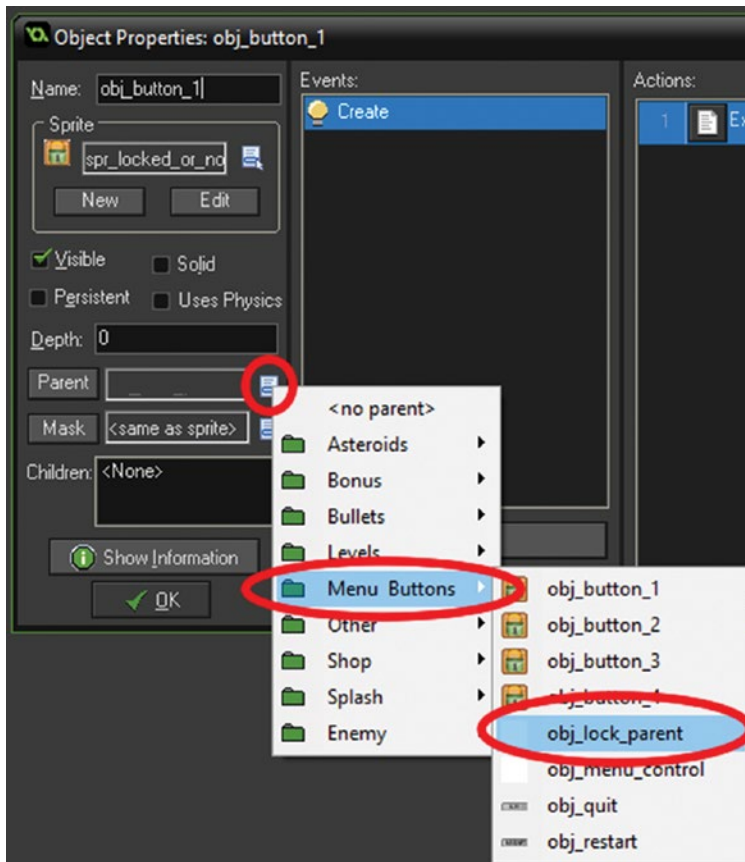


# Creating a Game – Parent Objects

In order to allow you to create the objects in a progressive, organized manner, first we're going to create objects that are used as parent objects.

You assign parents from other objects. Put basically, a child object will perform events implemented by its parent, except those that have been explicitly specified in the child.

For example, assigning **obj\_lock\_parent** as a **parent** of **obj\_button\_1**, as shown in Figure 33-1:



**Figure 33-1.** Assigning a parent object

The following objects are parent objects:

**obj\_bullet\_parent**  
**obj\_asteroid\_parent**  
**obj\_lock\_parent**  
**obj\_shop\_button\_parent**

The information for each of these objects is in the resource folder:

**Project Assets & GMZ Files ► Final Game Assets ► Parent Objects**

Each file can be opened in a text editor such as Notepad.

You can then use the information to create the object.

Start with **obj\_bullet\_parent**.

The contents of this file look like this:

**Information about object: obj\_bullet\_parent**

**Sprite:**

**Solid: false**

**Visible: true**

**Depth: 0**

**Persistent: false**

**Parent:**

**Children**

**obj\_double\_gun\_bullet**

**obj\_gun\_bullet**

**obj\_missile\_bullet**

**obj\_nuke\_bullet**

**Mask:**

**No Physics Object**

**Alarm Event for alarm 0:**

**execute code:**

```
///Alarm trigger - start fading  
live=false;
```

**Step Event:**

**execute code:**

```
///Create fading if enabled  
scr_fading();
```

wrap in both directions when an instance moves outside the room

The header tells you basic info:

```
Information about object: obj_bullet_parent
Sprite:
Solid: false
Visible: true
Depth: 0
Persistent:
```

The object name is: **obj\_bullet\_parent**, it has no sprite, isn't set to solid and has a depth of 0 and is not persistent.

This provides all information required to re-create this object. For example, this part:

**Alarm Event for alarm 0:**

**execute code:**

```
///Alarm trigger - start fading
live=false;
```

Tells you to create an **Alarm[0] Event** with the code:

```
///Alarm trigger - start fading
live=false;
```

The following tells you it has a **Step Event** that calls `scr_fading()` :

**Step Event:**

**execute code:**

```
///Create fading if enabled
scr_fading();
```

This part:

wrap in both directions when an instance moves outside the room

Indicates that a D&D action has been used, in this case the **Wrap Room In Both Direction**, which can be found in the **Move Tab**. This is in the **Step Event**.

Complete for the remaining objects in this resource folder.

The rest of the information for the other parent objects is below.

**obj\_asteroid\_parent**

**Information about object: obj\_asteroid\_parent**

**Sprite:**

**Solid: false**

**Visible: true**

**Depth: 0**

**Persistent: false**

**Parent:**



**Children**

**obj\_asteroid\_big**  
**obj\_asteroid\_medium**  
**obj\_asteroid\_small**  
**obj\_enemy**

**Mask:****No Physics Object**

**Collision Event** with object **obj\_bullet\_parent**:

**execute code:**

```
scr_bullet_hit();
```

**Draw Event:****execute code:**

```
///draw self and a healthbar

draw_self();
if health>0
{
    draw_set_colour(c_blue);
    draw_rectangle(x-50,y-50,x+50,y-40,0)
    draw_set_colour(c_red);
    draw_rectangle(x-50,y-50,x-50+((100/my_starting_health)*my_health),y-40,0);
    draw_set_colour(c_black);
    draw_rectangle(x-50,y-50,x+50,y-40,1)
}
```

**obj\_lock\_parent**

Information about object: **obj\_lock\_parent**

Sprite:

Solid: false

Visible: true

Depth: 0

Persistent: false

Parent:

Children

obj\_button\_1

obj\_button\_2

obj\_button\_3

obj\_button\_4

Mask:

No Physics Object

**Step Event:****execute code:**

```

///Check If Unlocked - Goto Shop If OK
if global.level>=my_id
{
    locked=false;
}
else
{
    locked=true;
}
if locked
{
    image_index=0;
}
else
{
    image_index=1
}
if (!locked && position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left))
{
    //level check
    if global.level=my_id
    {
        global.current=true;
    }
    else
    {
        global.current=false;
    }
    global.my_level=my_id;
    room_goto(room_shop);
}

```

**Draw Event:****execute code:**

```

///Draw Locked / Unlocked & ID
draw_self();
draw_set_font(font_lock);
draw_set_valign(fa_middle);
draw_set_halign(fa_center);
draw_set_colour(c_black); //These two lines create
draw_text(x-1,y-1,my_id); //a shadow effect
draw_set_colour(c_white);
draw_text(x,y,my_id); //Draw Over Black Text

```

***obj\_shop\_button\_parent***

**Information about object: obj\_shop\_button\_parent**

**Sprite:**

**Solid: false**

**Visible: true**

**Depth: 0**

**Persistent: false**

**Parent:**

**Children**

**obj\_shop\_button\_1**

**obj\_shop\_button\_2**

**obj\_shop\_button\_3**

**obj\_shop\_button\_4**

**Mask:**

**No Physics Object**

**Step Event:**

**execute code:**

```
/// Buy If Enough && voices  
scr_buy(button_id);
```

**Draw Event:**

**execute code:**

```
//Draw Button & Data  
scr_draw_shop();  
draw_self();
```

## CHAPTER 34



# Creating a Game – Objects

As before, the text files for these objects are available in the resources, at: **Project Assets & GMZ Files** ►  
**Final Game Assets** ► **Object**

In the object information below, where this is more than one **execute code**: within an event, each is a separate code block.

Next repeat the process with the remaining object; the suggested order is below:

*obj\_asteroid\_big*

**Information about object: obj\_asteroid\_big**

**Sprite: spr\_asteroid\_big**

**Solid: false**

**Visible: true**

**Depth: 0**

**Persistent: false**

**Parent: obj\_asteroid\_parent**

**Children:**

**Mask:**

**No Physics Object**

**Step Event:**

**execute code:**

```
///Rotate & Health Check  
image_angle+=rotate;
```

```
if my_health<1
```

```
{
```

```
    scr_msg("Large##Asteroid##Destroyed");
```

```
    //Add cash
```

```
    global.cash+=my_starting_health;
```

```
    //create a medium first
```

```
    scr_spawn(x,y,obj_asteroid_medium,my_starting_health/2,my_starting_health/2,rotate*-1,rotate*-1,direction-30,speed,speed);
```

```
    scr_spawn(x,y,obj_asteroid_medium,my_starting_health/2,my_starting_health/2,rotate*-1,rotate*-1,direction+30,speed,speed);
```

```

//create enemy
enemy=instance_create(x,y,obj_enemy);
enemy.my_health=20*global.level*global.level;
enemy.my_starting_health=20*global.level*global.level;

instance_destroy();
}

```

wrap in both directions when an instance moves outside the room

### ***obj\_asteroid\_medium***

**Information about object: obj\_asteroid\_medium**

**Sprite: spr\_asteroid\_medium**

**Solid: false**

**Visible: true**

**Depth: 0**

**Persistent: false**

**Parent: obj\_asteroid\_parent**

**Children:**

**Mask:**

**No Physics Object**

**Step Event:**

**execute code:**

```

///Rotate & Health Check
image_angle+=rotate

if my_health<1
{
//Add cash
global.cash+=my_starting_health;
scr_spawn(x,y,obj_asteroid_small,my_starting_health/2,my_starting_health/2,rotate*-1,rotate*-1,direction-50,speed*1.5,speed*1.5);
scr_spawn(x,y,obj_asteroid_small,my_starting_health/2,my_starting_health/2,rotate*-1,rotate*-1,direction+50,speed*1.5,speed*1.5);
scr_spawn(x,y,obj_asteroid_small,my_starting_health/2,my_starting_health/2,rotate*-1,rotate*-1,direction-180,speed*1.5,speed*1.5);

instance_destroy();
}

```

wrap in both directions when an instance moves outside the room

### ***obj\_asteroid\_small***

**Information about object: obj\_asteroid\_small**

**Sprite: spr\_asteroid\_small**

**Solid: false**

**Visible: true**  
**Depth: 0**  
**Persistent: false**  
**Parent: obj\_asteroid\_parent**  
**Children:**  
**Mask:**  
**No Physics Object**

**Step Event:**

**execute code:**

```

///Rotate & Health Check
image_angle+=rotate;

if my_health<1
{
    global.cash+=my_starting_health;
    instance_destroy();
}

```

wrap in both directions when an instance moves outside the room

***obj\_cloud***

**Information about object: obj\_cloud**  
**Sprite: spr\_cloud**  
**Solid: false**  
**Visible: true**  
**Depth: 0**  
**Persistent: false**  
**Parent:**  
**Children:**  
**Mask:**  
**No Physics Object**

**Create Event:**

**execute code:**

```

///Set image and start path & play voice
image_index=0;
image_speed=0;
y=irandom_range(100,700);
path_start(path_cloud,irandom_range(2,10),path_action_continue,false);
scr_voice(snd_voice_weather_warning);

```

**Alarm Event for alarm 0:****execute code:**

```
instance_destroy();
```

**Collision Event with object obj\_bullet\_parent:****execute code:**

```
///Change Image When Hit By Bullet
if image_index==0
{
    global.cash+=1000;
    image_index=1;
}
alarm[0]=room_speed*3;
```

**obj\_coin****Information about object: obj\_coin****Sprite: spr\_coin****Solid: false****Visible: true****Depth: 0****Persistent: false****Parent:****Children:****Mask:****No Physics Object****Create Event:****execute code:**

```
///start path & play voice
y=irandom_range(200,600);
path_start(path_coin_bonus, 5, path_action_continue, false);
scr_voice(snd_voice_bonus);
```

**Collision Event with object obj\_ship:****execute code:**

```
///Increase Cash & Destroy
global.cash+=10000;
scr_voice(snd_voice_cash_boost);
scr_msg("Bonus Cash");
effect_create_above(ef_star,x,y,2,c_orange);
instance_destroy();
```

***obj\_double\_gun\_bullet***

**Information about object: obj\_double\_gun\_bullet**  
**Sprite: spr\_double\_gun\_bullet**  
**Solid: false**  
**Visible: true**  
**Depth: 0**  
**Persistent: false**  
**Parent: obj\_bullet\_parent**  
**Children:**  
**Mask:**  
**No Physics Object**

Create Event:

