GameMaker Programming By Example

With game development becoming more and more popular, GameMaker: Studio provides modern tools for new and experienced developers alike to rapidly create computer games. More and more developers are taking advantage of its many easy-to-use features in order to create games that meet modern standards.

This book is an excellent resource for developers with any level of experience of GameMaker: Studio. At the start, we’ll provide an overview of the basic use of GameMaker: Studio, and show you how to handle input and collisions in a top-down perspective game. We continue on to showcase its more advanced features via six different example projects. The first example game demonstrates platforming with file I/O, followed by animation, views, and multiplayer networking. The next game illustrates AI and particle systems, while the final one will get you started with the built-in Box2D physics engine. By the end of this book, you will have mastered lots of powerful techniques that can be utilized in various 2D games.

Who this book is written for
If you have at least some basic programming experience of JavaScript or any other C-like languages, then this book will be great for you. No experience beyond that is assumed.

What you will learn from this book
- Understand the GameMaker: Studio interface and tools to quickly create the various assets used in your games
- Translate some of the GameMaker: Studio drag and drop functions to the GameMaker Language
- Create games with random elements for exciting gameplay
- Use the basic GameMaker file I/O and encryption systems
- Utilize the GameMaker networking functions to create multiplayer games
- Give AI routines to your enemies to make challenging gameplay
- Create particle systems to give your game exciting graphics
- Understand the various debugging techniques available in GameMaker: Studio

Community Experience Distilled

Master the development of 2D games by learning to use the powerful GameMaker Language and tools provided by the GameMaker: Studio workspace and engine!
In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 1 'Introduction to GameMaker: Studio'
- A synopsis of the book’s content
- More information on GameMaker Programming By Example
About the Authors

**Brian Christian** is a high school student with years of experience in programming, much of it focused on game development using GameMaker: Studio via self-teaching the GameMaker Language. He has also worked in C++ and JavaScript programming along with some web development. Additionally, he is knowledgeable in topics such as data formats and interpretation, compilers and interpreters, and networking. Other languages he has used include Intel x86 assembly, Lua, Ruby, and Python.

Brian's spark for computer programming began with an online Ruby course from Codecademy. Since then, he has been fascinated with how computers and the different programming languages worked. Since he wanted to know about this, he taught himself C from the second edition of *The C Programming Language* book, followed by the more advanced C++ language.

Wanting to try his hand at making advanced games with a language predominantly intended for it, Brian became invested in the GameMaker Language since he'd already been using the drag and drop functionality, but found it limiting. He spent a lot of time learning how different parts of the language worked and how to use them.
Steven Isaacs has been fascinated with technology since the days of his Apple II Plus computer and 300 baud modem. Tinkering, playing MUDs, MOOs, and visiting BBSs occupied much of his free time. In high school, Steven took a programming course in BASIC and created an adventure game as his final project. Many hours were spent in and outside of school working on this game, and it became somewhat of an obsession. It has become abundantly clear that these activities were incredibly influential in the evolution of his professional life. Steven has been teaching since 1992. In addition to teaching, Steven and his wife Cathy Cheo-Isaacs owned Liberty Corner Computing (LCC), a computer training and gaming center with Paul and Sarah Tarantiles. LCC provided innovative summer camps and after school programs offering young people an opportunity to use technology in creative ways. Courses included programming, website design, graphic design and animation, and game development.

Soon after opening LCC, Steven was hired to bring his innovative ideas to William Annin Middle School in Basking Ridge, NJ. At William Annin, Steven taught a number of computer-related courses. His passion for teaching students to create their own games led to an after school club in game design, then a unit in the Gifted and Talented Program. The success of these programs led Steven to develop a full semester 8th grade class in game design and development, and later, a six-week exploratory course in game design and digital storytelling, which is taken by all seventh graders.

