
INTRODUCTION
TO
GAME DESIGN
&
PROGRAMMING
FROM ZERO TO HERO
BEN TYERS
IN
GAMEMAKER
STUDIO 2

Introduction To Game Design
&
Programming
In
GameMaker Studio 2

©2019 Ben Tyers

LearnGameMakerStudio.com

Special Thanks to The Following, Who Pre-Ordered This Project & Made It Possible:

Michał Kamiński

Corey Cuhay

Honey

Pedro Santos

Mark Porter

Dean Radcliffe

Mickey Everett

Vasco

Mike Cowel

Gaven Renwick

Thanks Also to The Following People:

Yellow Afterlife – Thanks for your help

Nathan Brown

Loukas Bozikis

Alesia Buonomo

Kehran Carr

Arik Chadima

Rom Haviv

Zachary Helm

Introduction To Game Design & Programming

ISBN: 9781795199537
Copyright 2019 © Ben Tyers
First Edition

If you find any issues of problems with this book (such as omissions or mistakes) please drop me an email:

Ben@LearnGameMakerStudio.com

Educational Use

I am more than happy the this or material from it being used in an educational setting, such as schools or clubs. As an educator, I am sure you appreciate how much effort and time goes into making a book such as this, therefore I ask that one copy is purchased (ebook or paperback) for every 10 students using it. If you have any questions, please email:

Ben@LearnGameMakerStudio.com

ACKNOWLEDGMENTS

Graphics In Main Chapters: GameDeveloperStudio.com

All Audio In Main Chapters: SoundImage.org

Assets are not needed to enjoy this book

If you wish to make the game covered in this book, you can access assets from the above sites.

No assets are included with this book project, except for Chapter 7 Introduction, which is optional, and the appendix.

Chapter 7 Introduction Project:

Buttons: DaButtonFactory.com

Heart: OpenGameArt.org cdgramos cc0

Monster: OpenGameArt.org bevouliin.com cc0

Appendix 4 Cloud BananaOwl / opengameart.org CC-BY 3.0

Appendix 4 Gun sight Lucian Pavel / opengameart.org CC0

Appendix 6 Brick and ball Zealex / opengameart.org CC0

Appendix 7 Rotating coin Puddin / opengameart.org CCO

Appendix 7 Character rileygombart / opengameart.org

Appendix 7 Horse reivaxcorp / opengameart.org CC-BY 3.0

Appendix 9 Audio http://soundimage.org

Appendix 12 Songs http://soundimage.org

Appendix 13 Car sheikh_tuhin!

Rock - Jasper / OpenGameArt.org CC0

Appendix 15 Bird bevouliin.com / OpenGameArt.org CC0

Appendix 20 Chess Sprites: mr0.0nerd : https://2dartforgames.wordpress.com/

Appendix 21 Crosshair: Red Eclipse / OpenGameArt.org CC-BY-SA 3.0

Appendix 21: Sounds: SoundImage.org

Appendix 22: Cards Kenney.nl

Includes text taken from Wikipedia, some of which is edited CC-BY 3.0

Creative Commons

Some of the resources used in the appendix is licensed in Creative Commons.

Some extracts from Wikipedia is also in Creative Commons.

See https://creativecommons.org/ for full info

The Main Ones Are:

License Conditions

Creators choose a set of conditions they wish to apply to their work.

Attribution (by)

All CC licenses require that others who use your work in any way must give you credit the way you request, but not in a way that suggests you endorse them or their use. If they want to use your work without giving you credit or for endorsement purposes, they must get your permission first.

ShareAlike (sa)

You let others copy, distribute, display, perform, and modify your work, as long as they distribute any modified work on the same terms. If they want to distribute modified works under other terms, they must get your permission first.

NonCommercial (nc)

You let others copy, distribute, display, perform, and (unless you have chosen NoDerivatives) modify and use your work for any purpose other than commercially unless they get your permission first.

NoDerivatives (nd)

You let others copy, distribute, display and perform only original copies of your work. If they want to modify your work, they must get your permission first.

Attribution 4.0 International (CC BY 4.0)

This is a human-readable summary of (and not a substitute for) the license. Disclaimer.

You are free to:

- Share — copy and redistribute the material in any medium or format

- Adapt — remix, transform, and build upon the material
- for any purpose, even commercially.
- The licensor cannot revoke these freedoms as long as you follow the license terms.

_____

Under the following terms:

- Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

This is a human-readable summary of (and not a substitute for) the license. Disclaimer.

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material
- for any purpose, even commercially.
- The licensor cannot revoke these freedoms as long as you follow the license terms.

_____

Under the following terms:

- Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
- No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

Public domain

Our licenses help authors keep and manage their copyright on terms they choose. Our public domain tools,

on the other hand, enable authors and copyright owners who want to dedicate their works to the worldwide public domain to do so, and facilitate the labeling and discovery of works that are already free of known copyright restrictions.

CC0

Use this universal tool if you are a holder of copyright or database rights, and you wish to waive all your interests that may exist in your work worldwide. Because copyright laws differ around the world, you may use this tool even though you may not have copyright in your jurisdiction, but want to be sure to eliminate any copyrights you may have in other jurisdictions.

- Learn more
- Use this tool

Public Domain Mark

Use this tool if you have identified a work that is free of known copyright restrictions. Creative Commons does not recommend this tool for works that are restricted by copyright laws in one or more jurisdictions.

Introduction

Welcome

A note from the author:

Congratulations!

You are about to learn the basics of GameMaker Studio 2, and potentially start a career in game making.

This book is an introduction to the game making process, an introduction to GameMaker Studio 2, and other considerations when making your first game.

GameMaker Studio 2 is a powerful piece of software for making games. This book only covers the basics, but is a great place start.

Best of luck with you game making endeavours,

Ben

Over the last ten years or so I have written many books on game programming and have completed over two-hundred game projects. During that time I have learnt GML coding to a reasonable level, and have picked up many skills, tips and tricks and methology for making games in GameMaker & Game Maker Studio 2.

The purpose of this book is to provide you with some of the knowledge that I have acquired. I make no claim that I'm the best coder or designer, but I do have a proficient understanding that I would like to instil on other budding game makers.

Through my website, I set up a number of polls and gained feedback on what game to make and which graphics to use, in total over 500 people have voted on my site, and chose a side-scrolling war zone themed shooter. Thanks to everyone who voted. This book covers my approach to make said game.

Unlike previous books of mine that focused mainly on the actual GML code, this book covers the full design progress, with some code thrown in. It focuses on:

- Starting With An Idea
- Initial Planning & Preparation
- Software 7 Financing
- Game Assets
- Refining Resources
- Beta Testing & Debugging
- Programming
- Final Testing
- Publishing
- Game Promotion
- Additional Considerations
- Summary
- +An Appendix Of Commonly Used GML Coding

I will be the first to admit that the process of making a game is dynamic and fluid, and as such may not follow the order above. This will change depending on your level on GML, whether you have made a similar game before, the genre, and the complexity of the game. That said, the above order is a great place to start.

So, you have GameMaker Studio 2 installed. Let's start it up and start making a game. Then again, let's not. Jumping straight in is a bad idea, not least for the following reasons:

- You have no idea at this stage what the game will be about
- You have not yet decided on the look of the game
- You have no idea what the objects will be and how they will interact
- Jumping in blindly will make the whole game creation process more difficult for you
- You will come up with extra ideas for the game, and adding them when the basic game has been made will make this confusing and difficult

That said, if you just want to create one game element to see what it looks like, or how a certain feature works, or basic player movement, I consider that perfectly OK. Attempting the whole game with no planning is a big no-no, especially if you are quite new to GameMaker or the game making process.