**execute code:**

```
///Set up
live=true;
alarm[0]=room_speed;
strength=global.weapon_info[1,2];
global.weapon_info[6,2]+=1;
```

Collision Event with object obj\_asteroid\_parent:

**execute code:**

```
///explosion and sound effect
scr_play_effect(snd_explosion_double_gun);
effect_create_above(ef_explosion,x,y,5,c_orange);
```

***obj\_gun\_bullet***

**Information about object: obj\_gun\_bullet**  
**Sprite: spr\_gun\_bullet**  
**Solid: false**  
**Visible: true**  
**Depth: 0**  
**Persistent: false**  
**Parent: obj\_bullet\_parent**  
**Children:**  
**Mask:**  
**No Physics Object**

Create Event:

**execute code:**

```
///Set up
live=true;
alarm[0]=room_speed;
strength=global.weapon_info[1,1];
global.weapon_info[6,1]+=1;
```



**Collision Event with object `obj_asteroid_parent`:****execute code:**

```

//explosion and sound effect
scr_play_effect(snd_explosion_gun);
effect_create_above(ef_explosion,x,y,2,c_red);

```

***obj\_missile\_bullet***

```

Information about object: obj_missile_bullet
Sprite: spr_missile_bullet
Solid: false
Visible: true
Depth: 0
Persistent: false
Parent: obj_bullet_parent
Children:
Mask:
No Physics Object

```

**Create Event:****execute code:**

```

///Set Up
live=true;
active=true;
alarm[0]=room_speed*5;
strength=global.weapon_info[1,3];
global.weapon_info[6,3]+=1;

```

**execute code:**

```

///Find a Target
if instance_exists(obj_asteroid_parent)
{
    target=scr_target(x,y,obj_asteroid_parent);
}

```

**Step Event:****execute code:**

```

///Look For Target - if destroyed find another
if instance_exists(target) && instance_exists(obj_asteroid_parent)
{
    target=scr_target(x,y,obj_asteroid_parent);
}
else//if no target available (ie all asteroids destroyed)

```

```
{
    active=false;
}
```

**execute code:**

```
///Set Moving Direction Angle
```

```
if active
{
    tx = target.x;
    ty = target.y;
    direction = scr_angle_rotate(direction, point_direction(x, y, tx, ty), 5);
    image_angle = direction;
}
```

**execute code:**

```
///Create Fading if enabled
scr_fading();
```

**Collision Event with object obj\_asteroid\_parent:****execute code:**

```
///Explosion Sound and Effects
scr_play_effect(snd_explosion_missile);
effect_create_above(ef_explosion,x,y,2,c_white);
```

**obj\_nuke\_bullet****Information about object: obj\_nuke\_bullet****Sprite: spr\_nuke\_bullet****Solid: false****Visible: true****Depth: 0****Persistent: false****Parent: obj\_bullet\_parent****Children:****Mask:****No Physics Object****Create Event:****execute code:**

```
///Set up
live=true;
alarm[0]=room_speed;
strength=global.weapon_info[1,4];
global.weapon_info[6,4]++;
```

**Collision Event with object obj\_asteroid\_parent:****execute code:**

```

//explosion sound and effects
scr_play_effect(snd_explosion_nuke);
repeat 20
{
    effect_create_above(ef_explosion,x+50-irandom(100),y+50-irandom(100),2,c_red);
}

```

***obj\_level\_control\_and\_hud*****Information about object: obj\_level\_control\_and\_hud****Sprite:****Solid: false****Visible: true****Depth: 0****Persistent: false****Parent:****Children:****Mask:****No Physics Object****Create Event:****execute code:**

```

//Set Background

if global.my_level==1 background_index[0]=bg_level_1;
if global.my_level==2 background_index[0]=bg_level_2;
if global.my_level==3 background_index[0]=bg_level_3;
if global.my_level==4 background_index[0]=bg_level_4;

```

**execute code:**

```

//Create Ship & Set Variables
instance_create(400,400,obj_ship);
//create message system
instance_create(10,10,obj_message);
//Show some user messages:
scr_msg("Get Ready");
scr_msg("Move With#Arrow Keys");
scr_msg("Fire With#Z");
scr_msg("Select Weapon#With# 1 2 3 4");
//set other variables
low_health=false;
//start timer
timer=0;
alarm[1]=room_speed;
//set flag for low health warning
health_alarm=false;

```

**execute code:**

```

/// Play Voice

if global.my_level==1 scr_voice(snd_voice_level_1);
if global.my_level==2 scr_voice(snd_voice_level_2);
if global.my_level==3 scr_voice(snd_voice_level_3);
if global.my_level==4 scr_voice(snd_voice_level_4);

```

**execute code:**

```

///Set Count to 0
count=0;

```

**execute code:**

```

/// Spawn Asteroids
//scr_spawn = Create an asteroid, x,y,asteroid,my_health,my_starting_health,min rotate, max
rotate,
//direction,speed min, speed max
//level 1 Spawn
if global.my_level==1
{
    //asteroid no 1
    scr_spawn(200,200,obj_asteroid_big,20,20,1,6,25,2,5);

    //asteroid no 2
    scr_spawn(600,600,obj_asteroid_big,20,20,1,6,180,2,5);

    //asteroid no 3
    scr_spawn(800,200,obj_asteroid_big,20,20,1,6,230,2,5);
}

//level 2 Spawn
if global.my_level==2
{
    //asteroid no 1
    scr_spawn(200,200,obj_asteroid_big,200,290,1,6,75,2,5);

    //asteroid no 2
    scr_spawn(600,600,obj_asteroid_big,200,200,1,6,300,2,5);

    //asteroid no 3
    scr_spawn(800,200,obj_asteroid_big,200,200,1,6,45,2,5);

    //asteroid no 4
    scr_spawn(800,200,obj_asteroid_big,200,200,1,6,130,2,5);
}

//level 3 Spawn
if global.my_level==3
{

```

```

//asteroid no 1
scr_spawn(200,200,obj_asteroid_big,200,290,1,6,75,2,5);

//asteroid no 2
scr_spawn(100,500,obj_asteroid_big,2000,2000,1,6,300,2,5);

//asteroid no 3
scr_spawn(800,200,obj_asteroid_big,2000,2000,1,6,45,2,5);

//asteroid no 4
scr_spawn(800,200,obj_asteroid_big,2000,2000,1,6,130,2,5);

//asteroid no 5
scr_spawn(800,200,obj_asteroid_small,2000,2000,1,6,130,2,5);

//asteroid no 6
scr_spawn(0,0,obj_asteroid_small,2000,2000,1,6,130,2,5);
}

//level 4 Spawn
if global.my_level==4
{
//asteroid no 1
scr_spawn(200,200,obj_asteroid_big,8000,8000,1,6,75,2,5);

//asteroid no 2
scr_spawn(100,500,obj_asteroid_big,2000,2000,1,6,300,2,5);

//asteroid no 3
scr_spawn(800,200,obj_asteroid_big,6000,6000,1,6,45,2,5);

//asteroid no 4
scr_spawn(800,200,obj_asteroid_big,20000,20000,1,6,130,2,5);

//asteroid no 5
scr_spawn(800,200,obj_asteroid_small,2000,2000,1,6,130,2,5);

//asteroid no 6
scr_spawn(0,0,obj_asteroid_small,4000,4000,1,6,130,2,5);
}

```

**Alarm Event for alarm 0:****execute code:**

```

///Alarm for Health warning
alarm[0]=room_speed*4; // set alarm at 4 seconds
scr_voice(snd_voice_low_health);
scr_msg("Low#Health");

```

**Alarm Event for alarm 1:****execute code:**

```

//Increase & Reset Timer
timer+=1;
alarm[1]=5;

```

**Step Event:****execute code:**

```

//Monitor Player
if health<=0
{
    health=100;
    lives-=1;
    scr_voice(snd_voice_you_are_dead);
    health_alarm=false;
    alarm[0]--;
}
if lives<0 room_goto(room_game_over);
//count shots of all weapons
global.total_shots=global.weapon_info[6,1]+global.weapon_info[6,2]+global.weapon_
info[6,3]+global.weapon_info[6,4];
if !object_exists(obj_asteroid_parent)
{
    global.level+=global.level;
    room_goto(room_menu);
}

//check if low health, if it is play warning and set a new alarm
if (health<25) && (!health_alarm=false)
{
    health_alarm=true;
    alarm[0]=1;
}

```

**execute code:**

```

//Timer Events
if timer==5
{
    // prevent activating twice
    timer+=1;
    //perform action
    instance_create(0,0,obj_cloud);
}

```

```

if timer==10
{
    // prevent activating twice
    timer+=1;
    //perform action
    instance_create(0,0,obj_coin);
}
if timer==100
{
    // prevent activating twice
    timer+=1;
    //perform action
    instance_create(0,0,obj_coin);
}
if timer==200
{
    // prevent activating twice
    timer+=1;
    //perform action
    instance_create(0,0,obj_cloud);
}

if timer==500 timer=0;
execute code:

///Instance Count
count=instance_number(obj_asteroid_parent);

execute code:

///Check If Level Complete
if (instance_number(obj_asteroid_parent) == 0)
{
    if global.current==true global.level+=1;
    room_goto(room_menu);
}

```

**Draw GUI Event:****execute code:**

```

///Draw Hud

scr_shop_set_text();//draw HUD background
//top bar
draw_set_colour(c_blue);
draw_rectangle(0,0,800,90,0);
//border
draw_rectangle(1,1,799,799,1);

```

```
//draw weapon and ammo info
draw_set_font(font_shop);
draw_set_colour(c_white);
draw_text(10,40,"Type: "+global.weapon_info[0,global.selected_weapon]);
draw_text(320,40,"Ammo: "+string(global.weapon_info[3,global.selected_weapon]));
draw_text(10,70,"Stength: "+string(global.weapon_info[1,global.selected_weapon]));
draw_text(160,70,"Fired: "+string(global.weapon_info[6,global.selected_weapon]));
draw_text(300,70,"Fired All: "+string(global.total_shots));
draw_text(450,70,"Total Hits: "+string(global.hits));
draw_text(620,70,"Cash: "+string(global.cash));
```

draw the lives at (500,40) with sprite spr\_lives

draw the health bar with size (2,2,798,28) with back color blue and bar color green to red

**execute code:**

```
//Draw Radar
draw_set_alpha(0.2);
draw_circle(75,500,75,0);
draw_set_alpha(1);
draw_set_colour(c_green);
draw_circle(75,500,75,1);
draw_circle(75,500,10,1);
draw_line(75,425,75,575);
draw_line(0,500,150,500);

//set internal radar coords to Ship.x and Ship.y
if (!instance_exists(obj_ship))
{
    exit;
}

var d,a,radarX,radarY;

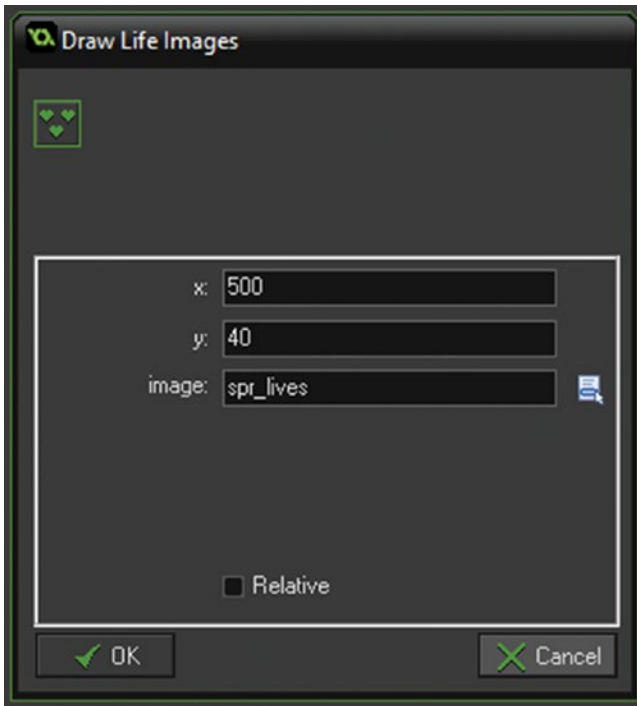
radarX = obj_ship.x;
radarY = obj_ship.y;

with(obj_asteroid_parent)
{
    //how far
    d = point_distance(radarX,radarY,x,y);
    //in range
    if( d < 800 && d > 600) // This will set the blips to the outside edge, creating a
    lingering effect
    {
        //convert radar range to radar display radius
        d = 75;
        //angle to target
        a = point_direction(radarX,radarY,x,y)
        //draw relative to center of radar using simplified lengthdir function
        draw_sprite(spr_blip, 0, 75 + lengthdir_x(d,a), 700 + lengthdir_y(d,a));
    }
}
```



```
else if(d <= 600) // This is the standard distance conversion on the radaar screen.
{
    d = d/600*75;
    a = point_direction(radarX,radarY,x,y)
    draw_sprite(spr_blip, 0, 75 + lengthdir_x(d,a), 500 + lengthdir_y(d,a));
}
}
```