In addition, Steven developed and also teaches an online course in game development with GameMaker for The Virtual High School (http://thevhscollaborative.org/). In his teaching, Steven strives to create an environment that empowers student learning through choice in their learning path. He prefers to think of himself as a co-learner and guide for his students. Steven’s passion for learning is obvious to his students as he is constantly looking at creative ways to solve problems, dabbling with new game development and programming environments, and learning how to play and manage Minecraft servers from his students. Students in Steven’s class have many opportunities to explore topics in greater depth independently. The collaboration here with Brian came from his choice to delve deep into the GameMaker Language to further his learning.
Recently, Steven wrote a chapter for the book, *Teacher Pioneers: Visions from the Edge of the Map*, edited by Caro Williams-Pierce on applying the iterative design process to teaching game design and development. In addition, Steven was an editor on *TeacherCraft: How Teachers Learn to Use MineCraft in Their Classrooms* by Seann Dikkers.
GameMaker: Studio is a game development engine that is easy to learn, yet robust enough to use to create commercial games. Budding developers can use the drag and drop approach to coding or the built-in programming language, Game Maker Language (GML), which will be featured in this book.

What this book covers

Chapter 1, *Introduction to GameMaker: Studio*, introduces you to the GameMaker interface and the basic concepts related to getting started with GameMaker.

Chapter 2, *Your First Game – Escape the Dungeon*, gets you oriented with GameMaker and uses the drag and drop approach to create your first game, a maze/adventure game.

Chapter 3, *Introducing the GameMaker Language*, introduces the GameMaker Language conceptually and then goes through a step-by-step recreation of the Escape the Dungeon game written entirely with code.

Chapter 4, *Fun with Infinity and Gravity – An Endless Platformer*, guides you through the creation of an endless platform game. Through the process, you will work with gravity, speed, random spawning, and further explore collision events.

Chapter 5, *Saving and Loading Data*, expands the functionality of your game by programming GameMaker to save and load data. The chapter will cover saving and loading high score data as well as a custom player keyboard binding configuration.

Chapter 6, *A Multiplayer Sidescrolling Platformer*, expands upon the platform game by adding multiplayer, animation, and Xbox Controller support. You will learn about client/server networking to drastically expand upon the possibilities of what can be accomplished with GameMaker.
Preface

Chapter 7, Programming a Scrolling Shooter, covers scrolling shooters, such as Xevious, which represent a classic genre in gaming. This chapter will guide you through the process of coding your own shooter, including a scrolling background and random obstacles to avoid and enemies to shoot down!

Chapter 8, Introducing the GameMaker: Studio Physics Engine, introduces the built-in GameMaker physics engine, which allows you to create physics-based games. In this chapter, you will program two small game environments based on the physics engine.

Chapter 9, Wrapping Up, addresses error checking and debugging. You will learn about the built-in GameMaker debugging features and strategies to troubleshoot your code and fix errors.
In the 15 years of GameMaker experience between us, we have both found it to be one of the best tools for teaching design as well as the introduction to computer science principles. The drag and drop interface lends beautifully to the understanding of computer science concepts related to object-oriented programming including sequencing, loops, conditional statements, variables, among others. Moving from the drag and drop approach to coding in GameMaker Language (GML) is logical and the transition demystifies coding. It is also important to point out the importance of iterative design and debugging that people learning game development with GameMaker will become overly familiar with through the process.

By definition, a game is a special kind of program that is run inside a loop that repeats as long as the user has not decided to close the game program. This loop contains code that creates objects that each have their own loops to control animation, movement, actions, and logic statements based on various aspects of what is occurring in the game. The objects are run until the time they are removed from the program. The main loop also contains logic statements that run code based on different possible user input values. An example of such logic would test if the user has pressed the spacebar, and if so, the game will modify variables of the player object so that it rises, and then falls at a certain point. In order for the player to understand what they have done in their game, the main loop contains code to push graphics to the screen and audio to the speakers, both of which are affected by what different game objects are doing.
There are many different ways to create games. Some people choose to write them in native C++ code without any sort of specialized Integrated Development Environment (IDE). However, GameMaker: Studio is a collection of tools contained in an IDE that make the game creation process much easier, by providing various tools and code functions that are specialized for creating games. For example, there is a visual object editor so that the developer doesn't have to define every single property of objects in code, but rather they can select various boxes that define object properties. Objects control nearly every aspect of a GameMaker game. Rather than using an external spritesheet file containing separate frames of animation, GameMaker contains a sprite editor where each sprite in itself is a special type of object that contains the images. All the developer has to do is to set the sprite of an object to the main sprite, which will then provide access to all of the images inside this sprite. The main game loop is run in rooms, where all of the objects are placed, in turn creating the full game. At compile time, everything produced in the project is converted into C++ code, then compiled, so in the end, the developer has essentially written their game in C++, but by using a much easier method.

