

INTRODUCTION TO GAME DESIGN & PROGRAMMING

FROM ZERO TO HERO

BEN TYERS

IN
**G A M E M A K E R
S T U D I O 2**

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

Learn GameMaker Studio



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

ISBN: 9781795199537
Copyright 2019 © Ben Tyers
First Edition

If you find any issues or 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.

You can download resources & completed project here:

www.learngamemakerstudio.com/assets.zip

Introduction

Welcome

Game Resources

Main Book Contents

About GameMaker

Chapter 1 Starting With An Idea

Initial Idea

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

Chapter 2 Initial Planning & Preparation

Chapter 3 Software & Financing

Working with Different Budgets

Cost of software

Development

Graphics

Free Graphics Software

Paid Graphics Software

Audio

Free Audio Software

Paid Audio Software

Pre-Made Graphics

Free Graphics

Paid Graphics

Ways of raising funds

Crowd Funding

Patreon

Social Media

Steam Early Access

Chapter 4 Game Assets

Chapter 5 Refining Resources

Graphics

Audio

Chapter 6 Beta Testing & Debugging

Beta Testing

Graphics are too large

Not responsive enough

Too easy

Player weapons are too slow

HUD is not in keeping with the rest of the game

Aspect ratio should be changed

Collision masks are imprecise

GUI life and dots not in keeping with game style

To visually show damage to player

Debugging

Chapter 7 Programming

Programming Introduction

Game Programming

Scripts

Objects

Rooms

Paths

Audio

Chapter 8 Final Testing

Chapter 9 Publishing & Game Promotion

Social Media

YoYo Games Forum

Steam

Itch.io

GameJolt

Google Play

Chapter 10 Summary

Target Audience

Pricing

Working as a Team

Useful Links

Crediting Creators

Educational Use

Where Next

Conclusion

Appendix

Appendix 1 Variables

Appendix 2 Conditionals

Appendix 3 Drawing

Appendix 4 Drawing Continued

Appendix 5 Keyboard Input & Simple Movement

Appendix 6 Objects & Events

Appendix 7 Sprites

Appendix 8 Health, Lives & Score

Appendix 9 Mouse

Appendix 10 Alarms

Appendix 11 Collisions

Appendix 12 Rooms

Appendix 13 Backgrounds

Appendix 14 Sounds

Appendix 15 Splash Screens & Menu

Appendix 16 Random

Appendix 17 AI

Appendix 18 INI Files

Appendix 19 Effects

Appendix 20 Loops

Appendix 21 Arrays

Appendix 22 DS Lists

Appendix 24 Scripts

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 methodology 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 & 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 vaguely remember Animo, and have followed its 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:

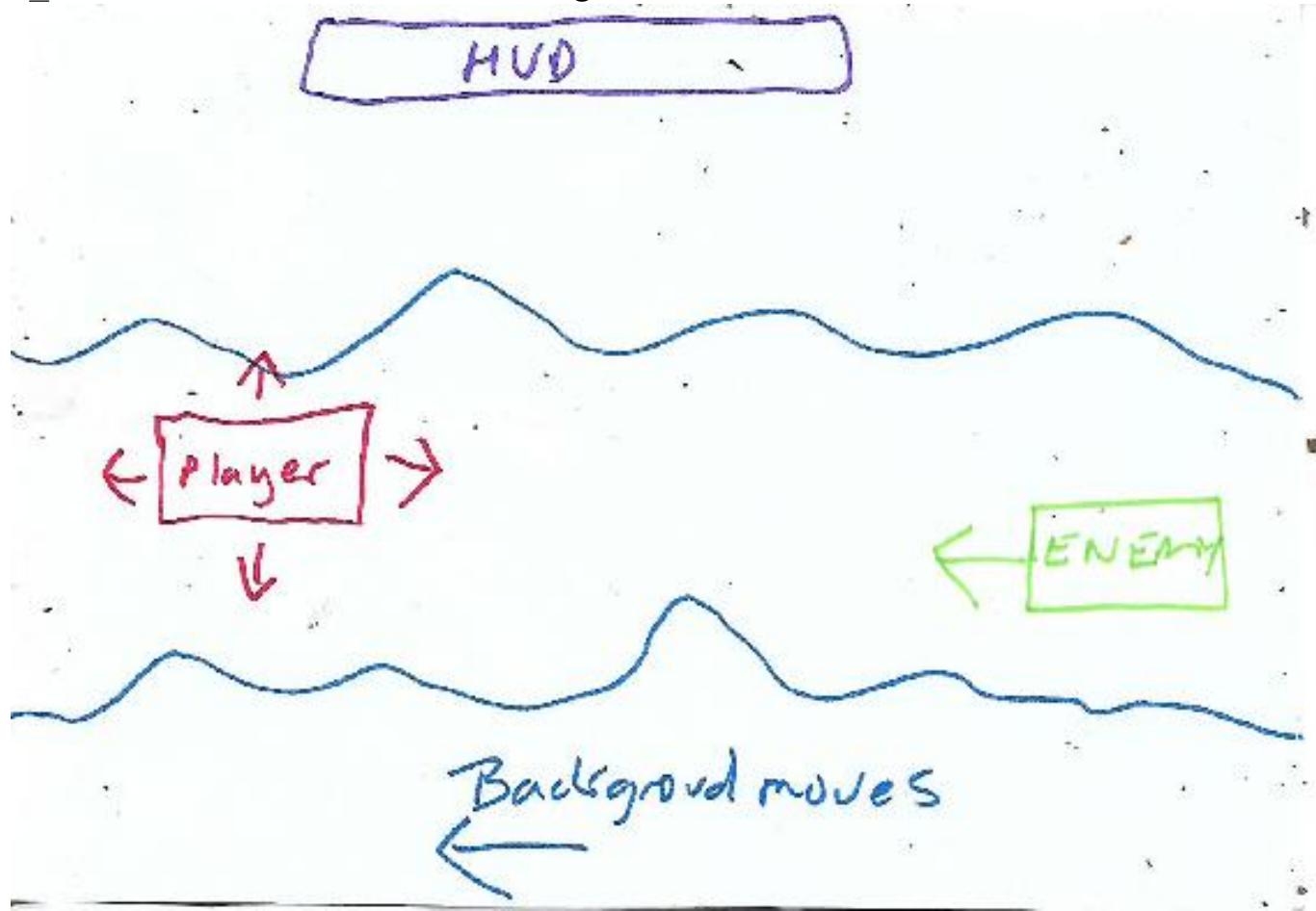
- Infinite Scroller – Survive as long as you can
- Parallax Backgrounds – Give a sense of depth to the game
- Moveable Player – Move the player and allow to shoot weapons
- Basic Enemy – Moves across the screen
- Advanced Enemy – Moves in circular path and shoots at the player
- Boss Enemy – Formidable 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.