GameMaker Studio 2 is a very powerful and adaptable software for making 2D games, I would go as far as to say that if you can make pretty much any 2D game that you can think of.

Game Resources

This game will consist of graphics and audio from a couple of sites, due to licensing restrictions I can't provide them as a download, but I will include a link so you can access them should you decide to make the game covered in this book. The main focus of this book is on the design and programming considerations, with some of the more prominent coding dissected and explained. You will not need the assets to enjoy this book.

All game graphical assets used in the main game are from the great website **GameDeveloperStudio.com**. If you wish to remake the game made in the book, you can access the assets directly from this site. The site does have several free assets, so you can swap them in instead of using purchased assets if you are working on a low budget.

Main Book Contents

The main areas covered in the book are:

1 Starting With An Idea
This section covers what you need to do with your initial ideas and how to take them forward.

2 Initial Planning & Preparation
Take your ideas forward, design the basic game layout, what objects will be present, and how they will interact.

3 Software & Financing
Software and resources cost money, this chapter covers some of the options available when funding your game.

4 Game Assets
Possible design issues, and how to tweak your ideas.

5 Refining Resources
Setting up and editing resources so they are ready for your game,

6 Beta Testing & Debugging
Testing the game, fixing bugs, and implementing feedback.

7 Programming
Covers some of the coding required to implement aspects from your game design. This also covers a way to make the game in small chunks, so you can test it as you go.

8 Final Testing
Polishing off the game and making it ready for publication.

9 Publishing & Game Promotion
Where to publish your game.

10 Game Promotion
Summary of the book.

About GameMaker

(Edited From Wikipedia) CC-SA 3.0

GameMaker Studio (formerly Animo until 1999, Game Maker until 2011, GameMaker until 2012, and GameMaker: Studio until 2017) is a cross-platform game engine developed by YoYo Games. *I'm showing my age here, but vagely remember Animo, and have followed it's progression since then.*

GameMaker accommodates the creation of cross-platform and multi-genre video games using a custom drag-and-drop visual programming language or a scripting language known as Game Maker Language, which can be used to develop more advanced games that could not be created just by using the drag and drop features. GameMaker was originally designed to allow novice computer programmers to be able to make computer games without much programming knowledge by use of these actions. Recent versions of software also focus on appealing to advanced developers.

I would add, that the software has now reached a point that if you can design a 2D game, it is possible to make it in GameMaker Studio 2.

Overview

GameMaker is primarily intended for making games with 2D graphics, allowing out-of-box use of raster graphics, vector graphics (via SWF), and 2D skeletal animations (via Esoteric Software's Spine) along with a large standard library for drawing graphics and 2D primitives. While the software allows for use of 3D graphics, this is in form of vertex buffer and matrix functions, and as such not intended for novice users.

The engine uses Direct3D on Windows, UWP, and Xbox One; OpenGL on macOS and Linux; OpenGL ES on Android and iOS, WebGL or 2d canvas on HTML5, and proprietary APIs on consoles.

The engine's primary element is an IDE with built-in editors for raster graphics, level design, scripting, paths, and shaders (GLSL or HLSL). Additional functionality can be implemented in software's scripting language or platform-specific native extensions.

Supported platforms

GameMaker supports building for Microsoft Windows, macOS, Ubuntu, HTML5, Android, iOS, Amazon Fire TV, Android TV, Microsoft UWP, PlayStation 4, and Xbox One; support for the Nintendo Switch was announced in March 2018, with Undertale to be the first such title to be brought to the Switch.

In past, GameMaker supported building for Windows Phone (deprecated in favor of UWP), Tizen, PlayStation 3, and PlayStation Vita (not supported in GMS2 "largely for business reasons").

PlayStation Portable support was demonstrated in May 2010,but never made publicly available (with only a small selection of titles using it).

Raspberry Pi support was demonstrated in February 2016, but as of May 2018 not released.

Between 2007 and 2011, YoYo Games maintained a custom web player plugin for GameMaker games before releasing it as open-source mid-2011 and finally deprecating in favor of HTML5 export.

Drag and Drop

Drag and Drop (DnD) is GameMaker's visual scripting tool.

DnD allows developers to perform common tasks (like instantiating objects, calling functions, or working with files and data structures) without having to write a single line of code. It remains to be largely aimed at novice users.

While historically DnD remained fairly limited in what can be comfortably done with it, GameMaker Studio 2 had seen an overhaul to the system, allowing more tasks to be done with DnD, and having it translate directly to code (with an in-IDE preview for users interested in migrating to code).

GameMaker Language

GameMaker Language is GameMaker's scripting language. It is an imperative, dynamically typed language commonly likened to JavaScript and C-like languages.

The language historically tries to accommodate different programming backgrounds and styles - BASIC/Lua style "and" / "or" keywords can be used interchangeably with C-style "&&" / "||" operators; parentheses around conditions in if-statements and loops can be omitted; semicolons are largely optional (insertion happens at the end of statement; compile error is raised in case of ambiguity).

The language's default mode of operation on native platforms is via a stack machine; it can also be source-to-source compiled to C++ via LLVM for higher performance. On HTML5, GML is source-to-source compiled to JavaScript with optimizations and minification applied in non-debug builds.

History

GameMaker was originally developed by Mark Overmars. The program was first released on 15 November 1999 under the name of Animo (at the time, a graphics tool with limited visual scripting capabilities). First versions of program were being developed in Delphi.

Subsequent releases seen the name changed to Game Maker and software moving towards more general-purpose 2d game development.

Versions below 5.0 have been freeware; version 5.1 introduced an optional registration fee; version 5.3 (January 2004) introduced a number of new features for registered users, including particle systems, networking, and possibility to extend games using DLLs.

Version 6.0 (October 2004) introduced limited functionality for use of 3D graphics, as well as migrating the runtime's drawing pipeline from VCL to DirectX.

Growing public interest led Overmars to seek help in expanding the program, which led to partnership with YoYo Games in 2007.From this point onward, development was handled by YoYo Games while Overmars retained a position as one of company's directors. Version 7.0 was the first to emerge under this partnership.

The first macOS compatible version of program was released in 2009, allowing games to be made for two operating systems with minimal changes.

Version 8.1 (April 2011) sees the name changed to GameMaker (lacking a space) to avoid any confusion with the 1991 software Game-Maker. This version also had the runtime rewritten in C++ to address performance concerns with previous versions.

September 2011 sees the initial release of "GameMaker: HTML5" - a new version of software with capability to export games for web browsers alongside with desktop.

GameMaker: Studio entered public beta in March 2012 and enjoyed a full release in May 2012. Initial supported platforms included Windows, Mac, HTML5, Android, and iOS. Additional platforms and features were introduced over the years following; Late 2012 there was an accident with anti-piracy measures misfiring for some legitimate users.

In February 2015, GameMaker was acquired by Playtech together with YoYo Games. Announcement reassured that GameMaker will be further improved and states plans to appeal to broader demographic, including advanced developers.

November 2016 sees the initial release of GameMaker Studio 2 beta, with full release in March 2017. This version spots a completely redesigned IDE (rewritten in C#) and a number of new editor and runtime features.

Reception

The program currently holds a rating of 8.5/10 on Mod DB based on 223 user reviews; many cite its flexibility and ease of use as positives and instability, crashes, project corruption and outdated features as negatives. Douglas Clements of Indie Game Magazine wrote that the program "[s]implifies and streamlines game development" and is "easy for beginners yet powerful enough to grow as you develop", though noting that "resource objects have to be gathered if unable to create" and that licensing between Steam and the YoYo Games website is "convoluted".

Chapter 1 Starting With An Idea

Initial Idea

The idea for the game covered in this book was chosen by visitors to my website. The initial idea was a war themed side scrolling shooter.