We will begin with an overview of the different versions of GameMaker: Studio that are available for download. This will be followed by a guide to the User Interface (UI) of the IDE to orientate you to working in the GameMaker environment. In this simple example, you will learn about creating resources (various kinds of assets) and their purposes, naming conventions, and some drag and drop coding among other skills. The IDE is essentially a collection of integrated editing tools used in a programming environment.

GameMaker: Studio is an application used by newcomers to game design and by experienced developers alike, for both personal and commercial purposes:

- This contains a very clean interface including a fully functional drag and drop programming interface making it very easy to get started creating games. Experienced GameMaker developers typically choose to create games using the text-based coding functionality of the built-in GML.

- This also contains many high-level functions and components for a variety of things that developers might need done in their game and for a variety of platforms. Examples include networking, Steamworks™ SDK support, in-app purchases, and more for platforms such as Microsoft Windows™, Apple Mac OS X™, Linux™ in terms of desktop and laptops, and Google Android™, and Apple iOS for mobile. Many more platforms are available for use with GameMaker: Studio.

- GameMaker: Studio has been used to create many commercial games, a varying selection of which you can view at http://yoyogames.com/showcase.
Choosing your version

GameMaker: Studio is available for download exclusively for Microsoft Windows at http://yoyogames.com/studio. There are a few different versions available, each adding more functionality as the price point increases. All of the versions allow you to create games using the game engine. For the most part, the increased functionality relates to publishing your game to different platforms as well as providing some additional features.

The Standard Edition is fully functioning and is perfectly adequate for developing games in GameMaker: Studio. The Professional Edition includes many more features than the Standard, such as team features, early access builds, the ability to purchase additional export modules, and allows the developer to publish games to platforms beyond Windows and GameMaker: Player. The Master Collection contains everything available, which is the Professional Edition bundled with all the export modules. The additional export modules cannot be purchased with only the Standard Edition of GameMaker: Studio.

Whatever version you choose, keep in mind its limitations and what you might need, but do not pay for more than you need. You might purchase the Master Collection and end up only using the additional HTML5 module, making the additional features unnecessary. However, keep in mind that you can commercially publish your games with any version of the IDE, but the Standard Edition will force a splash screen and application name that both say "Made with GameMaker: Studio".

You can always upgrade later, so the Standard version is a perfectly appropriate place to start. Once you've chosen your version, get a license key for it and download the GameMaker: Studio installer and run it.

The following is a note about GameMaker: Studio licenses. The IDE will revalidate its license key with the YoYo Games servers once a month. However, if it cannot connect to them, you must manually relicense it with a special license file. Read more about this at http://help.yoyogames.com/entries/27068613-Offline-Licence-Verification-for-GameMaker.
The GameMaker: Studio interface

Now that you've completely installed GameMaker: Studio, let's begin learning some of the basics of GameMaker: Studio. Create a new game project by selecting the **New** tab at the top (after launching it), and choosing your project directory, as well as a name for this, we recommend **testGame** or something similar. Once your project is created, you should see a window, the GameMaker: Studio IDE, that looks quite similar to the one shown in the following screenshot:

![GameMaker Studio IDE](image)

The left-hand side of the IDE consists of your resources, which is where all your sounds, sprites, rooms, scripts, and other resources can be found. Clicking on it simply selects them, and double-clicking will uncollapse folders and open up the properties of resources. Right-clicking on any of those folders will open a menu allowing you to create a new instance of the resource type. For example, right-clicking on the **Sprites** folder would give you the option to create a new sprite that you can edit and use in your game. If you right-click on **Scripts**, you can create a new script. Right-clicking on any of those folders also allows the creation of a new group, which is a subfolder strictly for organizational purposes. If a resource or group is right-clicked on, many more options will be available, such as renaming, editing properties, and so on. At the bottom of the resource tree are the **Game Information** and **Global Game Settings** options. The former is like a small notepad that you as the developer can write small notes or reminders for yourself in, but it is not packaged with the game.
Chapter 2

The latter, the **Global Game Settings** option, will allow the editing of many properties of the game. When working with the Professional Edition or additional modules it contains, many additional settings that can be edited for specific targets and SDKs.