For reference, Figure 34-1 shows how the Draw Life as Images D&D looks:



**Figure 34-1.** Showing draw lives as image

Figure 34-2 below shows how the Draw Health Bar D&D will look:



Figure 34-2. Drawing health bar D&D

### ***obj\_ship***

**Information about object: obj\_ship**

**Sprite: spr\_ship\_gun**

**Solid: false**

**Visible: true**

**Depth: 0**

**Persistent: false**

**Parent:**

**Children:**

**Mask:**

No Physics Object

**Create Event:**

**execute code:**

```
///Set Variables
can_be_hit=true;
can_shoot=true;
low_ammo=false;
```

```

ship_speed=0;
direction=0;
rotation=0;
turning_speed=2;
acc_speed=0.3;
max_speed=7;
friction=0.1;
slowing_speed=0.1;

```

**Alarm Event for alarm 0:**

**execute code:**

```

///Set Shooting to True
can_shoot=true;

```

**Alarm Event for alarm 1:**

**execute code:**

```

///Can Be Hit (can lose health)
can_be_hit=true;

```

**execute code:**

```

//Low Ammo Set
low_ammo=false;

```

**Step Event:**

**execute code:**

```

///Weapon Keyboard Control

//change weapon / ship type
if keyboard_check_pressed(ord('1'))
{
    global.selected_weapon=1;
    scr_voice(snd_voice_weapon_selected);
}
if keyboard_check_pressed(ord('2'))
{
    global.selected_weapon=2;
    scr_voice(snd_voice_weapon_selected);
}
if keyboard_check_pressed(ord('3'))
{
    global.selected_weapon=3;
    scr_voice(snd_voice_weapon_selected);
}

```

```

if keyboard_check_pressed(ord('4'))
{
    global.selected_weapon=4;
    scr_voice(snd_voice_weapon_selected);
}

```

**execute code:**

```

//Spawn Bullets
if (global.weapon_info[3,global.selected_weapon]==0 && keyboard_check(ord('Z')) && can_
shoot)
{
scr_voice(choose(snd_voice_weapon_empty,snd_voice_no_amm0,snd_voice_number_keys_to_select_
weapons));
    can_shoot=false;
    alarm[0]=room_speed; // Stops constant playing on keypress
}
//create weapon 1 bullet
if (keyboard_check(ord('Z')) && global.selected_weapon==1 && global.weapon_info[3,global.
selected_weapon]>0 && can_shoot)

{
    global.weapon_info[6,global.selected_weapon]+=1;
    bullet_id = instance_create(x,y,obj_gun_bullet);
    bullet_id.direction = image_angle;
    bullet_id.speed = 6+speed;
    global.weapon_info[3,global.selected_weapon]-=1;
    alarm[0]=global.selected_weapon*8;
    can_shoot=false;
    scr_play_effect(global.weapon_info[5,global.selected_weapon]);
}

//create weapon 2 bullets
if (keyboard_check(ord('Z')) && global.selected_weapon==2 && global.weapon_info[3,global.
selected_weapon]>0 && can_shoot)

{
    //This Weapon Only Has Two Bullets
    //one
    global.weapon_info[6,global.selected_weapon]+=1;
    bullet_id = instance_create(x,y,obj_double_gun_bullet);
    bullet_id.direction = image_angle-12;
    bullet_id.speed = 8+speed;

    //two
    global.weapon_info[6,global.selected_weapon]+=1;
    bullet_id = instance_create(x,y,obj_double_gun_bullet);
    bullet_id.direction = image_angle+12;
    bullet_id.speed = 8+speed;
}

```

```

    global.weapon_info[3,global.selected_weapon]-=2;
    alarm[0]=global.selected_weapon*8;
    can_shoot=false;
    scr_play_effect(global.weapon_info[5,global.selected_weapon]);
}

//create weapon 3 bullet
if (keyboard_check(ord('Z')) && global.selected_weapon==3 && global.weapon_info[3,global.
selected_weapon]>0 && can_shoot)

{
    global.weapon_info[6,global.selected_weapon]+=1;
    bullet_id = instance_create(x,y,obj_missile_bullet);
    bullet_id.direction = image_angle;
    bullet_id.speed = 5+speed;
    global.weapon_info[3,global.selected_weapon]-=1;
    alarm[0]=global.selected_weapon*4;
    can_shoot=false;
    scr_play_effect(global.weapon_info[5,global.selected_weapon]);
}

//create weapon 4 bullet
if (keyboard_check(ord('Z')) && global.selected_weapon==4 && global.weapon_info[3,global.
selected_weapon]>0 && can_shoot)

{
    global.weapon_info[6,global.selected_weapon]+=1;
    bullet_id = instance_create(x,y,obj_nuke_bullet);
    bullet_id.direction = image_angle;
    bullet_id.speed = 10+speed;
    global.weapon_info[3,global.selected_weapon]-=1;
    alarm[0]=global.selected_weapon*4;
    can_shoot=false;
    scr_play_effect(global.weapon_info[5,global.selected_weapon]);
}

```

**execute code:**

```

//Movement Code
if (image_angle > 360) image_angle -= 360;
if (image_angle < 0) image_angle += 360;
if keyboard_check(vk_up)
{
    if ship_speed<max_speed
    {
        ship_speed=ship_speed+acc_speed+(ship_speed/6);
    }
}

```

```

if keyboard_check(vk_down)
{
    if ship_speed>0
    {
        ship_speed-=.5;
    }
    else if ship_speed>-3
    {
        ship_speed-=.05;
    }
}
if ! keyboard_check(vk_down) && ! keyboard_check(vk_up)
{
    if ship_speed<-.5
    {
        ship_speed+=.5;
    }
    else if ship_speed>.5
    {
        ship_speed-=.2;
    }
    else ship_speed=0;
}
if keyboard_check(vk_right)
{
    if rotation>-40
    {
        rotation-=turning_speed;
    }
}
if keyboard_check(vk_left)
{
    if rotation<40
    {
        rotation+=turning_speed;
    }
}
if ! keyboard_check(vk_left) && ! keyboard_check(vk_right)
{
    if rotation<-5
    {
        rotation+=5;
    }
    else if rotation>5
    {
        rotation-=5;
    }
    else rotation=0;
}

```

```

if ship_speed>0
{
    image_angle+=ship_speed/30*rotation;
}
else
if ship_speed<0
{
    image_angle+=ship_speed/20*rotation;
}
if image_angle>360
{
    image_angle-=360;
}
if image_angle<0
{
    image_angle+=360;
}
friction=speed/10;
motion_add(image_angle,ship_speed/6);

```

**execute code:**

```

///Low Ammo Warning

if (global.weapon_info[3,global.selected_weapon]<5 &&global.weapon_info[3,global.selected_
weapon]>0 && !low_ammo)
{
    low_ammo=true;
    alarm[2]=room_speed*3;
    scr_voice(snd_voice_low_ammo);
}

```

**execute code:**

```

///Testing
////////////////////////////////////testing buttons
//for testing reset to middle
if keyboard_check(ord('R')) {room_restart();}
if keyboard_check(ord('A')) {health-=1;}
if keyboard_check(ord('S')) {health+=1;}
////////////////////////////////////testing buttons
wrap in both directions when an instance moves outside the room

```

**Collision Event with object obj\_asteroid\_parent:****execute code:**

```

///If Can Be Hit is true lose health and reset alarm[1]
if can_be_hit
{
    health-=global.level*2;
    can_be_hit=false;
}

```

```

alarm[1]=room_speed;
scr_voice(choose(snd_voice_im_bleeding,snd_voice_ouch,snd_voice_ow,snd_voice_thats_got_
to_hurt));
effect_create_above(ef_firework,x,y,2,c_red);
}

```

#### Collision Event with object `obj_enemy_bullet`:

##### execute code:

```

//Hit by Bullet
health-=global.level*2;
scr_voice(choose(snd_voice_im_bleeding,snd_voice_ouch,snd_voice_ow,snd_voice_thats_got_to_
hurt))
effect_create_above(ef_firework,x,y,3,c_red);
with (other) instance_destroy();

```

#### Draw Event:

##### execute code:

```

//Draw Selected Ship
draw_sprite_ext(global.weapon_info[4,global.selected_weapon],0,x,y,1,1,image_angle,c_
white,1);

```

#### *obj\_button\_1*

##### Information about object: `obj_button_1`

**Sprite:** `spr_locked_or_not`

**Solid:** `false`

**Visible:** `true`

**Depth:** `0`

**Persistent:** `false`

**Parent:** `obj_lock_parent`

**Children:**

**Mask:**

**No Physics Object**

#### Create Event:

##### execute code:

```

//Set a button ID
my_id=1;

```

#### *obj\_button\_2*

##### Information about object: `obj_button_2`

**Sprite:** `spr_locked_or_not`

**Solid:** `false`

**Visible:** `true`



**Depth: 0**  
**Persistent: false**  
**Parent: obj\_lock\_parent**  
**Children:**  
**Mask:**  
**No Physics Object**

Create Event:

**execute code:**

```
//Set a button ID  
my_id=2;
```

*obj\_button\_3*

**Information about object: obj\_button\_3**  
**Sprite: spr\_locked\_or\_not**  
**Solid: false**  
**Visible: true**  
**Depth: 0**  
**Persistent: false**  
**Parent: obj\_lock\_parent**  
**Children:**  
**Mask:**  
**No Physics Object**

Create Event:

**execute code:**

```
//Set a button ID  
my_id=3;
```

*obj\_button\_4*

**Information about object: obj\_button\_4**  
**Sprite: spr\_locked\_or\_not**  
**Solid: false**  
**Visible: true**  
**Depth: 0**  
**Persistent: false**  
**Parent: obj\_lock\_parent**  
**Children:**  
**Mask:**  
**No Physics Object**

Create Event:

**execute code:**

```
//Set a button ID  
my_id=4;
```

***obj\_menu\_control*****Information about object: obj\_menu\_control****Sprite:****Solid: false****Visible: true****Depth: 0****Persistent: false****Parent:****Children:****Mask:****No Physics Object****Create Event:****execute code:**

```

///Save to INI Save File
save=ini_open("save.ini");
ini_write_real("main", "cash", global.cash);
ini_write_real("main", "health", health);
ini_write_real("main", "lives", lives);
ini_write_real("main", "hits", global.hits);
ini_write_real("main", "shots", global.shots);
ini_write_real("main", "level", global.level);

//weapon 1
ini_write_real("weapon1", "bullets", global.weapon_info[3,1]);
ini_write_real("weapon1", "shots", global.weapon_info[6,1]);

//weapon 2
ini_write_real("weapon2", "bullets", global.weapon_info[3,2]);
ini_write_real("weapon2", "shots", global.weapon_info[6,2]);

//weapon 3
ini_write_real("weapon3", "bullets", global.weapon_info[3,3]);
ini_write_real("weapon3", "shots", global.weapon_info[6,3]);

//weapon 4
ini_write_real("weapon4", "bullets", global.weapon_info[3,4]);
ini_write_real("weapon4", "shots", global.weapon_info[6,4]);

ini_close();
execute code:

///Play Music
if !audio_is_playing(snd_music)
{
    audio_play_sound(snd_music,2,true);
}

```

**Step Event:****execute code:**

```

///Check If Game Complete

if global.level==5
{
    room_goto(room_game_complete);
}

```

**execute code:**

```

///For testing
if keyboard_check_pressed(ord('Q'))
{
    global.level+=1;
}
if keyboard_check_pressed(ord('A'))
{
    global.level-=1;
}