Let's start by defining a brief for the game, and list some of its features and elements. These are just initial ideas, we may drop some or add more as the design process progresses.

I'll be taking ideas from other games that have been made in a similar genre. I feel that borrowing ideas from other games is perfectly OK, taking actual game mechanics or graphics is not OK.

So here are some ideas that I think would work with the game we are making:

- Infinite Scroller – Survive as long as you can
- Parallax Backgrounds – Give a sense of depth to the game
- Moveable Player – Move the player and allow to shoot weapons
- Basic Enemy – Moves across the screen
- Advanced Enemy – Moves in circular path and shoots at the player
- Boss Enemy – Formidable enemy
- Multiple Weapons – Player can collect upgrades to their weapons
- Set Health – Player has set amount of health, game over when all lost
- Highscore System – Save and display the player's best score
- Game Aim – Survive as long as possible and get the highest score

That's great as a starting point. Let's dissect these ideas a bit further:

Please note that my initial sketches were pretty rough and have been re-done for the purpose of the book. So long as you understand your own sketches and what they represent, your sketches don't have to be works of art.

Chapter 2 Initial Planning & Preparation

Let's now draft what objects will be needed for the game, what they will do and some pseudo code.

By pseudo code I mean noting what the code will have to do, and how it will work, without yet writing any code. I consider this one of the most important steps in the overall design process, so when you get to the code writing stage you have a clear idea of what you need the code to do. Feel free to make as many sketches as you need, to visualize any sprites, movement or actions.

obj_player

This object is the one that the player will control.

The main features of this object:

Change y position on keypesses of W and S

Add or subtract to y position based on keypress.

Change x position on keypresses of A and D

Add or subtract to y position based on keypress.

Change the image angle based on y position

Check if y value is below or above the middile point, use this value to calculate the image angle. This is so the player's image points up when going up, and down accordingly.

Keep player within a certain area

Use the clamp function to keep x and y within fixed values.

Move back to central point when no keypress

Check if no keypress, move back to middle point.

Control parallax backgrounds based on y position

Change the y position of backgrounds based on player's y position. Have foreground move more to create a cool parallax effect.

Chapter 3 Software & Financing

Costs mentioned in the chapter are the full price at the time of publishing. Software pricing and functions may change.

Working with Different Budgets

Depending on your available budget, your game design and creation process will vary. I am approaching this chapter based on the options available as an indie-game-developer (an abbreviation of independent video game development). I see indie game development as a game created by a small team or an individual, typically with a small (or zero) budget.

The budgets I consider for this chapter are:

- Zero (or as close to $0 as possible)
- A medium budget (which I set at $500)
- A big budget (which I set at $1000)

Zero (or as close to $0 as possible)

It is totally feasible to make a game (even quite a good one) with practically no cash at all. There is a plethora of free assets available (both for graphics and audio) for making your game (see chapter 4 on game assets) which creators have made and are happy for you to use in your game projects (ensure to check what usage restrictions there are – usually just giving credit is enough but do check).

Software for making audio and graphics is available for free.

The Creators version of GameMaker Studio 2 is available for a mere $39, which allows publishing of games that can be played on Windows.

Therefore, it is possible to start making indie-games for under $40.

A medium budget (which I set at $500)

Spending a little more on assets can improve the look of your game, as much as I respect sites such as **OpenGameArt.org**, an do use it a lot when prototyping games, it has the problem that the style of the artwork varies a lot, so trying to make a game with matching assets can be difficult. A site such as Robert Brooks' **GameDeveloperStudio.com**, has assets that can be used in unison, so all assets follow the same theme. The assets on his site are fairly priced and he has a simple licence for reuse.

A budget of this size also allows for software that allows you to make your own graphics, a great example is Sprite Pro, which will set you back $59. You also may consider purchasing Spine for $69 which can work in unison with GameMaker Studio and do some pretty cool things with your graphics resources. If you are also

Chapter 4 Game Assets

Just a brief chapter on game assets, as it is covered in more detail in other chapters.
I consider the following important points to consider when sourcing assets (both audio and graphics):

- Is it worth the price? Are you over paying for an asset that will mean many game sales before you get your investment back?
- Does the graphics / audio match in with you game theme and other assets?
- Have you purchased the correct licence?
- Have read the licence correctly?
- If using a free font, did you say thanks with a donation?
- Is the asset ready to use in your project? Or require lots of additional work?
- If it is a free asset, did you give credit to the author?

Chapter 6 Beta Testing & Debugging

Beta Testing

I have decided to place this chapter before the programming one, even though you would obviously take on feedback after you have programmed an initial idea. The reason for this is that the programming section will include changes made after the debugging stage, so in chapter 7 you will see the finished project.

You're unikely to go error free on the first attempt. Beta testing allows you to find errors, issues and problems and act upton them. Usually your beta project will be sent to as many people as possible in order to get some feedback. You don't need to act upon every suggestion – though you are likely to get many good ideas. Feedback could cover issues such as:

- Crashing
- Poor controls
- Graphic issues
- Playability suggestions
- Changing game difficulty

When I do beta testing, I'm always amazed by some of the great ideas that beta testers provide, and help propel my game to the next level.

There is a file in the download resources: Game_Base.zip which shows the game before feedback from the beta testers has been taken on board. You may find it of interest to compare this with the finished game project.

Chapter 7 Programming

Programming Introduction

The Programming Introduction covers the initial basics you will need to work through the programming chapter, it is strongly suggested that you do this section before attempting anything else, if you are new to GameMaker Studio 2 / GML.

In this section we'll make a very basic clicker type 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		

There will be an object instance 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 and it will magicaly jump to a new position. If you don't click the object before the time runs out, you will lose a life – and it jumps to a new position. If you lose all your lives, then the game is over. We'll also have basic menu that shows the current highscore of the game if present.

This game is here to show you around GameMaker Studio 2's IDE, setting up objects and programming what they do, how to assign sprites and play sounds. It certainly wont win any awards, but does serve as great introduction. The sketch below shows the ideas for the menu room and game room:

SKETCH A

Chapter 8 Final Testing

The final stage is to make some final checks and make sure everything is working and correctly set up. Ideally there should only be minor tweaks required at this point. You should check for the following:

- Audio does not have any blank sound at start or end
- Texture pages are set large enough for over sized graphics you may have used
- Variables change as expected
- You are using the correct variable types (i.e. local, global or var)
- Collision boxes are set up according to the sprite and how it behaves
- You are destroying instances when no longer needed (i.e. outside the room)
- You are cleaning up paths, or ds structures when done with
- Room sizes are set up with the same aspect ratio (or room size)
- Your beta testers have toughly tested the game (see below)

Finally, send out the compiled project file (whether for PC, iOS, Android, etc.) and get your beta-testers to check it works OK on a range of different devices as expected. Fingers crossed that everything works as intended – otherwise a few more weeks of tweaks maybe required.

Congratulations! You have made a computer game. I hope it leads on to a great career in game making.

Chapter 9 Publishing & Game Promotion

So, you've spent several hundred hours making your game, the least people can do is play it!

There are a few places you can promote your game:

Social Media

A great way to make gamers aware of your game, and to direct them to sites to download / play your creation.

YoYo Games Forum

Fellow game creators love to see what you've made, and are more than happy to provide feedback.

Steam

Get your game listed on probably the biggest gaming site around. Requires a fee, which is recouped if you reach a certain amount of sales.

Itch.io

A great site made for indies' games. A great way to get your game seen and make a few $$.

GameJolt

Another great place to list your game. Easy to use, and simple to create a revenue stream for your game.

Google Play

If you have the export module for Google Play, it is a great to list and sell your games. Your game will probably need some tweaking if originally made for Windows – though worth the effort.