Along the top are various buttons, some of which are for creating resources (an alternative to right-clicking the folders on the left of the IDE) and others for game compilation options:

![Compilation and Debugging Options](image)

The green arrow is for regular compilation and testing, while the red one plays the game in debug mode with a debugger attached so as to view variables and other properties of the game as it runs to help you locate problems in your game. We'll teach you more about the debugger in *Chapter 9, Wrapping Up*. The brush is for cleaning the target, which comes in handy when you want to see if a glitch was caused by compilation issues or a fault in your code. The hot keys for these three buttons are F5, F6, and F7, respectively.

Everything from the green Pac-Man to the white rectangle and inclusive, are the buttons for creating resources. The next button is a button to access the **Global Game Settings**, mentioned previously. You can always hover your mouse over any of the buttons to see what they do.

The drop-down list all the way at the end of the top bar allows you to choose your target (or the OS for which your game should be compiled), which will vary based on what modules you may or may not have purchased. All will show up in the list regardless of what you bought, but Microsoft Windows and GameMaker: Player are the only ones that will actually work if you use the Standard Edition of GameMaker: Studio.

Additional keyboard shortcuts that you will find handy are Ctrl + S for saving, Ctrl + N to create a new project, and Ctrl + O for opening an existing project.
GameMaker: Studio documentation

One of the most important buttons on that bar is the question mark surrounded by a circle. Clicking on it opens the complete GameMaker documentation in a new window. It is a comprehensive reference for everything about the IDE, its functionality, drag and drop coding, GML coding, and all other aspects of GameMaker: Studio. It is very useful, and you will likely reference it often. This documentation can also be accessed at http://docs.yoyogames.com/. Additionally, when working in your code editor, if you put your cursor on one of GameMaker's built-in functions or variables, and click your middle mouse button, the documentation page for it will open so you can learn more about how to use it.

An example project

Now that you've got everything set up and you know some of your way around the IDE, let's start with an example game to get started with using GameMaker and to make sure everything is working correctly. You'll first want to create a new sprite, either by right-clicking on the Sprites folder and selecting Create Sprite or clicking on the Pac-Man symbol located on the top bar. This will open a new window that allows you to edit all of the properties of your new sprite. We're going to make a simple sprite that looks like a square.

Downloading the example code

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

Naming convention – resource prefixes

Start with naming it, but don't just name it testSprite or something like that. There are some very important points to be made in naming conventions of resources. It is a good idea to choose names that are related to the resources you are naming. The most important consideration is that your naming should be consistent regardless of the naming convention you choose. If you start using camelCase, always use it. camelCase is where the first letter of each name is lowercase, but every next word in the name has a capital first letter, for example, myLastName or playerHorizontalSpeed. If you always use underscores for naming (and this might be easier since GameMaker's built-in variables and functions use underscores for naming, an example of this would be the variable image_index), then always use underscores.
Also, you must never have spaces in any of your names because in coding, this would cause GameMaker to think they are two separate names and your game will not function properly. It is also good practice to follow and incorporate the resource prefixes. The GameMaker compiler does not recognize prefixes; they are simply part of the name, but all of your resources must have different names, as resource types are not distinguished by the compiler. Using prefixes is an easy way to group your resources though (so using `spr_player` and `obj_player` is better than `player` and `player1`).

Using the prefixes is also very helpful for you and any other programmers, since using them allows you to easily distinguish resource types. The resource prefixes are what help to distinguish between a sprite named `test` and an object named `test`. Also, make sure that you use meaningful names so that you can easily find what you’re looking for or figure out what a resource is just by name. Following is a chart of the resource types and suggested prefixes. Eventually, you will have many resources in your games and will need to easily distinguish one from another.