```

**Draw GUI Event:****execute code:**

```

///draw stats and stuff
scr_set_menu_text();
draw_text(20,400,"Lives="+string(lives));
draw_text(400,400,"Health="+string(health));
draw_text(600,400,"Cash="+string(global.cash));
total=0;
for (i = 1; i < 5; i += 1)
{
    draw_text(20,400+(i*40),"Weapon: "+global.weapon_info[0,i]);
    draw_text(400,400+(i*40),"Shots Fired: "+string(global.weapon_info[6,i]));
    total+=global.weapon_info[6,i];
}
draw_text(20,640,"Total Shots Fired: "+string(total));

draw_text(20,700,"Total Hits: "+string(global.hits));

```

**obj\_quit****Information about object: obj\_quit****Sprite:** spr\_quit**Solid:** false**Visible:** true**Depth:** 0**Persistent:** false**Parent:**

**Children:****Mask:****No Physics Object****Alarm Event for alarm 0:****execute code:**

```
game_end();
```

**Step Event:****execute code:**

```

//End Game If Clicked
if (position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left))
{
    scr_voice(snd_voice_thanks_for_playing);
    alarm[0]=room_speed*3;
}

```

***obj\_restart*****Information about object: obj\_restart****Sprite: spr\_restart****Solid: false****Visible: true****Depth: 0****Persistent: false****Parent:****Children:****Mask:****No Physics Object****Step Event:****execute code:**

```

if (position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left))
{
    // delete all stats in ini and restart game
    scr_voice(snd_voice_game_restarted);
    //Save to INI Save File
    if file_exists("save.ini") file_delete("save.ini");
    game_restart();
}

```

**Draw Event:****execute code:**

```

//Draw Self and Message
draw_self();
draw_set_colour(c_white);
draw_set_font(font_mini_message);
draw_set_halign(fa_center);
draw_set_valign(fa_middle);
draw_text(x,y+40,"Clears All#Progress");

```

*obj\_music\_controller***Information about object: obj\_music\_controller****Sprite:****Solid: false****Visible: true****Depth: 0****Persistent: false****Parent:****Children:****Mask:****No Physics Object****Create Event:****execute code:**

```

if !audio_is_playing(snd_music)
{
    audio_play_sound(snd_music,3,true);
    audio_sound_gain(snd_music, 0.2, 0);
}

```

*obj\_game\_over\_voice***Information about object: obj\_game\_over\_voice****Sprite:****Solid: false****Visible: true****Depth: 0****Persistent: false****Parent:****Children:****Mask:****No Physics Object**

**Create Event:****execute code:**

```

//Voice Game Over
scr_voice(snd_voice_game_over);

```

***obj\_message*****Information about object: obj\_message****Sprite:****Solid: false****Visible: true****Depth: -100****Persistent: false****Parent:****Children:****Mask:****No Physics Object****Create Event:****execute code:**

```

//set up
can_show=true;
to_draw="";

```

**Alarm Event for alarm 0:****execute code:**

```

//alarm0 set alarm1
alarm[1]=room_speed;
to_draw="";

```

**Alarm Event for alarm 1:****execute code:**

```

//set as able to show
can_show=true;

```

**Step Event:****execute code:**

```

//check if message waiting
check=ds_list_size(global.message);
if check>0 && can_show
{

```

```

    to_draw=ds_list_find_value(global.message,0);
    ds_list_delete(global.message,0);
    can_show=false;
    alarm[0]=room_speed*2;
}

```

**Draw Event:****execute code:**

```

///draw a message
draw_set_font(font_message);
draw_set_halign(fa_center);
draw_set_valign(fa_middle);
draw_set_colour(c_black);
draw_text(400-2,400-2,to_draw);
draw_set_colour(c_red);
draw_text(400,400,to_draw);

```

***obj\_shop\_button\_1***

Information about object: *obj\_shop\_button\_1*  
 Sprite: *spr\_buy*  
 Solid: *false*  
 Visible: *true*  
 Depth: *0*  
 Persistent: *false*  
 Parent: *obj\_shop\_button\_parent*  
 Children:  
 Mask:  
 No Physics Object

**Create Event:****execute code:**

```

///Set ID
button_id=1;

```

***obj\_shop\_button\_2***

**Information about object: *obj\_shop\_button\_2***  
**Sprite: *spr\_buy***  
**Solid: *false***  
**Visible: *true***  
**Depth: *0***  
**Persistent: *false***  
**Parent: *obj\_shop\_button\_parent***  
**Children:**  
**Mask:**  
**No Physics Object**

Create Event:

**execute code:**

```
///Set ID
button_id=2;
```

***obj\_shop\_button\_3***

**Information about object: obj\_shop\_button\_3**

**Sprite: spr\_buy**

**Solid: false**

**Visible: true**

**Depth: 0**

**Persistent: false**

**Parent: obj\_shop\_button\_parent**

**Children:**

**Mask:**

**No Physics Object**

Create Event:

**execute code:**

```
///Set ID
button_id=3;
```

***obj\_shop\_button\_4***

**Information about object: obj\_shop\_button\_41**

**Sprite: spr\_buy**

**Solid: false**

**Visible: true**

**Depth: 0**

**Persistent: false**

**Parent: obj\_shop\_button\_parent**

**Children:**

**Mask:**

**No Physics Object**

Create Event:

**execute code:**

```
///Set ID
button_id=4;
```

***obj\_play\_shop\_button***

**Information about object: obj\_play\_shop\_button**

**Sprite: spr\_play**



**Solid: false**  
**Visible: true**  
**Depth: 0**  
**Persistent: false**  
**Parent:**  
**Children:**  
**Mask:**  
**No Physics Object**

**Create Event:**

**execute code:**

```
///Create Message Object, display message
scr_voice(snd_voice_shop);
instance_create(10,10,obj_message);
scr_msg("Buy Your##Weapons");
```

**Step Event:**

**execute code:**

```
///Goto Level When Clicked
if (position_meeting(mouse_x, mouse_y, id) && mouse_check_button_released(mb_left))
{
    room_goto(room_level);
}
```

**Draw Event:**

**execute code:**

```
///Draw Text For Shop
draw_self();
scr_shop_set_text();
draw_text(20,20,"Cash: "+string(global.cash));
draw_text(50,40, "Weapon Type");//draw name
draw_text(250,40, "Strength");//strength
draw_text(350,40, "Cost");//cost
draw_text(450,40, "Current");//current ammo
```

***obj\_splash***

**Information about object: obj\_splash**

**Sprite:**  
**Solid: false**  
**Visible: true**  
**Depth: 0**  
**Persistent: false**  
**Parent:**  
**Children:**

**Mask:****No Physics Object****Create Event:****execute code:**

```
///Set Splash Timer
alarm[0]=1*room_speed;
```

**execute code:**

```
///Declare Things Needed in Game
randomize();
//declare messaging system
global.message=ds_list_create();
global.selected_weapon=1;
```

**execute code:**

```
///Load From INI Save File
save=ini_open("save.ini");
global.cash=ini_read_real("main", "cash", 10000);
health=ini_read_real("main", "health", 100);
lives=ini_read_real("main", "lives", 8);
global.hits=ini_read_real("main", "hits", 0);
global.shots=ini_read_real("main", "shots", 0);
global.level=ini_read_real("main", "level", 1);

//weapon 1
global.weapon_info[3,1]=ini_read_real("weapon1", "bullets", 200);
global.weapon_info[6,1]=ini_read_real("weapon1", "shots", 0);

//weapon 2
global.weapon_info[3,2]=ini_read_real("weapon2", "bullets", 250);
global.weapon_info[6,2]=ini_read_real("weapon2", "shots", 0);

//weapon 3
global.weapon_info[3,3]=ini_read_real("weapon3", "bullets", 200);
global.weapon_info[6,3]=ini_read_real("weapon3", "shots", 0);

//weapon 4
global.weapon_info[3,4]=ini_read_real("weapon4", "bullets", 10);
global.weapon_info[6,4]=ini_read_real("weapon4", "shots", 0);

ini_close();
```

**execute code:**

```
///Declare Static Weapon Info
```

```

//Ship Type
global.weapon_info[0,1]="Gun";
global.weapon_info[0,2]="Double Gun";
global.weapon_info[0,3]="Missile";
global.weapon_info[0,4]="Nuke";

//Ship Weapon Strength
global.weapon_info[1,1]=1;
global.weapon_info[1,2]=5;
global.weapon_info[1,3]=250;
global.weapon_info[1,4]=1000;

//Weapon Cost Per Bullet
global.weapon_info[2,1]=1;
global.weapon_info[2,2]=10;
global.weapon_info[2,3]=300;
global.weapon_info[2,4]=5000;

//weapon sprite
global.weapon_info[4,1]=spr_ship_gun;
global.weapon_info[4,2]=spr_ship_double_gun;
global.weapon_info[4,3]=spr_ship_missile;
global.weapon_info[4,4]=spr_ship_nuke;

//weapon firing sound
global.weapon_info[5,1]=snd_gun;
global.weapon_info[5,2]=snd_double_gun;
global.weapon_info[5,3]=snd_missile;
global.weapon_info[5,4]=snd_nuke;

```

#### **Alarm Event for alarm 0:**

##### **execute code:**

```

///Splash Timer Exp - Goto Menu
room_goto(room_menu);

```

#### ***obj\_enemy\_bullet***

**Information about object: *obj\_enemy\_bullet***

**Sprite: *spr\_enemy\_bullet***

**Solid: false**

**Visible: true**

**Depth: 0**

**Persistent: false**

**Parent:**

**Children:**

**Mask:**

**No Physics Object**

**Create Event:****execute code:**

```

//Set Target and start moving
target=scr_target(x,y,obj_ship);
speed=2;

```

**Alarm Event for alarm 0:****execute code:**

```

//Destroy
instance_destroy();

```

***obj\_enemy*****Information about object: obj\_enemy****Sprite: spr\_enemy****Solid: false****Visible: true****Depth: 0****Persistent: false****Parent: obj\_asteroid\_parent****Children:****Mask:****No Physics Object****Create Event:****execute code:**

```

//Start Alarm Bullet
alarm[0]=((room_speed*4)/global.level);

```

**execute code:**

```

//Set Target and start moving
target=scr_target(x,y,obj_ship);
speed=2;

```

**Alarm Event for alarm 0:****execute code:**

```

//Fire Bullet & Reset Alarm
instance_create(x,y,obj_enemy_bullet);
alarm[0]=((room_speed*4)/global.level);

```

**Step Event:****execute code:**

```

///Target & Destroy

if instance_exists(target)
{
    tx = target.x;
    ty = target.y;
    direction = scr_angle_rotate(direction, point_direction(x, y, tx, ty), 5);
    image_angle = direction;
    if my_health<0
    {
        scr_voice(snd_voice_enemy_killed);
        effect_create_above(ef_firework,x,y,2,c_red);
        effect_create_above(ef_flare,x,y,2,c_yellow);
        instance_destroy();
    }
}

```

**Collision Event with object obj\_bullet\_parent:****execute code:**

```

///Hit By Player
effect_create_above(ef_ring,x,y,2,c_white);
scr_bullet_hit();

```

## CHAPTER 35



# Creating a Game – Rooms

The game consists of the following six rooms, with the following order in the resource tree:

- room\_splash\_screen
- room\_menu
- room\_shop
- room\_level
- room\_game\_over
- room\_game\_complete

All rooms have the same dimensions, with a width of 800 and a height of 720.