Appendix 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
- Saving values such as score
- Positioning instances
- Determing whether player has weapon upgrade
- + Lots more

Note: There are a number of variable types. The ones focused on in this book are built-in, instance, local 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 that start with global. Infront of them, for example, global.weapon, can also be accessed by any other object within your game. You'll learn more about instance and global variables, and how to use them as you work through this book.

Instance variables, for example x and y, and size. These are used only by the specific instance that uses it.

The basic code for drawing text is:

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

A real working example would be: To draw text “Hello World” at position 100x100:

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

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**, 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, add a **Draw Event**, and put the following code:

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

Create a room **room_example** 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***.

This will draw the value of ***example_text*** at the screen position 200,200, with 0,0 being at the top left. An example showing room positions is *Figure A_1_1:*

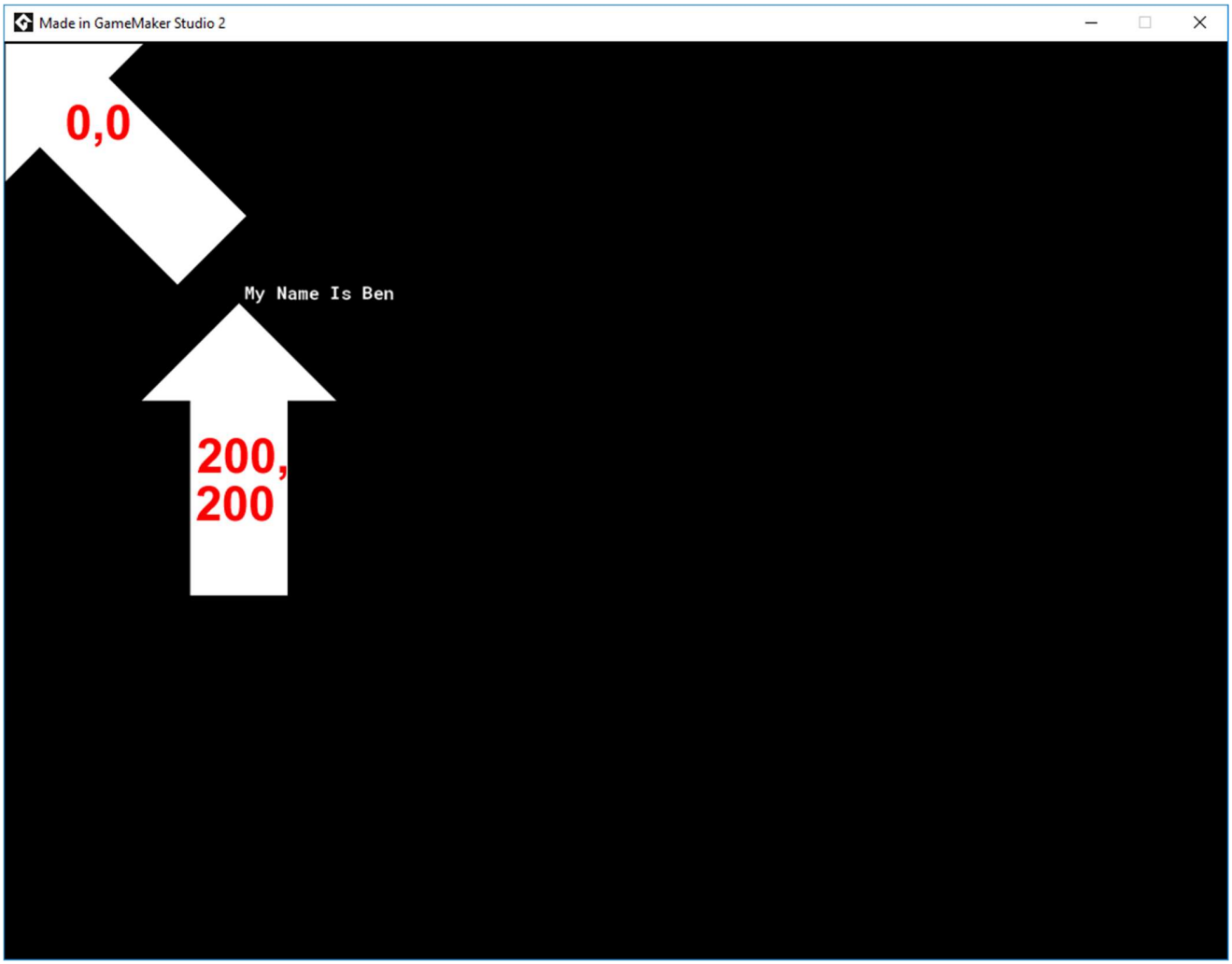

Figure A_1_1. *Showing various locations in a room*

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. Change the **Create Event** code to:

```
my_age=21;
```

Then use the following code in 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 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 mod b;
```

would set c as 6; (b goes into a twice with a remainder of 6).

```
c=a div b;
```

would set c as 2 (b goes into a 2 times).

```
c=a / b;
```

would set c as 2.65 (approx.).

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.

To make testing easier, GameMaker Studio 2 will create the same sequence of numbers each time a game is played through. 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);
```

Will draw "**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";
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 two 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, or create your own text input system. 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**). These are generally called flags, and will be used a lot when you make larger games. This type of variable is discussed more in Appendix 2 Conditionals.

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";
```

Basic Projects

A) Make a program that takes in name, age, and date of birth and displays it on the screen.

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.

Point for attempting this question

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.

Point for attempting this question

Point for good formatting

Points for using their own data input system

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");
}
```

Appendix 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.
- Determining if a weapon is active or not

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");
}
```

In the above example the **true** result will be processed.

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 2 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 here if true
}
else
{
    //do something second part here if false
}
```

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)
}
```

Basic Projects

- A) Create a password system where the user has to enter a correct password to continue.
- B) Create a simple text input system using keypresses. Allow the user to enter their name. Then store as global.name when enter is pressed. Limit name to 10 characters

  Project Note: Look up usage of keyboard_string

Advanced Projects

- C) 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.

Appendix 3 Drawing

GameMaker Studio 2 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 to. 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 A_1_1* in Appendix 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 A_3_1* shows how to select this, and the options available.

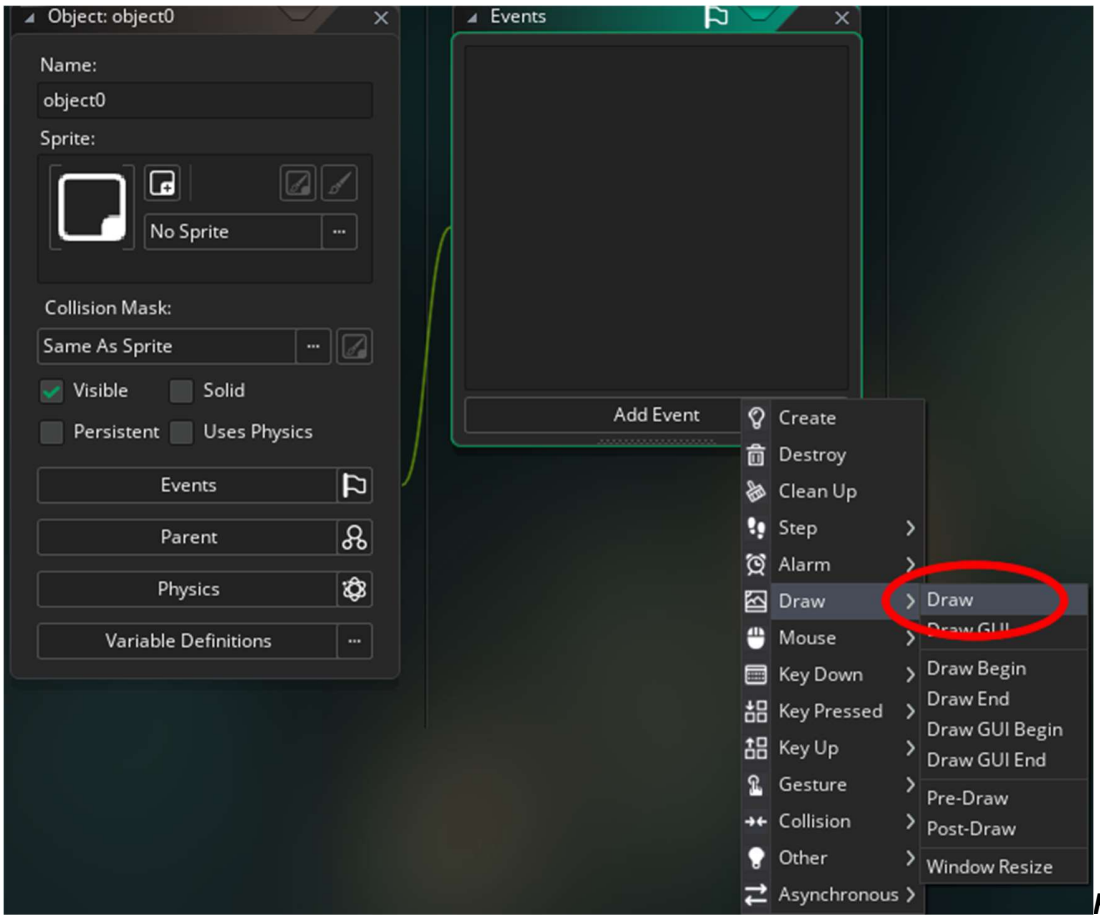

Figure A_3_1: *Showing how to select draw event*

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 colorr:

```
draw_set_colour(c_orange);
```

Colour can also be set using hexadecimal values prefixed with a '***$***' character, which in GameMaker Studio 2 is in the format BBGGRR:

```
draw_set_colour($FFA040);
```

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

```
colour=make_colour_rgb(240, 90, 100);
```

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(255, 160, 64);
draw_set_colour(my_colour);
draw_circle(50, 50, 25, false);
```

The above example would 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 2.

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).

Advanced Project

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

note on projects for Appendix 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();
```

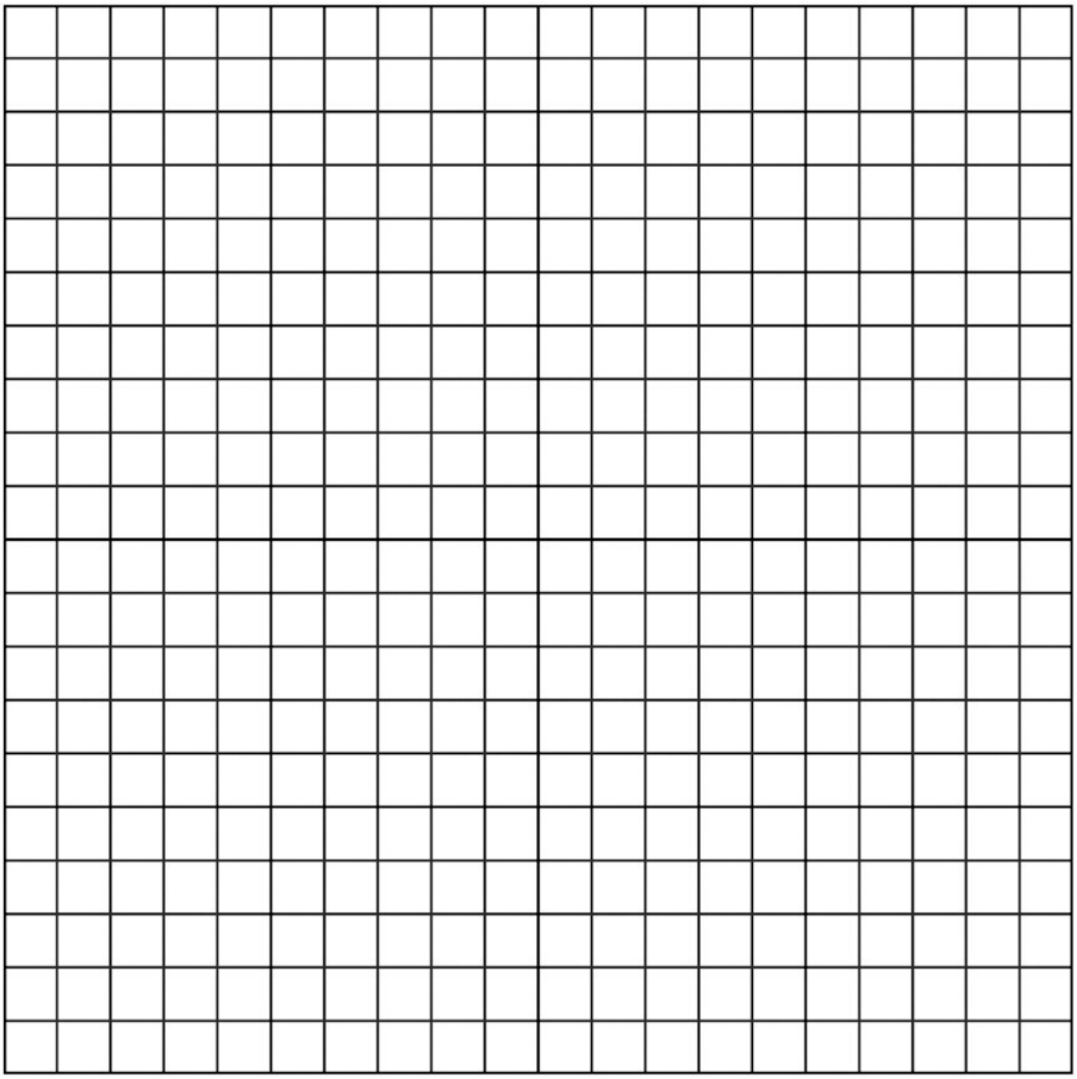

Figure A_3_1: *Graph sheet for drawing on*
Scale: 1 Square=___ Pixels

Appendix 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
- Backgrounds and Foregrounds

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* ***health, lives*** *and* ***score*** *are OK to draw without being declared.*

Create a new project in GameMaker Studio 2, along with a new object **obj_example**.

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 by right clicking on 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, 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`

Note: You can insert a new line using /n

If you want to draw a value of a variable that is not a string, use the string() function with the real 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.

When you apply drawing formatting, like font, colour, alpha, 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, and to reset alpha back to 1 after you have changed it.