<table>
<thead>
<tr>
<th>Resource type</th>
<th>Acceptable prefix(es)</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprites</td>
<td><code>spr</code> or <code>s</code></td>
<td><code>spr_test</code> or <code>s_test</code></td>
</tr>
<tr>
<td>Sounds</td>
<td><code>snd</code></td>
<td><code>snd_test</code></td>
</tr>
<tr>
<td>Backgrounds</td>
<td><code>bg</code></td>
<td><code>bg_test</code></td>
</tr>
<tr>
<td>Scripts</td>
<td><code>scr</code></td>
<td><code>scr_test</code></td>
</tr>
<tr>
<td>Fonts</td>
<td><code>fnt</code></td>
<td><code>fnt_test</code></td>
</tr>
<tr>
<td>Objects</td>
<td><code>obj</code> or <code>o</code></td>
<td><code>obj_test</code> or <code>o_test</code></td>
</tr>
<tr>
<td>Rooms</td>
<td><code>rm</code> or <code>lvl</code></td>
<td><code>rm_test</code> or <code>lvl_test</code></td>
</tr>
</tbody>
</table>

We recommend you use these prefixes when naming your resources, as they are universal and will help you to stay consistent in your naming. This is not a complete list of resources and prefixes but this table should give you a good sense of the naming convention.
Drawing the sprite

It's time to get started with sprites! Create your first sprite by right-clicking the Sprites folder and choosing Create Sprite or clicking on the Pac-Man icon in the toolbar. As discussed, we suggest naming your sprite spr_square or something similar, so enter that in the Name field. Next, you can select either the Load Sprite or Edit Sprite options. The first allows you to select an image that you already have saved on your computer, so select that if you already have an image of a square premade or prefer to use external programs for editing your sprites. GameMaker supports a wide array of image types including .bmp, .gif, .jpg, .png, and .swf. We're going to go through the built-in sprite editor, but in time, you will probably be using both approaches of sprite editing.

To start making your own images, you'll first have to click on Edit Sprite. Right now, the only thing you need to know how to do is to make a new subimage (more on these later) inside of your sprite. Click on the white dog-eared paper next to the green checkmark (which is the equivalent of OK in all windows and options in GameMaker: Studio). It will ask you what you want the dimensions of the image to be, which carry across all subimages in the sprite. Leave the 32 by 32 default alone for now—we're just making a simple square:
Now double-click on the new subimage that was created, which should be labeled as **image 0**, as GameMaker: Studio uses **zero-based indexing** (where the first entry of something, in this case subimages in a sprite, is entry 0), and the sprite editor will open, which should look like this:

![Image Editor](image.png)

The left-hand side of the editor consists of your tools and dot sizes, while the right-hand side is your color picker. Two colors can be held at a time, one corresponding to each mouse button (left and right). As far as the tools, the most important ones would be the pencil for basic drawing, the eraser for erasing, the selection tool to select an area for manipulation, and the fill option to fill in an area with a single color. Hovering over any of these tools tells you their function and hotkey. We're only going to use one of these tools right now. Start making your square by selecting any color from the color picker. Next, use the fill option to fill in the entire grid with your selected color and then click on the green checkmark three times (in the three separate windows of course) to completely close out of the sprite and return to the main IDE.
Creating an object

Now that our sprite is finished up, we’re going to create a basic object to put in our game. It is worth pointing out an important distinction here. Sprites are simply graphics to be used in your game while objects are programmed with events and actions in order to function in the game. Without an object, the game is just going to be an unchanging screen. Objects are what control and do everything in the game. Start with creating your object, named obj_square with your method of choice. The following window should appear:

To assign a sprite to your object (which is not always necessary, in fact many objects in your games will not use sprites) select the blue button in the box labeled Sprite and pick the sprite we previously created, spr_square. Don’t worry about anything else you see on the left sidebar, we will get into this in more detail later. For now, focus your attention on the first large box, labeled Events. Events are simply things that happen in your game, such as player input or when an instance of an object is created, and they are handled similarly to if statements, and run the associated Action(s). Actions are the same as statements (what should happen based on the event). So one event might be a keypress of the space key, and when that occurs, the player should jump. In pseudo-code for that event, you might say, if the space key is pressed, then alter the player's y coordinate so that they rise. Then, once they've reached a certain point, change the y coordinate so that the player falls. Right now, we shall be putting an event into our game. Select the Add Event button at the bottom, select the Draw button, followed by the Draw menu option. This is a kind of event that is run over and over for each instance until the instance is destroyed (removed from the game), and is used for rendering. You can display whatever you want: shapes, text, sprites, and much more.
Another event, the **Step** event, is similar to this (it runs over and over) but it does not control rendering. You shouldn't use the **Draw** event as a replacement **Step** event though we use both together (or just one of them if you only need one of them).