Room room\_splash\_screen has a single object, **obj\_splash** in it and the background bg\_splash, as shown in Figure 35-1:

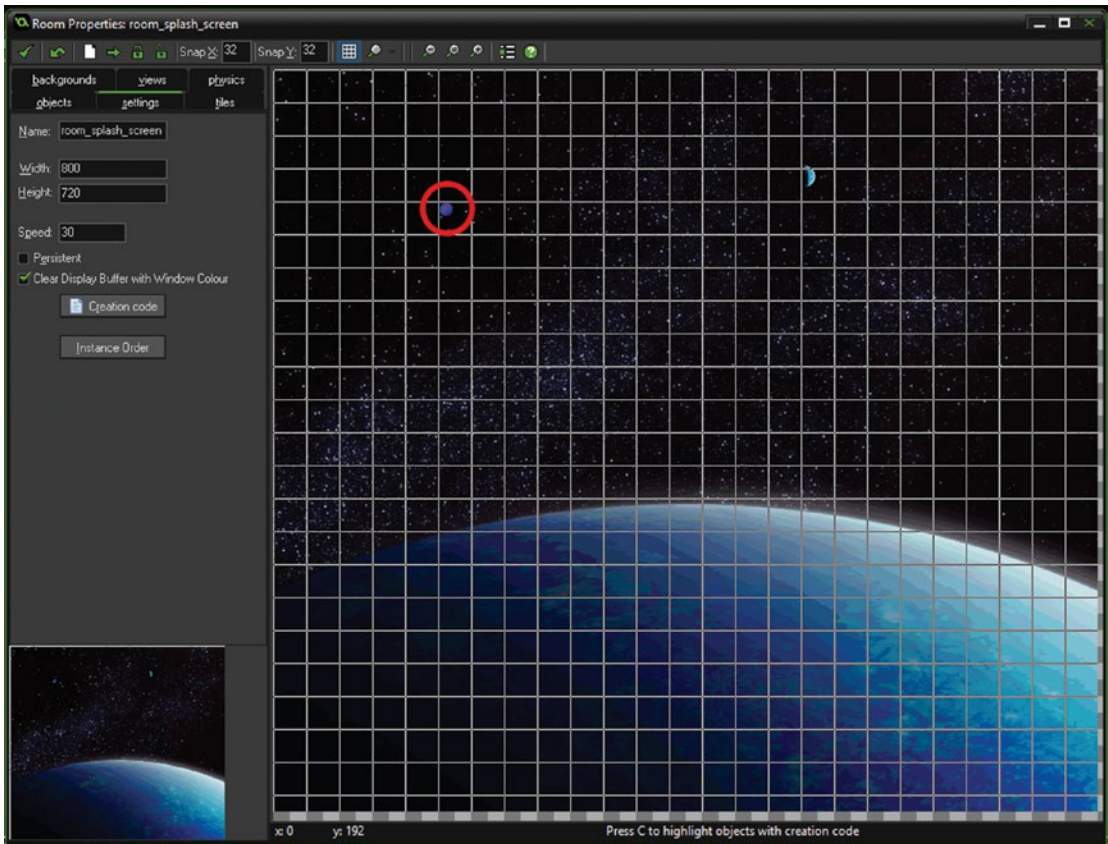


Figure 35-1. Showing *obj\_splash* (circled) placed in room

The next room is **room\_menu**, as shown in Figure 35-2. It contains one each of (from left to right) of **obj\_button\_1**, **obj\_button\_2**, **obj\_button\_3**, **obj\_button\_4**. It also has one instance each of **obj\_restart** and **obj\_quit**. The three circled instances are: **obj\_menu\_control**, **obj\_message**, and **obj\_music\_controller**. The background image set is **bg\_menu**.

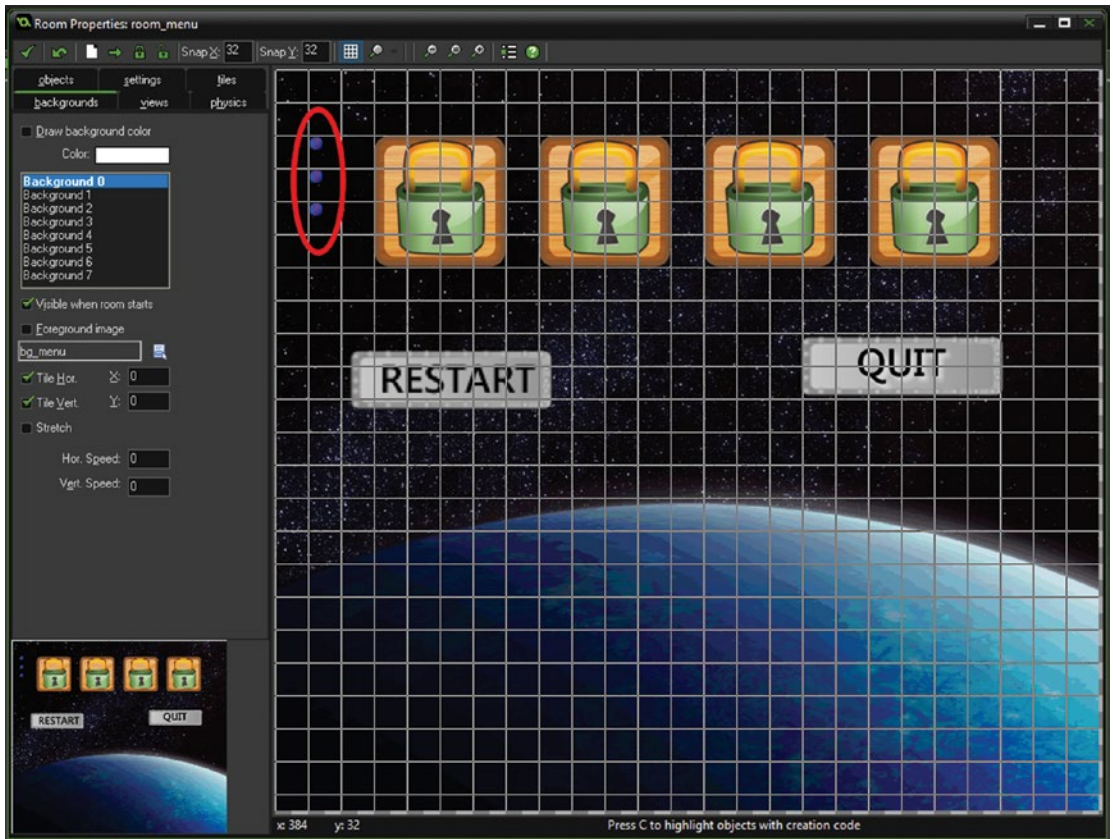


Figure 35-2. Showing object placement in `room_menu`

Figure 35-3 shows the object placement for `room_shop`. It consists (top to bottom) of `obj_shop_button_1`, `obj_shop_button_2`, `obj_shop_button_3`, `obj_shop_button_4` and below them `obj_play_shop_button`. The background is `bg_shop`.



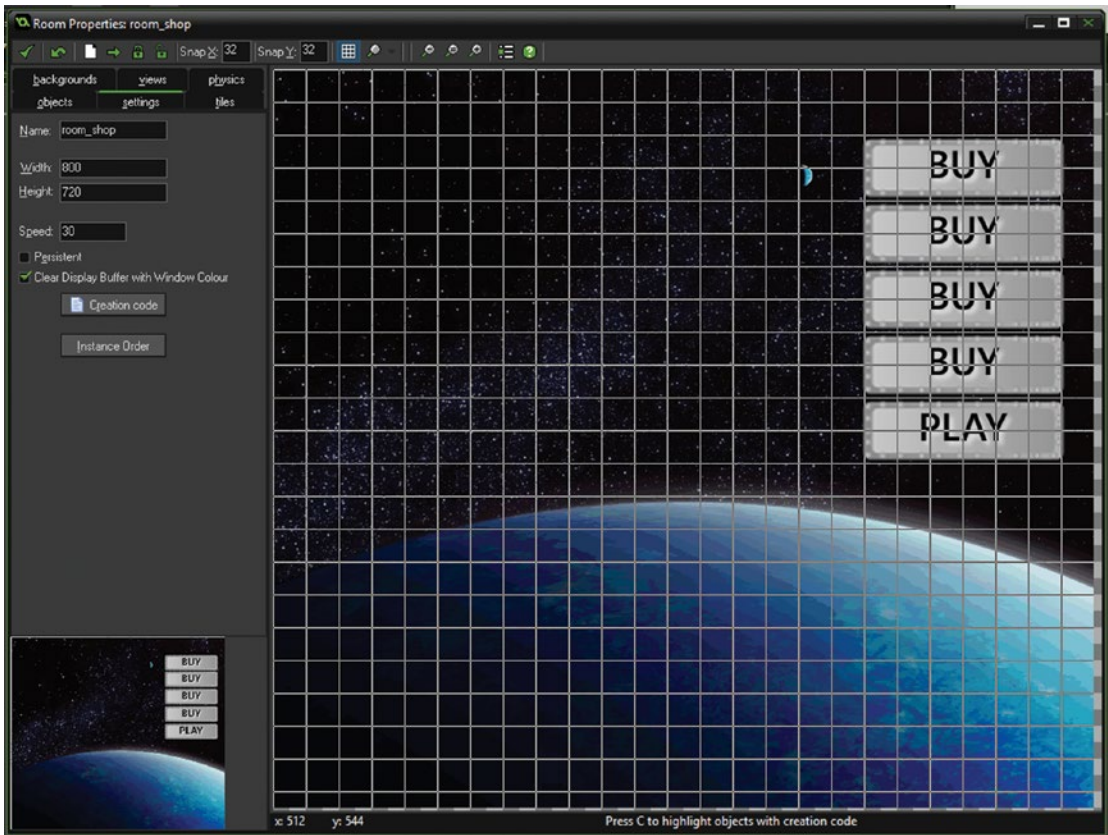
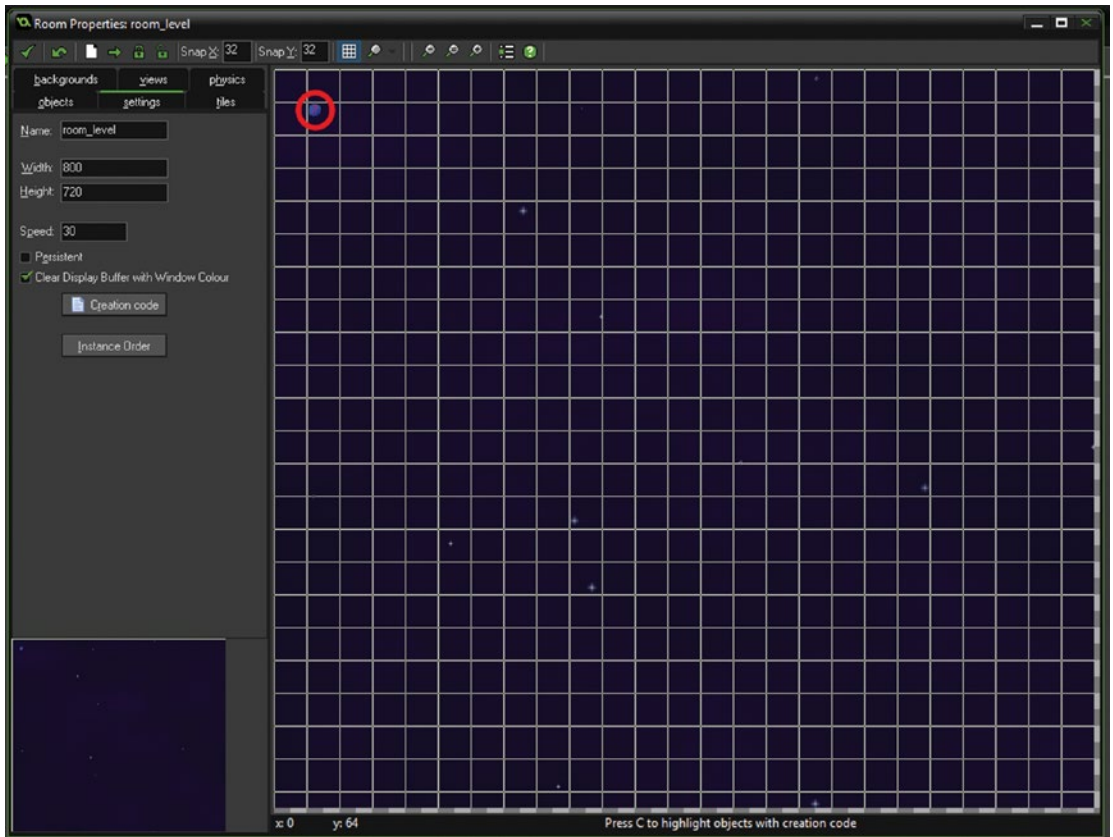


Figure 35-3. Showing room\_shop setup

The room `room_level` has a single object, `obj_level_control_and_hud` and the background `bg_level` set. This is shown in Figure 35-4 below:



**Figure 35-4.** Room `room_level` with an instance of `obj_level_control_and_hud` (circled)

The next room is `room_game_over`. This has the background `bg_game_over` set. It has one instance of `obj_restart` and one of `obj_game_over_voice`. This is shown in Figure 35-5.