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 A_4_1*:

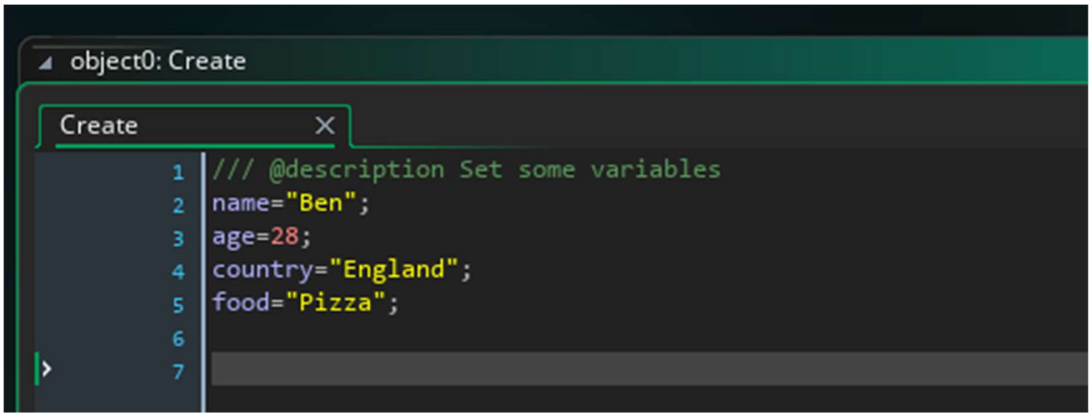

Figure A_4_1. *Showing create event*

You can then set a font, for example, as shown in *Figure A_4_2*:

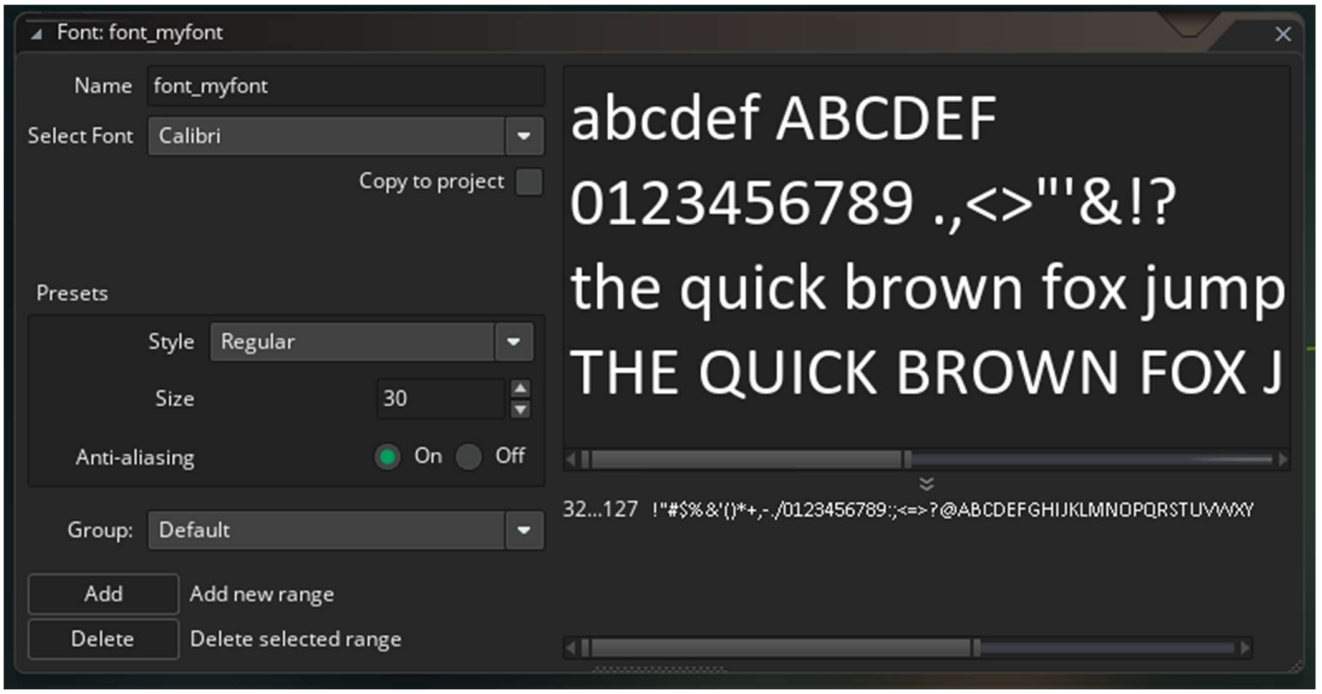

Figure A_4_2: *Setting a font*

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_myfont);
draw_set_halign(fa_center);
draw_set_valign(fa_middle); draw_set_colour(c_red);
draw_text(300,200,"His name is "+name+". \n He is
"+string(age)+" years old. \n He lives in "+country+".  \n 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 A_4_3:*

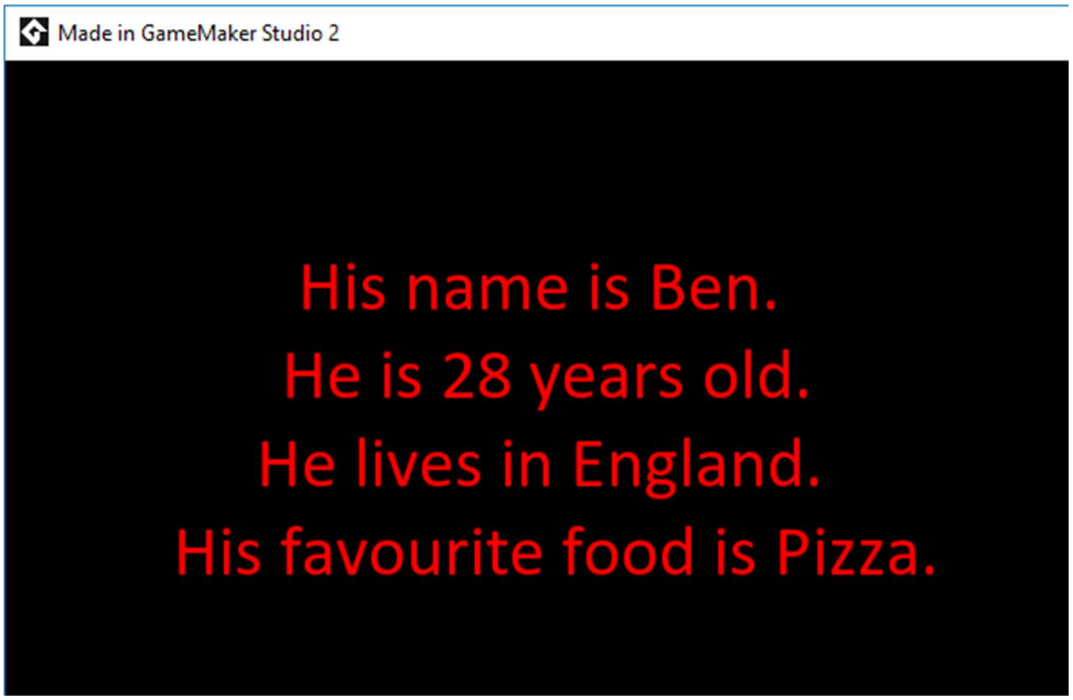

Figure A_4_3: *Showing example output*

Create a new project. Load the sprite from the resources folder, and give it a practical name, something short like **spr_test**.

Our goal is to draw this sprite in a few different ways, so create an object, **obj_test** do not assign a sprite, we will be drawing this using code in the **Draw Event.** Add a **Draw Event** with the following code to draw a normal sprite on the screen:

```
/// @description drawing
draw_sprite(spr_test, 0, 200, 200);
```

Place one instance of this object in **room0** and test. This will draw the sprite **spr_test**, using sub image 0 and position 200, 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 200,200 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, when it is assigned to an object. 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.

Note: The image index counting starts at 0. So if your sprite has just one image it will be index 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:

```
/// @description drawing
draw_sprite(spr_test, 0, 200, 200);
draw_sprite_ext(spr_test, 0, 400, 400, 0.8, 1.2, 45, c_blue,1);
```

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:

Figure A_4_4 shows a sprite drawn normally, and with the code above:

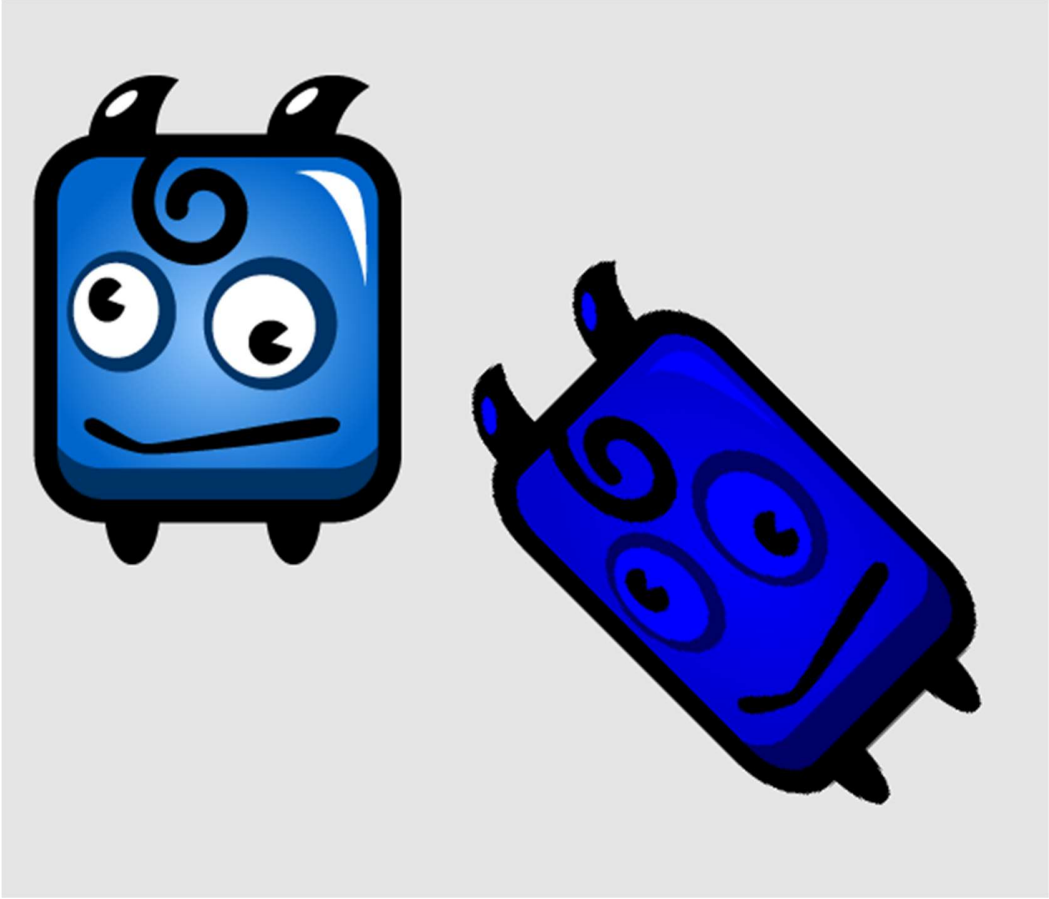

Figure A_4_4: *Showing sprite drawn normally, and with draw_sprite_ext*

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

Basic Projects

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

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).
- 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

Appendix 5 Keyboard Input & Simple Movement

Keyboard interaction is one of the key elements of a game.

They can be used for:

- Moving a player
- Choosing a level to play
- Changing game options
- Setting cheat mode
- Switching weapons
- Picking up items

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

In addition to keyboard_check, there are other options available

. Note that there is a **strong distinction** between these three functions:

keyboard_check checks whether the key is currently being pressed.

keyboard_check_pressed checks whether the key has just been pressed.

keyboard_check_released checks whether the key has just been released.

---

*Note: There is a **strong distinction** between these three functions – it is very important that you understand the difference and use the correct code when making your game*

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 being used. The same applies for the other arrow keys. You can combine keypresses in a cool way to make movement:

```
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 2 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_set_colour(c_white);
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.

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.
- 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.

Advanced Project

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

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

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

Appendix 6 Objects & Events

This appendix describes using objects and reflects what has been learned previouly. Objects are the lifeblood of GameMaker Studio 2. 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
- Draw sprites and variables on screen

Objects consist of events. You put your code 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:

```
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**.

Note: Global mouse events allows actions to be done if the mouse button is clicked anywhere on the screen, not just over the sprite of the object. Standard mouse events trigger when clicked over the sprite assigned to the object (actually the mask set for the sprite).

Figure A_6_1 shows the **Mouse Event** options available:

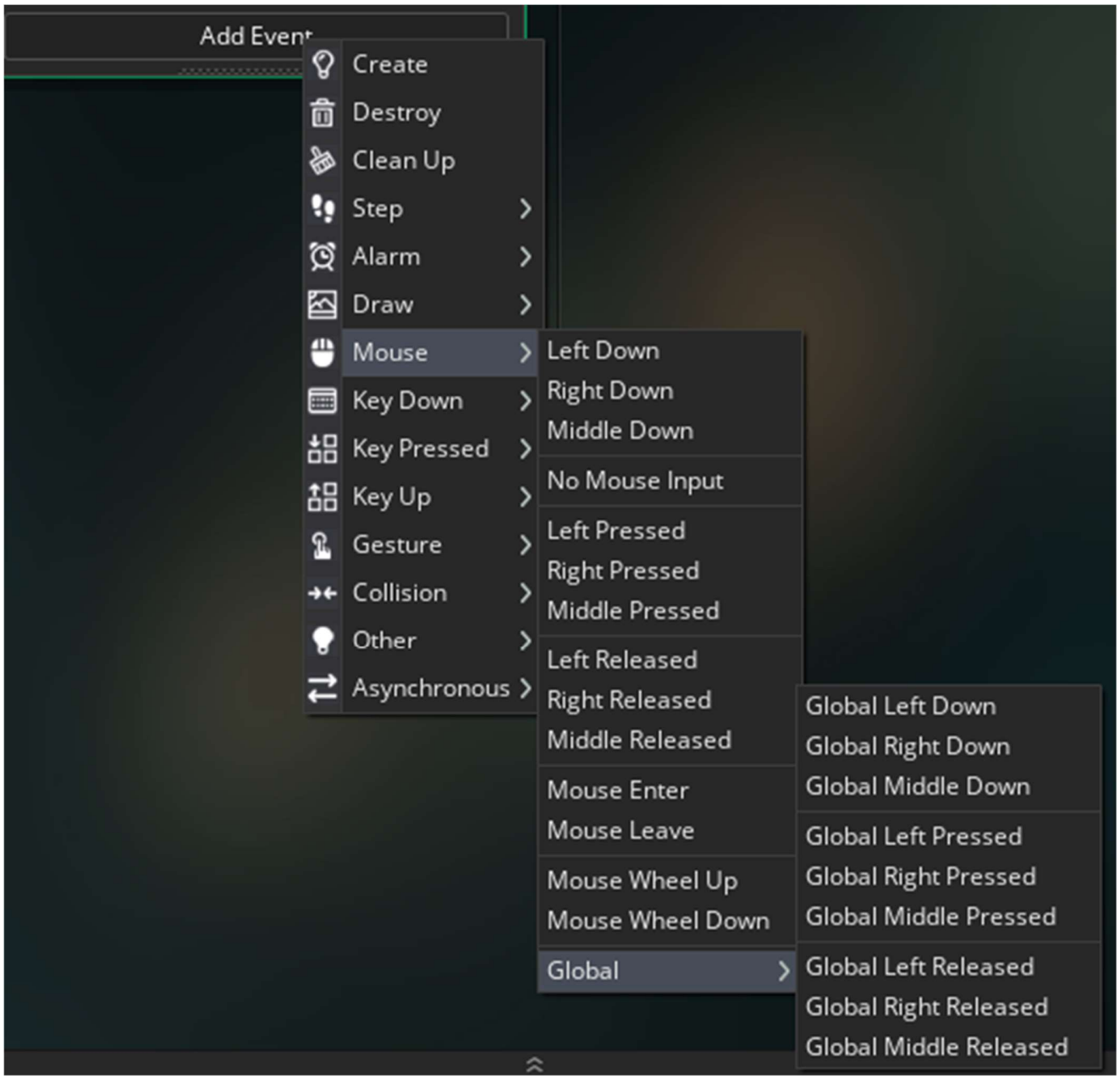

Figure A_6_1: Showing available mouse events

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 Event**, 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;
```

Note: 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 actions for drawing should be put here, drawing text, shapes, or sprites.