Now focus on the right sidebar. This is where all of your available **Actions** are, and whatever you drag into the **Actions** pane will be run when the associated event occurs. Select the **draw** tab at the bottom of the list of tabs on the right and then drag the **Draw Self** box (the first action) into the **Actions** window. This will draw in the room the sprite of the instance of the object that is calling the function. You do not always need to use the function, but we do need it now because we have manually included a **Draw** event. If we don't include it, then the square won't be displayed; only whatever we tell it to display in the next step will be displayed. If you manually include a **Draw** event in your object, it will not draw its sprite and will only draw what you tell it to draw inside this event. So if we don't tell the object to draw itself in this event, you will see no squares drawn. Now, drag the **Draw Text** box, the fourth option or the first in the second row, into the **Actions** box:

![Draw Text window](image)

This should open up a box that looks like the one in the preceding screenshot where you can edit the arguments that you pass to the **Draw Text** function, which include, in this case, the text itself and the $x/y$ coordinates where you want to draw the text. Edit the **text** field so that it holds, `Hello, World!` (no quotes) but do not enter anything into the $x$ and $y$ fields yet—there is something unexpected about these.
Coordinate planes in GameMaker

Typically, on a coordinate plane, your $x$ value would increase as you move right and your $y$ value would increase as you move up. This stays true for $x$, but not for $y$. For $y$, the value will increase as you move down, meaning that the fourth quadrant, which usually holds negative $y$ values, will hold positive $y$ values, and the first quadrant, usually holding positive $y$ values, will hold negative $y$ values. This is a very important note to remember when thinking about where things will be placed in your game. Although, do know that the room only exists in your fourth quadrant, and so what is shown on your screen is also only the fourth quadrant. However, everything can be placed anywhere, so you can actually place objects and whatever you draw at negative coordinates, but of course, either none or only a portion of whatever you have placed will be shown in the game window. A third (or $z$) axis does exist; it is called depth and refers to the placement of objects in your game in terms of being above and below each other. You will learn about this in depth in the next chapter.