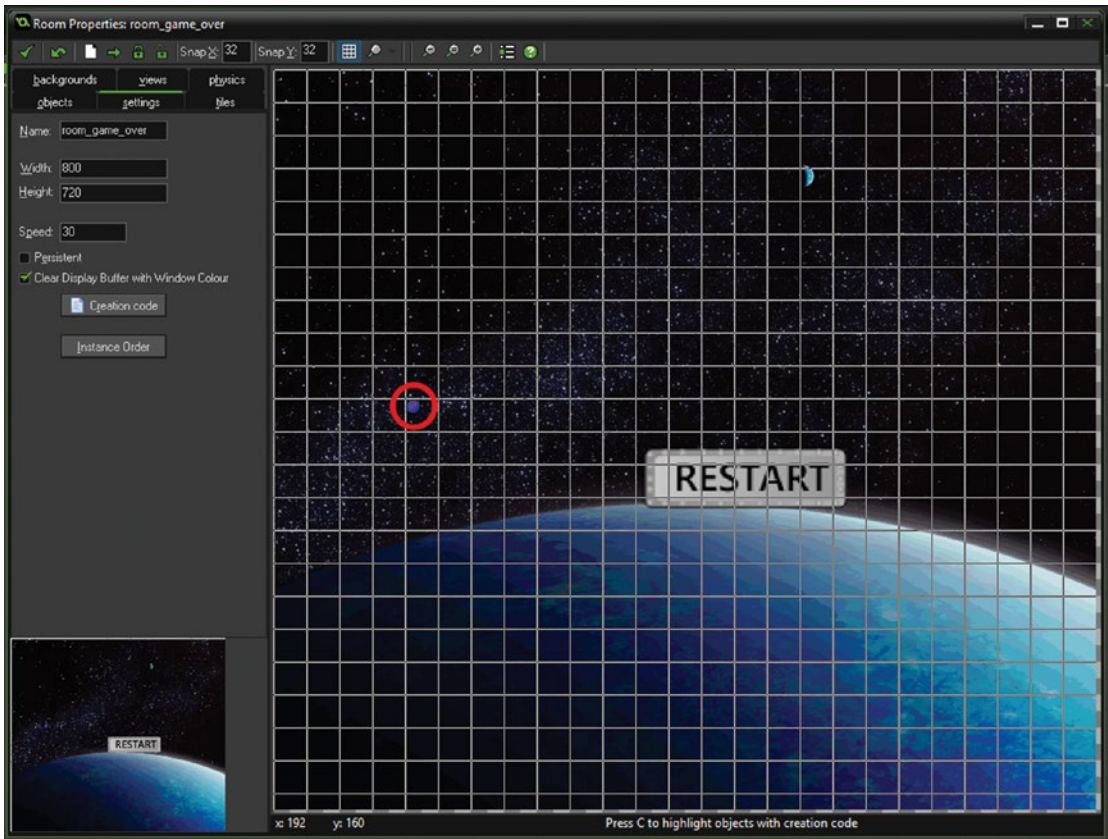
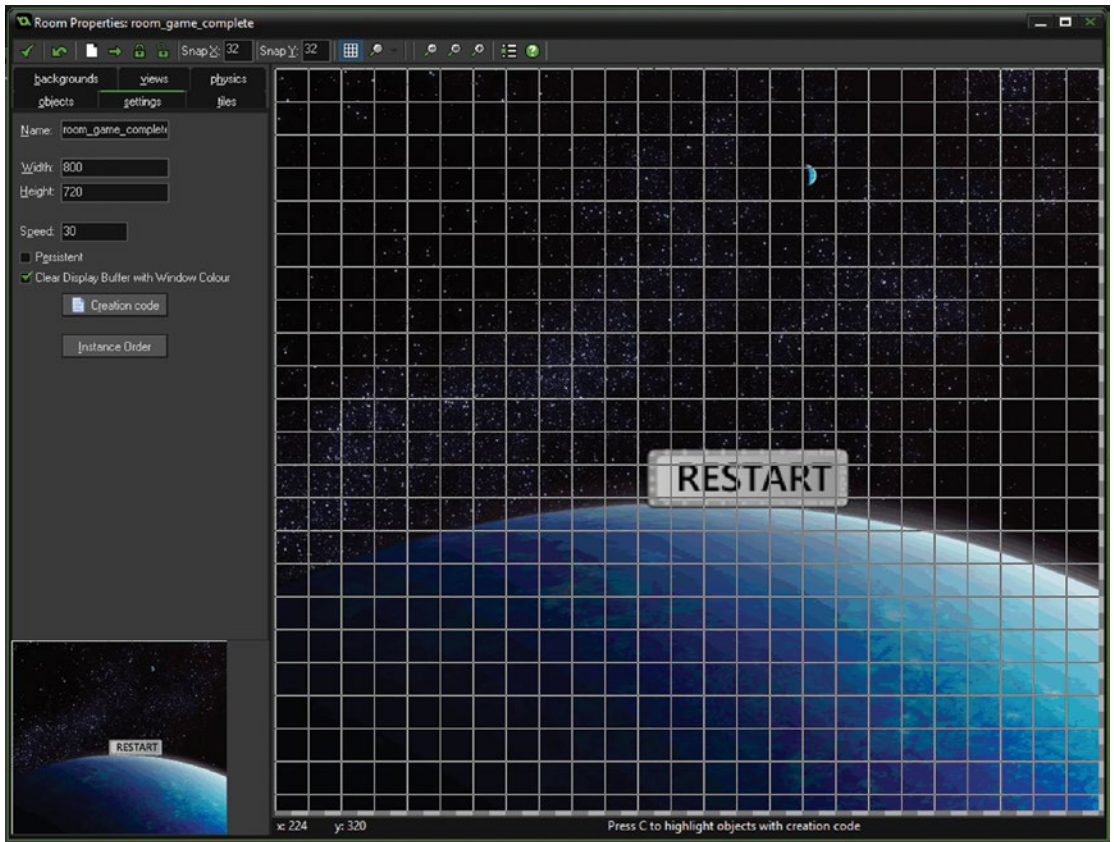


Figure 35-5. Showing placement in room room\_game\_over

Finally room room\_game\_complete has the background **bg\_game\_complete** and a single instance of **obj\_restart**. This is shown in Figure 35-6:



*Figure 35-6. Showing setup of room\_game\_complete*

# CHAPTER 36



## Creating a Game – Progress Sheet

Student Progress Sheet

Student Name: \_\_\_\_\_ Class: \_\_\_\_\_

Project Scores:

Project 1	Project 2	Project 3	Project 4	Project 5
Project 6	Project 7	Project 8	Project 9	Project 10
Project 11	Project 12	Project 13	Project 14	Project 15
Project 16	Project 17	Project 18	Project 19	Project 20
Project 21	Project 22	Project 23	Project 24	Project 25
				Total From 250

Exam Score	From 100
------------	----------

Assignment	From 150
------------	----------

Final Score From 500	
----------------------	--

Grade	
-------	--

Comments
----------

## CHAPTER 37



# Creating a Game – Marking Guide

The following is a suggested marking strategy; you can, of course, use your own marking convention.

Worksheets are used to test your understanding but do not carry any marks.

Each section has a number of projects, worth 10 points in total for each, with scoring detailed for each project. These total 250 points across all the topics.

The final assignment is worth 150 points. A marking strategy for this is provided. Only attempt one final assignment. The final exam is worth 100 points, and a marking guide is provided. Therefore, the maximum you can achieve is 500. You can divide this by 5 to reach a percentage. A suggested final mark is this:

---

400-500	Grade A+	350-400	Grade A	300-350	Grade B+	250-300	Grade B
200-250	Grade C+	150-200	Grade C	100-150	Grade D	50-100	Grade E
0-50	Grade F						

---

## CHAPTER 38



# Creating a Game – End of Projects Assignments

## Endless Runner

Design and create an endless runner type game with the following features:

- Randomly generated objects the player must avoid
- Move player up / down using cursors keys or middle mouse button
- Create coins for player to collect
- Increase **score** for every 10 pixels traveled
- Create a system using INI files that saves / displays the highest score
- Game gets faster for every 1000 points
- GUI that displays distance traveled and coins collected
- Some graphical effects when colliding or collecting bonuses
- A simple menu system that displays highest score and a start game button
- Background music and sound effect when collecting coins

For each element included in the game you receive 10 points

Extra points are awarded for (up to 10 points for each):

- Formatted and commented code, with suitably named variables and assets
- Usage of more advanced GML coding
- Easy player interaction with the game
- Use of graphical effects
- Overall professional finish to the game

In total there are 150 points available.

## Shoot The Ducks

- Create duck objects that fly across the screen
- Make a gun sight movable using mouse
- Create a menu system with 5 unlockable levels
- Create a save system using INI files that stores the current levels currently unlocked and number of coins
- Give player a choice of 4 weapons to choose from, selectable using 1-4 or middle mouse button
- Player earns coins for each duck shot; the closer to the center of the duck, the more points
- Player must have accuracy of at least 90% to unlock next level
- Create a shop where player can use coins to buy weapons
- Create a graphical effect when a duck has been shot
- Use a different background for each level

For each element included in the game you receive 10 points

Extra points are awarded for (up to 10 points for each):

- Formatted and commented code, with suitably named variables and assets
- Usage of more advanced GML coding
- Easy player interaction with the game
- Use of graphical effects
- Overall professional finish to the game

In total there are 150 points available.



# Pontoon

Create a Pontoon(21 or Blackjack) style card where player plays against the banker

- Player starts with 1000 coins and can bet up to 100 per hand
- Create a DS list for the card deck
- Use a separate sprite image for each card
- Deal two cards each to player and banker, showing one of the banker's cards
- Player can hit, stand, double down, or split if two cards are the same. Aces can be worth 1 or 11; make sure to include this
- Banker must hit if 16 or below
- Player wins if higher than banker without going over 21
- Add winnings to player if they win
- Create a save system to save players coins
- If a player hits pontoon (21) take them to bonus room to play high / low to win extra cash

For each element included in the game you receive 10 points

Extra points are awarded for (up to 10 points for each):

- Formatted and commented code, with suitably named variables and assets
- Usage of more advanced GML coding
- Easy player interaction with the game
- Use of graphical effects
- Overall professional finish to the game

In total there are 150 points available.

## Side-Scrolling Shooter

- Create a player controllable using arrow keys and space
- Enemy types should each follow a separate path
- Draw player score, lives, and health using GUI
- Create 5 weapon upgrade types (i.e., missile, side-shooting)
- A collectible that makes player invincible for 30 seconds
- Five unlockable levels, each with an end boss
- Sound effects for each weapon type
- A suitable save system to save player values
- A shop between levels where player can use points to purchase extra weapons
- A save system using INI files

For each element included in the game you receive 10 points

Extra points are awarded for (up to 10 points for each):

- Formatted and commented code, with suitably named variables and assets
- Usage of more advanced GML coding
- Easy player interaction with the game
- Use of graphical effects
- Overall professional finish to the game

In total there are 150 points available.

## End of Project Marking Guide

Choose one option from the previous as an end-of-term assignment.

For each element included in the game you receive 10 points.

Extra points are awarded for (up to 10 points for each):

- Formatted and commented code, with suitably named variables and assets
- Usage of more advanced GML coding
- Easy player interaction with the game
- Use of graphical effects
- Overall professional finish to the game

In total there are 150 points available.

## CHAPTER 39



# End of Project Test

Student's Name: \_\_\_\_\_ Class: \_\_\_\_\_

Date: \_\_\_/\_\_\_/\_\_\_

### Instructions

You have one hour (60 minutes) to complete this paper.

Only start when your proctor tells you to.

Answers should be given in English.

For multiple choice questions, circle the answer you deem to be correct.

Any coding answers should be written in GML.

If you require extra space, there are blank pages at the end of the test paper for this.

There are 50 questions, and each correct answer is worth 2 points.

Access to a computer is allowed, solely for the purpose of accessing GameMaker: Studio's manual / help system.

### Question 1

What type of variable is each of the following?

```
name="Benjamin";
```

```
height=182;
```

### Question 2

What will be drawn on the screen in the following example?

```
name="Benjamin";
```

```
height=182;
```

```
draw_text(20,20, "His name is "+name+". His height is "+string(height));
```

### Question 3

In the following example, which block will be executed, A or B?

```
i=15;
```

```
j=5;
```

```
k=i mod j;
```

```
if k==0
```

```
{
```

```
    //block A
```

```
}
```

```
else
```

```
{
  //block B
}
```

**Question 4**

Write the code that will execute every step that keys Z and up arrow are held down.

**Question 5**

Write the code to draw a solid blue circle at position 100,100 with a radius of 50, with a black border.

**Question 6**

Correct the mistake in the following example:

```
my_colour=rgb_make_colour(40, 80, 70);
```

**Question 7**

Assume a sprite has 3 sub images, 0 1 and 2. Write the code to prevent animation, and set it to show sub image 1.