---

*Note: 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 :, and which ever sprite is currently assigned to the instance of that object:

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

Note: The above code will draw the text first, then the sprite. If you want the text over the sprite, just change the order.

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, going to room **room_game_over** if the player is out of lives:

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

There may be 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 right each time the left arrow key is pressed. **To imphesise, the following code will execute once each time the right arrow is pressed:**

```
if (keyboard_check(vk_right))
{
    x+=5;
}
```

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.

If you want to make code execute every step while the key is being held down use:

```
if (keyboard_check(vk_right))
{
    x+=5;
}
```

You can of course use Keyboard Events, as shown in *Figure A_6_2*:

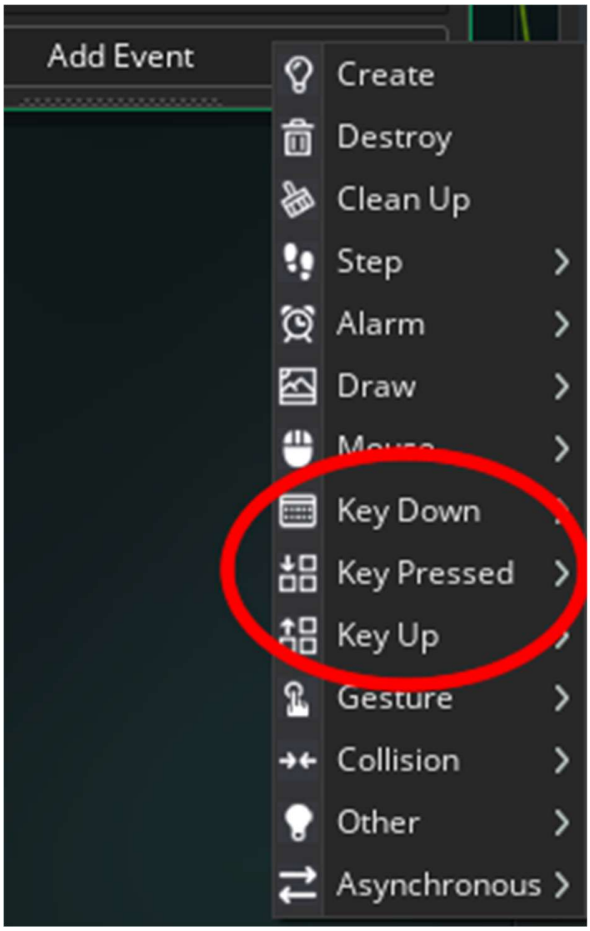

Figure A_6_2: Keyboard events

Note: There is nothing wrong in using keyboard events over gml code. In fact, sometimes it is preferable as it keeps your project more organized.

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 more often than GML code, though GML does give more flexibility in how you process collisions.

You select which object to test for a collision with, and any code 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 A_6_3*.

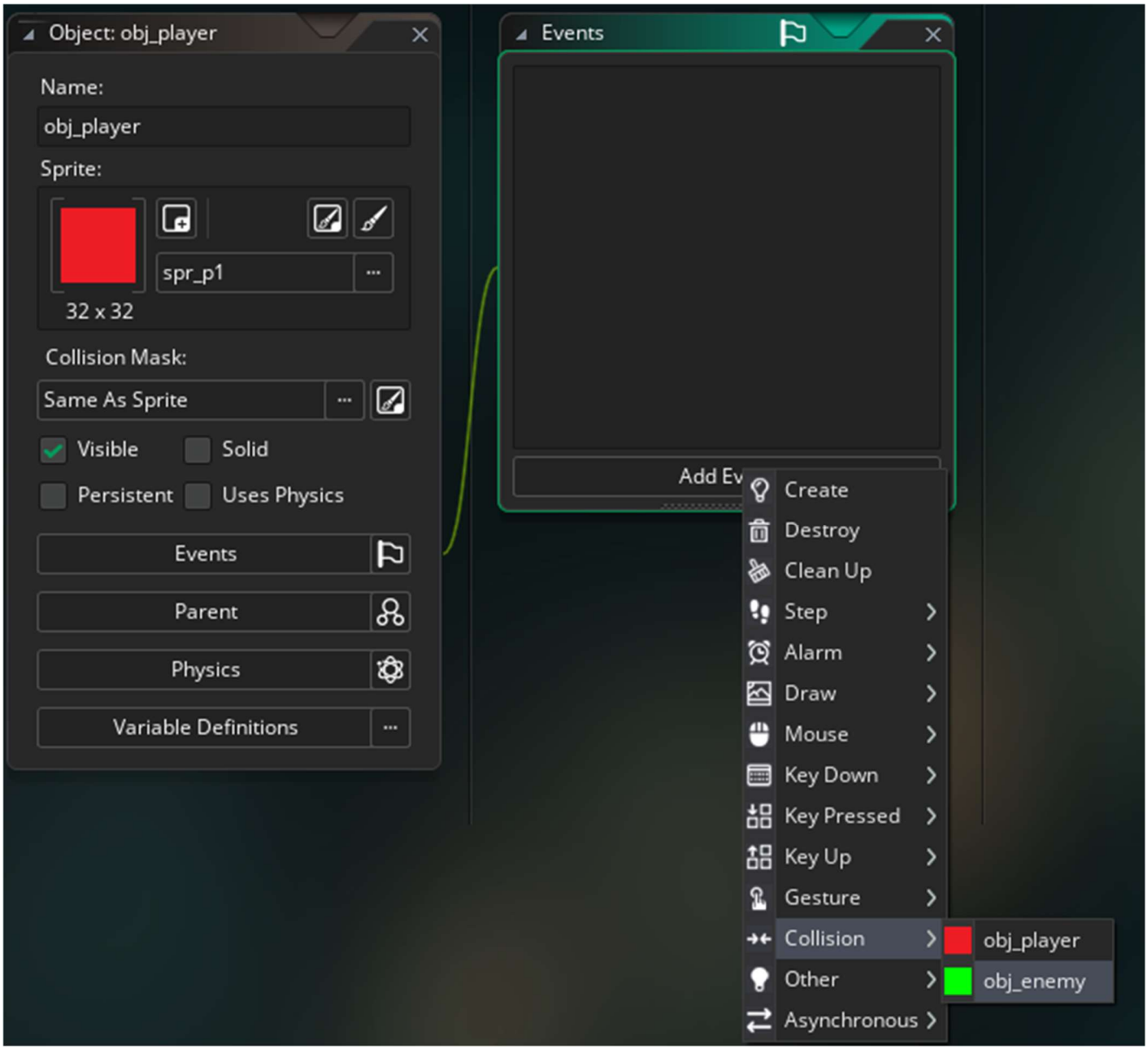

Figure A_6_3: Setting up a collision event

In this event only, other can refer to the colliding instance.

For example, based on *Figure A_6_3* above, the code could be that below which would reduce the hp of **obj_enemy** by 1:

```
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.

Basic Projects

- A) Create a moveable player. Draw the health of a player as text in red above a player when health is less than 20. When over 20 draw in white. Set it up so P and L change the value of health.
- B) Make some text change colour, at random, each time the space bar is pressed.
- 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.

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.