Here is a simple coordinate plane to explain this:

```
<table>
<thead>
<tr>
<th>Quadrant II</th>
<th>Quadrant I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negative x values</td>
<td>Positive x values</td>
</tr>
<tr>
<td></td>
<td>Room encompasses this quadrant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quadrant III</th>
<th>Quadrant IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive y values</td>
<td></td>
</tr>
</tbody>
</table>

Now that you know this, we can resume passing arguments to the Draw Text function. For this example, we are going to have a room with dimensions of 1024 by 768 pixels, and we want this message to appear in the middle of the room, so we're going to pass an $x$ value of 512 and a $y$ value of 384 to the function. Do not select the Relative checkbox, as what this would do is have the message appear at (512, 348) relative to the origin of the square object, which, unless we put the object exactly in the top left corner of the room (which we won't), will have the message display somewhere not in the middle. Select the OK button twice to get back to the main IDE.
Creating a room

We’re now going to create one more final resource, and that is a room. Without a room, the game will not compile, so you must always have at least one room, even if it is completely empty. A room is where your game takes place and where your objects are placed. While your objects control the mechanics of your game, they have to be placed in a room for them to actually be used. Create a room with your method of choice and name it rm_main or something similar. Many games contain sandbox or testing rooms that the player wouldn’t have access to, and are just for testing various parts of your game. Even though we only have one object in our game and aren’t really testing anything, you can think of this room as a testing room:

The default dimensions, which we will be using, are already set, as is the room speed, or how many times per second each object in the room runs its code. The room speed of 30 is perfectly fine for us right now. Select the objects tab on the left sidebar, and the only object we have, obj_square, will be already assigned to your left mouse button. Click anywhere in the room to place this object, and place more than one. For this example, 10 would be a good number. Click on the green checkmark when you are done, and you will have finished creating your first game in GameMaker: Studio.
Testing your game

Only one step is left, and that is compiling and running your game. Press F5 on your keyboard or click on the green arrow on the top bar. This will begin the compilation process and then run your game. If everything worked correctly, then you should see a game window that looks similar to this, but with the squares wherever you placed them and in whatever color you chose for the sprite:

If your game window doesn't look like this, then check whether you followed all the instructions correctly (such as choosing the right event and the right parameters for your Draw Text action). You might want to check it with the sample code included with this book to ensure that everything is correct. If that doesn't work, perform a clean build with F7 to make sure your next compilation will start completely fresh so that any compiler-induced errors will not occur. If both of those failed to help you, look at the compiler window at the bottom to see if it tells you anything about errors:
If none of these options work, then you might need to perform a fresh install of GameMaker: Studio. YoYo Games has provided an excellent guide for this, which is located at http://help.yoyogames.com/entries/37903916-How-to-perform-a-fresh-install.

What's happening here in the game is that one instance of the message Hello, World! and one instance of obj_square are being created for each instance of the object that you placed in the room. The top-left corner of the message is where the coordinates we put in earlier are applied, which is why the message isn't precisely in the center of the room. The reason that you only see one instance of that message rather than many is because we told all the objects to display a message at that exact location. In turn, all of these instances of the message are actually overlapping directly on top of one another.
Summary

Now that you have finished the chapter, you have a good understanding of the GameMaker: Studio interface. You have learned what many of the buttons on the main interface do; how to create and name resources including sprites, objects, and rooms; how to add events and actions to your objects and assign them sprites; how rooms and their coordinates work; and how to compile your games.

Review questions

It's always good to be asked questions so that you can be sure you comprehend everything you have read; so, here are some review questions to go over. If you can't figure one out, that's a sign that you could benefit from more reviewing of the chapter:

- What are examples of common naming conventions used for the various resources available in GameMaker: Studio?
- How do you open the documentation for GameMaker: Studio and what is it useful for?
- What is the difference between the green play button and the red play button when it comes to compiling and running your game?
- How do coordinate planes in rooms here vary from the Cartesian coordinate system (regular mathematics)?
- What are events and actions and how are they related?

Quick drills

It's also good for comprehension to apply the skills you learned in different ways, so here are some little challenges that you can try out to make sure you understand how to perform the tasks you learned:

1. Edit the sprite in your spr_square and play around with some of the other tools in the editor to modify your original image (your final outcome doesn't even have to be a square); just rename the sprite to reflect this, as with your object.

2. Play around with some of the other actions on your obj_square to make it do more rather than be displayed and print text. Try something and immediately test it. It will become good practice to test your games often in order to discover errors where they originate. If you add ten features and discover a bug, you don't know where it came from—but if you add one and get a bug, you know where your problem is.
3. Create a new project named `testGame2` or something like that. It will be a clone of the original example, but you're not going to reread the instructions in the chapter or read them from your example game. You're instead going to see if you can recall how to do everything from memory. If you can't recall something, review the chapter to make sure you are comfortable with the content.

Remember that all of these drills and questions are for your comprehension and you're good, so do them—it only helps. With that, this chapter concludes and hopefully, you learned a lot about using GameMaker: Studio. The next chapter will cover the creation of a more complex game with which the player interacts. It will be an escape the dungeon game from which you will learn all sorts of components of programming games in GameMaker: Studio, much more so than you already know about.

An extra note that you might find helpful—GameMaker saves backups of your game by default in your `Documents\GameMaker\Backups\<project name>` folder. You might find that it saves them far too often and does not save enough, so under the **File** option in the IDE's top bar, select **Preferences**, and the left pane will allow you to specify some backup-related options (frequency, location, and amount). Backups are useful if you accidentally break your game and need to go back to a point where everything worked. It might be helpful to make a copy of a working project alongside using the built-in backup system.
Where to buy this book
You can buy GameMaker Programming By Example from the Packt Publishing website. Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.