**Question 8**

Write the code to center both the horizontal and vertical alignment of text.

**Question 9**

Explain what the following code does:

```
if (mouse_check_button_pressed(mb_right))
{
  // do something
}
```

**Question 10**

What does this code do:

```
key=keyboard_lastkey;
```

**Question 11**

When, and how many times is **Create Event** of an object executed?

**Question 12**

When does the following return as true?

```
position_meeting(mouse_x, mouse_y, id)
```

**Question 13**

How would you set an image speed of 0.5?

**Question 14**

What is the default image angle? Which direction is this (left, right, up, or down)?

**Question 15**

Name three variables that have global scope (i.e., can be accessed by any object without having to use `global.` prefix).

**Question 16**

What does the following code do, in a **Draw Event**:

```
lives=5;
for (var i = 0; i < lives; i += 1)
{
  draw_sprite(spr_lives,0,50,50+(50*i));
}
```

**Question 17**

What code would you use to execute code once when the middle mouse button is pressed down?

**Question 18**

What code would you use to set `i` to mouse's `x` position and `j` to the mouse's `y` position?

**Question 19**

What is default game room speed, in steps per second?

**Question 20**

How many alarm events are available per object?

**Question 21**

What does `place_empty(50, 75)` do?

**Question 22**

Explain what the function `collision_line` does.

**Question 23**

What code would use to go to the next room, below the current room in the resource tree?

**Question 24**

Explain the process of setting a 400x400 view that keeps **obj\_example** in the center of the view.

**Question 25**

How would you load background `bg_splash` and set it as the background of room `room_splash`?

**Question 26**

How would you make and use a tiled background that scrolls up 2 pixels for each step?

**Question 27**

Write the code that would check whether a music sound is playing; if it isn't, start it playing without looping.

**Question 28**

What code would you use to pause background music for 5 seconds when `X` is pressed, and then resume it?

**Question 29**

Provide three reasons for using a splash screen.

**Question 30**

Explain the process to only display a new button for a special level, when 3 other levels have been completed.

**Question 31**

What does this code do?

```
i=irandom_range(10,25);
```

**Question 32**

What does `randomize()`; do?

**Question 33**

Explain what the following code does:

```
mp_potential_step( obj_enemy.x, obj_enemy.y, 1, true);
```

**Question 34**

What does the following code do? (assume this is placed in the Step Event of `obj_player`, and there is one instance each of `obj_player` and `obj_enemy` in a room)

```
image_angle=(point_direction(x,y,obj_enemy.x, obj_enemy.y)+180) mod 360;
```

**Question 35**

Using this:

```
lives=22;
```

Assume you have a new INI file and have just executed the following:

```
ini_open("example.ini");
ini_write_string("health", "lives", string(lives));
ini_close();
```

What would the contents of the INI file look like?

**Question 36**

What functions would you use to write and read real values?

**Question 37**

Write the code to create a star effect at the mouse's position when the key T is pressed.

**Question 38**

What is the difference between `effect_create_above` and `effect_create_below`?

**Question 39**

How would you make a block of code run 25 times?

**Question 40**

Explain an example where you would use a for loop:

**Question 41**

Given the following:

```
size[0]=5;
size[1]=3;
size[2]=9;
```

Provide an example to double all values, use a loop of some kind.

**Question 42**

Write the code required to draw the data from question 41 vertically across the room.

**Question 43**

What is the code required to sort a `ds_list` descending?

**Question 44**

How would you replace an entry in a `ds_list`?

**Question 45**

Explain what the path end action `path_action_reverse` does.

**Question 46**

What code would you use to get the Y position of a path's 2nd position?

**Question 47**

Give two reasons for using scripts:

**Question 48**

Write a script that takes in two real numbers and returns the value of the highest.

**Question 49**

How would you search all scripts for references to `spr_example`?

**Question 50**

Explain the difference between `//`, `///`, and `/* */`.

# Test Paper Answers

## Question 1

What type of variable is each of the following?

`name="Benjamin";` **This is a string.**  
`height=182;` **This is a real.**

## Question 2

What will be drawn on the screen in the following example?

```
name="Benjamin";
height=182;
draw_text(20,20, "His name is "+name+". His height is "+string(height));
```

**His name is Benjamin. His height is 182**

## Question 3

In the following example, which block will be executed, A or B?

```
i=15;
j=5;
k=i mod j;
if k==0
{
    //block A
}
else
{
    //block B
}
```

**Block A will be executed.**

## Question 4

Write the code that will execute every step that keys Z and up arrow are held down.

**`if (keyboard_check(ord('Z')) && keyboard_check(vk_up))`**

## Question 5

Write the code to draw a solid blue circle at position 100,100 with a radius of 50, with a black border.

```
draw_set_colour(c_blue);
draw_circle(100,100,50,false);
draw_set_colour(c_black);
draw_circle(100,100,50,true);
```

## Question 6

Correct the mistake in the following example:

```
my_colour=rgb_make_colour(40, 80, 70);
Should be my_colour=make_colour_rgb(40, 80, 70);
```



**Question 7**

Assume a sprite has 3 sub images, 0 1 and 2. Write the code to prevent animation, and set it to show sub image 1.

```
image_speed=0;  
image_index=1;
```

**Question 8**

Write the code to center both the horizontal and vertical alignment of text.

```
draw_set_halign(fa_center);  
draw_set_valign(fa_middle);
```

**Question 9**

Explain what the following code does:

```
if (mouse_check_button_pressed(mb_right))  
{  
    // do something  
}
```

**This will execute the code one time only when the right mouse button is pressed.**

**Question 10**

What does this code do:

```
key=keyboard_lastkey;
```

**It will set key to hold the last key that was pressed.**

**Question 11**

When, and how many times is **Create Event** of an object executed?

**When the object is created in the room.**

**Question 12**

When does the following return as true?

```
position_meeting(mouse_x, mouse_y, id)
```

**This will return as true if the mouse cursor is over the object's sprite (if present), otherwise it will return false.**

**Question 13**

How would you set an image speed of 0.5?

```
image_speed=0.5;
```

**Question 14**

What is the default image angle? Which direction is this (left, right, up, or down)?

**The default image angle is 0, this points to the right.**

**Question 15**

Name three variables that have global scope (i.e., can be accessed by any object without having to use global. prefix):

**They are *health*, *lives*, and *score*.**

**Question 16**

What does the following code do, in a **Draw Event**:

```
lives=5;
for (var i = 0; i < lives; i += 1)
{
    draw_sprite(spr_lives,0,50,50+(50*i));
}
```

**It will draw the sprite spr\_lives 5 times vertically.**

**Question 17**

What code would you use to execute code once when the middle mouse button is pressed down?

```
if mouse_check_button_pressed(mb_middle)
{
    //do something
}
```

**Question 18**

What code would you use to set i to mouse's x position and j to the mouse's y position?

```
i=mouse_x;
j=mouse_y;
```

**Question 19**

What is default game room speed, in steps per second?

**30 frames per second.**

**Question 20**

How many alarms events are available per object?

**There are 12 alarms, 0 through 11.**

**Question 21**

What does place\_empty(50, 75) do?

**Checks whether the instance collides with any other instance at the given position.**

**Question 22**

Explain what the function collision\_line does.

**It will check a collision of an object between two points; if a collision is found it will return its instance, otherwise no one.**

**Question 23**

What code would use to go to the next room, below the current room in the resource tree?

```
room_goto_next()
```

**Question 24**

Explain the process of setting a 400x400 view that keeps **obj\_example** in the center of the view.

**In a room, enable the use of views and set as visible on room start. Set the view and port size. Choose the object to follow and set borders as 200.**

**Question 25**

How would you load background `bg_splash` and set it as the background of room `room_splash`?

**Load and name the background using the load background icon at the top. Open room `room_splash`, set visible on room start, and then choose the loaded background.**

**Question 26**

How would you make and use a tiled background that scrolls up 2 pixels for each step?

**This can be done several ways.**

**You can set the Vert Speed in the room editor's background tab to -2.**

**Or through code on a Step Event of an object that is in the room:**

```
background_y[0]-=2;
```

**Question 27**

Write the code that would check whether a music sound is playing, if it isn't, start it playing without looping.

```
if !audio_is_playing(snd_music)
{
  audio_play_sound(snd_music,1,false);
}
```

**Question 28**

What code would you use to pause background music for 5 seconds when X is pressed, and then resume it?

**Create Event:**

```
audio_play_sound(snd_music,1,true);
```

**Step Event:**

```
if (keyboard_check_pressed(ord('X')))
{
  if audio_is_playing(snd_music)
  {
    audio_pause_sound(snd_music);
    alarm[0]=room_speed*5;
  }
}
```

**Alarm 0 Event:**

```
audio_resume_sound(snd_music);
```

**Question 29**

Provide three reasons for using a splash screen.

**To show company logo / graphics.**

**To load any saved data.**

**Initialize any variables, ds\_lists, etc.**

**Other answers are possible.**

**Question 30**

Explain the process to only display a new button for a special level, when 3 other levels have been completed.

**Check if all levels done (i.e., global.level variable is used to check). In the Draw Event only draw if other levels complete.**

**Make this object clickable and take player to special level when clicked.**

**Question 31**

What does this code do?

```
i=irandom_range(10,25);
```

**It will choose a whole integer between 10 and 25 inclusive.**

**Question 32**

What does `randomize()`; do?

**It will allow random functions to have different results each time the game is played.**

**Question 33**

Explain what the following code does:

```
mp_potential_step( obj_enemy.x, obj_enemy.y, 1, true);
```

**It will move toward obj\_enemy 1 pixel each step, avoiding all objects.**

**Question 34**

What does the following code do? (assume this is placed in the **Step Event** of **obj\_player**, and there is one instance each of **obj\_player** and **obj\_enemy** in a room)

```
image_angle=(point_direction(x,y,obj_enemy.x, obj_enemy.y)+180) mod 360;
```

**It will make obj\_player point away from obj\_enemy.**

**Question 35**

Using this:

```
lives=22;
```

Assume you have a new INI file and have just executed the following:

```
ini_open("example.ini");
ini_write_string("health", "lives", string(lives));
ini_close();
```

What would the contents of the INI file look like?

```
[health]
```

```
lives=22
```

**Question 36**

What functions would you use to write and read real values?

```
ini_read_real
```

```
ini_write_real
```

**Question 37**

Write the code to create a star effect at the mouse's position when the key T is pressed.

```
if (keyboard_check_pressed(ord('T')))
{
    effect_create_above(ef_star,mouse_x,mouse_y,2,c_yellow);
}
```

**Question 38**

What is the difference between `effect_create_above` and `effect_create_below`?

**effect\_create\_above will draw above any other drawing the object has, effect\_create\_below will draw below it.**

**Question 39**

How would you make a block of code run 25 times?

```
repeat (2)
{
    //code here
}
```

**Question 40**

Explain an example where you would use a for loop.

**For example, for getting and drawing data from an array.**

**There are many possible answers to this, so any valid answer is correct.**

**Question 41**

Given the following:

```
size[0]=5;
size[1]=3;
size[2]=9;
```

Provide an example to double all values, use a loop of some kind.

```
for (i = 0; i < 3; i += 1)
{
    size[i]*=2
}
```

**Other valid examples are possible.**

**Question 42**

Write the code required to draw the data from question 41 vertically across the room.

```
for (i = 0; i < 3; i += 1)
{
    draw_text(100+(100*i),50, size[i]);
}
```

**Question 43**

What is the code required to sort a `ds_list` descending?

```
ds_list_sort(ds_list_name, false);
```

**Question 44**

How would you replace an entry in a `ds_list`?

For example:

```
ds_list_replace(ds_list_name, 2, new_value);
```

**Question 45**

Explain what the path end action `path_action_reverse` does.

**When the object following the path reaches the end of the path, it will reverse and follow the path backwards.**

**Question 46**

What code would you use to get the Y position of a path's 2<sup>nd</sup> position?

```
ypoint=path_get_point_y(path_example, 1);
```

**Question 47**

Give two reasons for using scripts:

**For example:**

**Great if same code used in more than one place.**

**Keeps things tidy.**

**Makes changing things quicker and easier,**

**Other answers possible.**

**Question 48**

Write a script that takes in two real numbers and returns the value of the highest.

```
///scr_highest  
if argument0 > argument1 return argument0; else return argument1;
```

**Question 49**

How would you search all scripts for references to `spr_example`?

**Go to Scripts ► Search In Scripts, or press Shift+CTRL+F, and enter `spr_example`.**

**Question 50**

Explain the difference between `//`, `///`, and `/*` `*/`.

**// is a general comment**

**/// can be used to give preview in actions of object's events, and used as auto complete for scripts.**

**/\* \*/ can be used to block out multiple lines**

## CHAPTER 40



# Summary

Now that you have completed this book, you have the basic knowledge of GameMaker: Studio, its IDE and GML coding language.

You can take these skills to start making your own games. When trying GML coding for functions not covered in the book, remember that the GameMaker: Studio help manual is your friend – just press F1 – clear explanations and example usage is shown.

If you're up for a challenge, try the next book in this series, *GameMaker: Studio Programming Challenges*. It's a collection of 100 programming projects to test your GML skills. Each has a challenge outline, useful code to use, and an example solution in GMZ.

# Index

## ■ A

Alarm Event, 41, 69, 70

### Alarms

- Advanced projects, 73
- basic projects, 72
- create event, 70
- room\_speed, 69
- worksheet, 71

### Arrays

- advanced projects, 145
- applications, 137, 140–142
- basic projects, 145
- car\_info, 142–144
- create event of an object, 138
- creation, data array, 140–141
- data, weapons, 145
- draw event, 138
- loop, 144
- one-dimensional, 137
- output, 139
- processing data, 138
- resources, 141
- storing data, 137
- two-dimensional, 137
- values, 138
- variable, 139

### Artificial intelligence (AI)

- advanced projects, 124
- basic projects, 124
- creation, 117
- functions, 117
- obj\_bullet and spr\_bullet, 119–120
- obj\_enemy, 117
- obj\_score, 122
- obj\_star, 121–122
- room layout, 119
- room\_splash, 121

Assets handling, 167–169

Available mouse events, 65

## ■ B

### Background files

- advanced projects, 97
- assets, 181
- basic projects, 97
- colour, 95
- creation, 93
- folder, 181
- image or tiles, 93
- level, 97
- new room creation, 94
- resource tree, 181
- room editor, 93
- rooms, 93
- setting, 94
- Step Event, 95
- uses, 93

## ■ C

Child object, 193

coin strip, 49

Collision Event, 42, 43, 50, 55–56

### Collisions

- adding GML, 78
- adding objects, 78
- advanced projects, 83
- basic projects, 82
- checking, 76
- different image index, 79
- event, 75
- game, 83
- line, 80–81
- motions/deciding actions, 75
- origin, 76
- sprite with two
  - sub images set, 79
- worksheet-answer sheet, 82
  - correction, code, 81



- Conditional statements
  - advanced projects, 13
  - basic projects, 12
  - false sentence, 10
  - ord, 10
  - virtual keycodes, 10
  - worksheet, 11

## ■ D, E

Draw GUI Event, 43

Drawing

- advanced projects, 21, 32
- basic projects, 21, 32
- colour blending, 29
- colour constants, 16
- event creation, 27
- formatting functions, 26
- geometric shapes, 15
- horizontal and vertical
  - alignment settings, 26
- images, 25
- worksheet-drawing, 19, 30

draw\_self(), 29

DS lists

- advanced project, 151
- basic projects, 151
- destroy, 149
- inventory system, 147
- one-dimensional arrays, 147
- outcomes, 148
- removal, value, 148
- RPG type game, 147
- sorting, 148–149
- store and process messages, 152
- values, 149

## ■ F

Fonts, 185

Formatting functions, 26

## ■ G

Game effects

- advanced projects, 132
- advantages, 129
- basic projects, 132
- Collision Event, 132
- combination, 132
- creation, 129, 131
- graphical, 129
- randomness, 129
- test, 129

- trail, 130
- types, 129

Game–end of projects

- awarded, 246
- duck objects, 246
- element, 246
- points, 245
- runner type game, 245

*GameMaker: Studio Programming Challenges*, 261

Game–marking guide, 243

Game–objects

- Alarm Event, 214, 223, 225
- code execution, 200
- Collision Event with object, 205–206, 218–219
- execute code, 205, 207–208, 210–211, 215–218, 221–222, 228–229, 232
- health bar D&D, 213
- images D&D, 212
- information, 199
- lives as image, 212
- obj\_asteroid\_medium, 200
- obj\_asteroid\_parent, 204
- obj\_asteroid\_small, 200
- obj\_button\_3, 220
- obj\_cloud, 201
- obj\_coin, 202
- obj\_double\_gun\_bullet, 203
- obj\_enemy\_bullet, 230
- obj\_game\_over\_voice, 224
- obj\_menu\_control, 221
- obj\_message, 225
- obj\_missile\_bullet, 204
- obj\_music\_controller, 224
- obj\_play\_shop\_button, 227
- obj\_quit, 222
- obj\_restart, 223
- obj\_shop\_button\_1, 226
- obj\_shop\_button\_3, 227
- process, 199
- Step Event, 214
- suggested order, 199

Game paths

- path\_cloud, 183
- path\_coin\_bonus, 184

Game–progress sheet, 241

Game sounds

- assets folders, 179
- loaded and organized, 180

Game sprites

- loaded and organized, 177
- spr\_cloud, 176
- spr\_coin, 174
- sub images, 175

GML coding language, 261

## H

Health, lives, and score  
 advanced projects, 61  
 basic projects, 60  
 Collision Event, 56  
 font\_score, 59  
 global variables, 55  
 health bar, 56  
 score, 58  
 spr\_lives, 58  
 Step Event, 56, 59  
 Worksheet, 59

## I, J

INI files  
 add value, levels, 127  
 advanced project, 128  
 basic projects, 127  
 elements, 125  
 game\_data.ini, 126  
 loading, 128  
 player data, 128  
 project tree, included files, 125  
 reading/writing, 126  
 save and load data, 125  
 store various game, 128  
 values, 126  
 variables, 126–128

## K

Keyboard input and simple movement  
 advanced project, 38  
 basic projects, 38  
 Boolean values, 36  
 draw event, 36  
 Step Event, 35  
 strong distinction, 35  
 Key Release Event, 42

## L

Loops  
 advanced projects, 135  
 basic projects, 135  
 DO function, 134  
 do Loop, 134  
 for loop, 133, 136  
 functions, 133  
 GameMaker Language, 133  
 infinite, 133  
 WHILE, 135

## M, N

Marking strategy, 243  
 Mouse Event options, 39, 65  
 Mouse interaction  
 advanced projects, 67  
 basic projects, 67  
 input, 63  
 logic applies, 63  
 Step Event, 63  
 worksheet, 66

## O

obj\_asteroid\_big, 199  
 obj\_asteroid\_medium, 200  
 obj\_asteroid\_small, 200  
 obj\_cloud, 201  
 obj\_coin, 50  
 obj\_double\_gun\_bullet, 203  
 Objects and events  
 advanced project, 45  
 Alarm Event, 41  
 basic projects, 45  
 Collision Event, 42  
 Create Event, 39  
 Destroy Event, 40  
 Draw Event, 41  
 equivalent event, 40  
 GameMaker, 39  
 Key Events, 41  
 Mouse Events, 39  
 Step Event, 41  
 obj\_gun\_bullet, 203  
 obj\_level\_control\_and\_hud, 206  
 obj\_missile\_bullet, 204  
 obj\_nuke\_bullet, 205–206  
 obj\_player, 42

## P, Q

Parent objects  
 assignment, 193  
 obj\_asteroid\_parent, 195–196  
 obj\_bullet\_parent, 194–195  
 obj\_lock\_parent, 196–197  
 obj\_shop\_button\_parent, 198  
 resource folder, 194  
 Paths  
 advanced projects, 157  
 basic projects, 157  
 closed, 155  
 closed and smooth curve, 154  
 creation, 153

Paths(*cont.*)

- delete(), 155
- editor, 155
- end path actions, 155
- GML, 154
- location, 156
- straight or curved, 155

Pontoon, 246

Project answer paper

- alarms events, 255
- background bg\_splash, 255
- background music, 256
- block, 253
- default image angle, 254
- Draw Event, 254
- ds\_list descending, 258
- function collision, 255, 257
- horizontal and vertical alignment, 254
- INI file, 257
- instance, 257
- integer, 256
- loop, 258
- mistake correction, 253
- mouse button, 254
- mouse cursor, 254
- mouse's position, 257
- mouse's x position, 255
- numbers, 259
- object's events, 259
- path\_action\_reverse, 258
- random functions, 257
- resource tree, 255
- screen, 253
- scripts, 259
- splash screen, 256
- variable, 252
- Vert Speed, 255
- z and up arrow keys, 253

Project marking guide, 248

Projects creation, 169

Project test paper

- bg\_splash, 251
- block, 249
- code, 250–251
- computer, 249
- ds\_list, 252
- function, 251
- horizontal and vertical alignment, 250
- image speed, 250
- INI file, 252
- loop, 252
- mouse's position, 251–252
- music sound, 251
- path\_action\_reverse, 252
- proctor, 249

- screen, 249
- scripts, 252
- solid blue circle, 250
- splash screen, 251
- variables, 249, 250

■ **R**

Random

- advanced project, 116
- applications, 114
- basic projects, 116
- direction and rotational speed, asteroids, 116
- game play, 113
- game replay value, 113
- integer number, 115
- number, 113
- randomize(), 113
- randomness, 113
- Studio, 113
- testing, 113
- values, 113–114

room\_game, 51

Rooms

- advanced project, 91
- backgrounds and views, 89
- basic projects, 91
- bg\_menu, 234
- Create Event, 85
- creation, GameMaker, 85
- editor, 85
- games, 85, 92
- instances, 85
- loading, background, 86
- objects, 85, 88, 92
- obj\_level\_control\_and\_hud (circled), 237
- obj\_splash (circled), 234
- player visible on the screen, 88
- resources folder, 89
- resource tree, 233
- room\_game\_complete, 238–239
- room\_game\_over, 237–238
- room\_menu, 234–235
- room\_shop setup, 236
- setting
  - dimensions, 87
  - setting, background, 87
- worksheet, 90–91

■ **S**

Scripts

- advanced projects, 163
- basic projects, 163
- codes, 163

- Create Event, 160
  - creation, 159
  - draw event, 160, 161
  - GML, 159
  - importing, 187
  - objects, 160
  - processing data, 159
  - resource tree, 187
  - scr\_angle\_rotate, 187
  - scr\_bullet\_hit, 188
  - scr\_buy, 188
  - scr\_cycle, 189
  - scr\_draw\_shop, 189
  - scr\_fading, 189
  - scr\_locked\_or\_unlocked, 189–190
  - scr\_msg, 190
  - scr\_play\_effect, 190
  - scr\_set\_menu\_text, 190
  - scr\_shop\_set\_text, 190
  - scr\_spawn, 190
  - scr\_target, 191
  - scr\_voice, 191
  - sharing code, 159
  - values, 160
- Scripts tricks
  - code handling, 165
  - search, 165–166
- Side-scrolling shooter, 247
- snd\_bounce, 65
- Sounds and music
  - advanced projects, 103
  - asteroid, 103
  - background track, 101
  - basic projects, 103
  - channel priority, 100
  - creation, new sound, 99
  - effects and voices, 99, 103, 104
  - GameMaker, 99
  - informative feedback, 104
  - loading, sound file, 100
  - pause and resume functions, 101
  - resources, 99
  - RPG, 99
  - single track, 103
  - style, 99
  - testing, 101
  - volume, 101

- Splash screens and menu
  - advanced project, 112
  - basic projects, 111
  - company logo or a
    - sponsor’s message, 105
  - creation, 105
  - Draw Event, 107
  - GML, 110
  - images, 110
  - level, 112
  - loading in sub images, 106
  - purposes, 112
  - room order, 106
  - room setup with objects, 109
  - rooms in resource
    - tree, 105–106
  - setting, parent object, 107
  - sound, 105
  - Step Event, 106
- sprite origin set, 48
- Sprites
  - coin strip, 49
  - object and loading, 48–49
  - origin, 47
  - Worksheet-Sprites, 51–52
- spr\_name, 65
- spr\_test, 28
- Step Event, 50, 64
- Student progress sheet, 241

## ■ T, U

- Testing, 167

## ■ V

- Variables
  - advanced project, 7
  - basic projects, 7
  - Book Game Variables, 8
  - control object, 8
  - strings and numbers, 1
- Virtual keycodes, 10

## ■ W, X, Y, Z

- Worksheets, 243