Infinite Scroller – Survive as long as you can

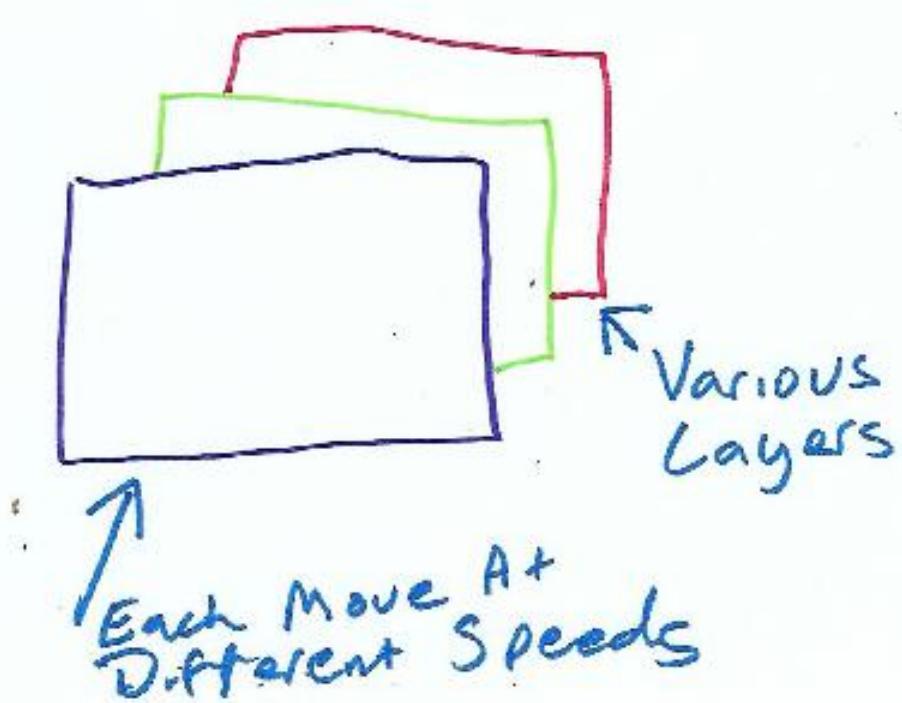
There are couple of approaches that you can be used to create the illusion of an ever-ending game. The approach that we will use is to create a static room the size of the game window, and have the elements move from the right to the left side of the room. We will do this for the backgrounds and enemies. Although a simplistic approach, when done well, it can give a great perception of depth, and can be easily tweaked. *Sketch 1_1* shows a the basic ideas for this game.



Sketch 1_1: Initial Idea

Parallax Backgrounds – Give a sense of depth to the game

Using several backgrounds that are layered and move at different speeds, an effect of movement and depth can be created. Adding a foreground at the front can enhance this effect greatly. Parallax is a graphical effect where objects closer to you move quicker than those further away. The layers will be graphics with a transparent background, so layers below can be seen. The game elements will be placed in front of these backgrounds, with a further foreground background in front of these to add a greater sense of depth. *Sketch 1_2* shows the idea:

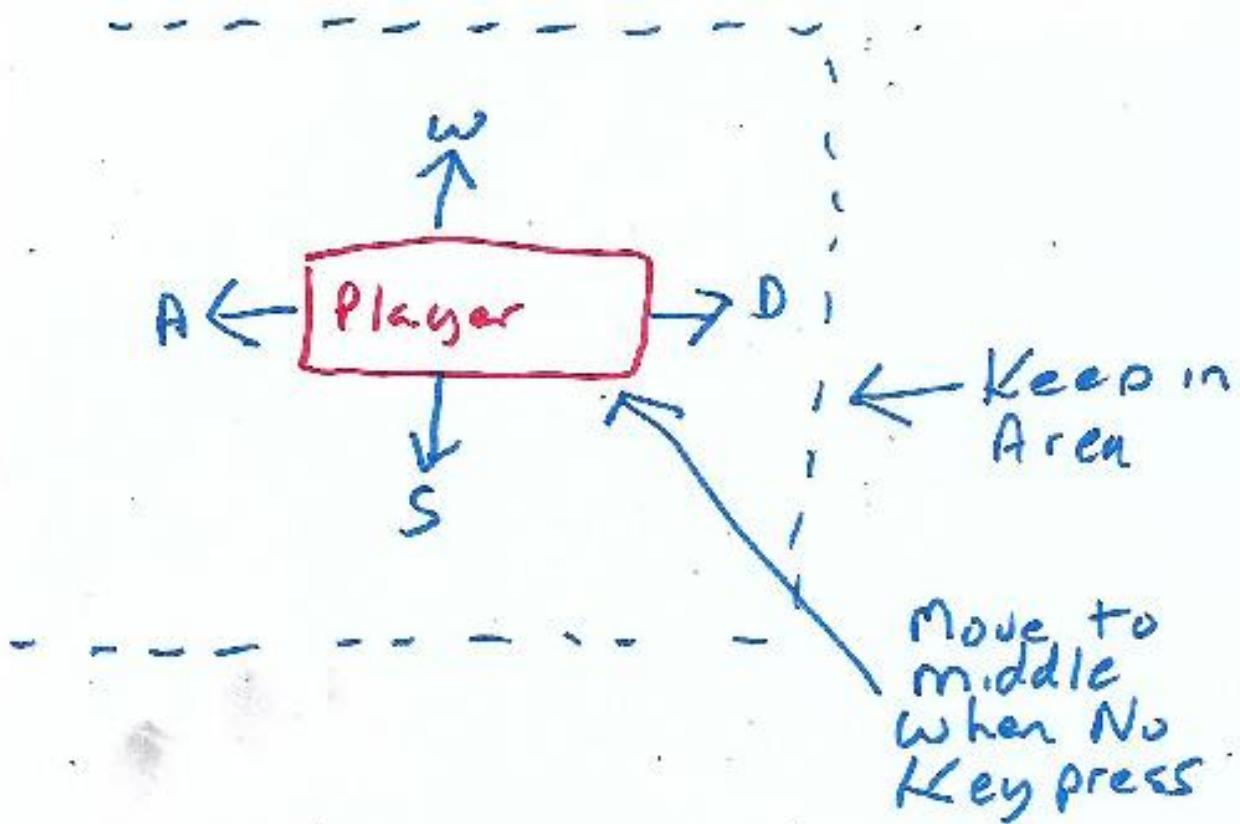


Sketch 1_2: Using different layers for a parallax effect

Moveable Player – Move the player and allow to shoot weapons

We'll set the player up so that it can be moved with WSAD keys, and left and right mouse buttons to fire the manual weapons. We will limit the area that the player can move in, and also make the player slowly move back to the middle of the movement area. We will want a smooth and responsive movement for the player. Player weapons will be set up to be used with the left and right mouse buttons (when weapon is available to use).

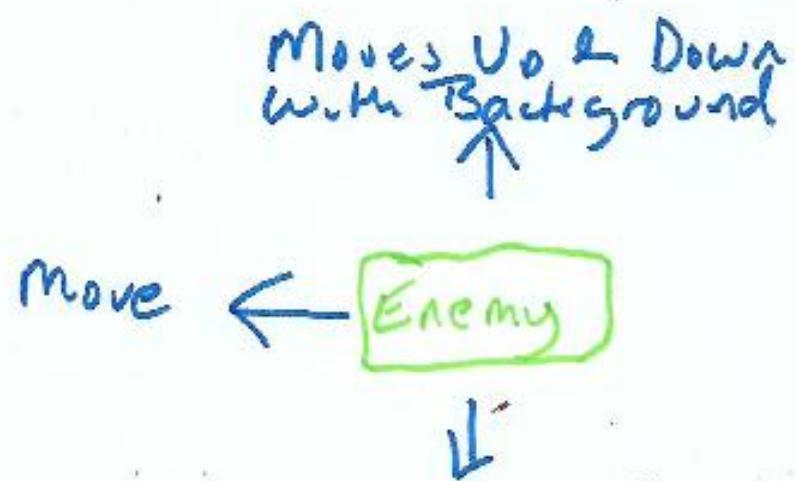
Sketch 1_3 shows this idea:



Sketch 1_3: Player movement

Basic Enemy – Moves across the screen

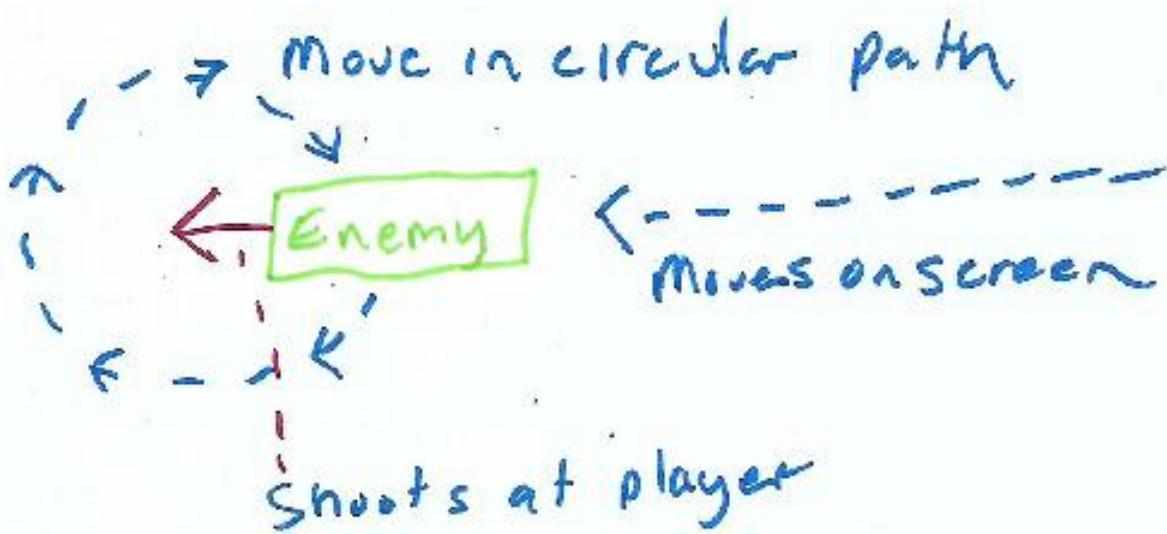
A basic enemy that moves across the bottom of the screen. Give the enemy some health that reduces when shot at by the player. Create some cook effect when it loses all health. Shown in *Sketch 1_4*:



Sketch 1_4: Basic enemy

Advanced Enemy – Moves in circular path and shoots at the player

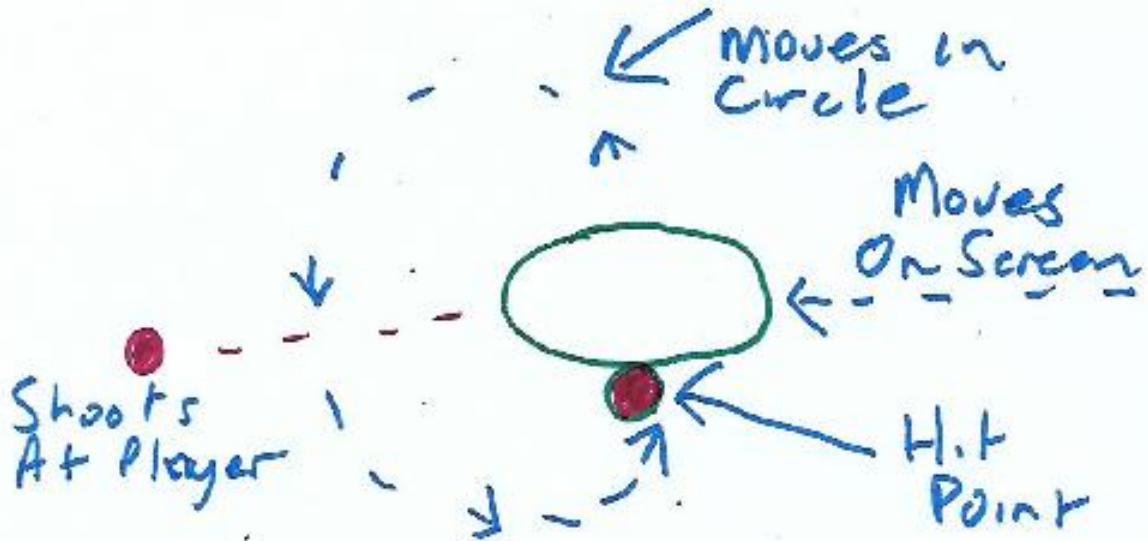
A more advanced enemy that moves in a circular pattern and shoots some sort of projectile at the player. We'll make the enemy move into the game window, then following a closed circular path. This is shown in *Sketch 1_5* :



Sketch 1_5: Showing advanced enemy movement

Boss Enemy – Formidable enemy

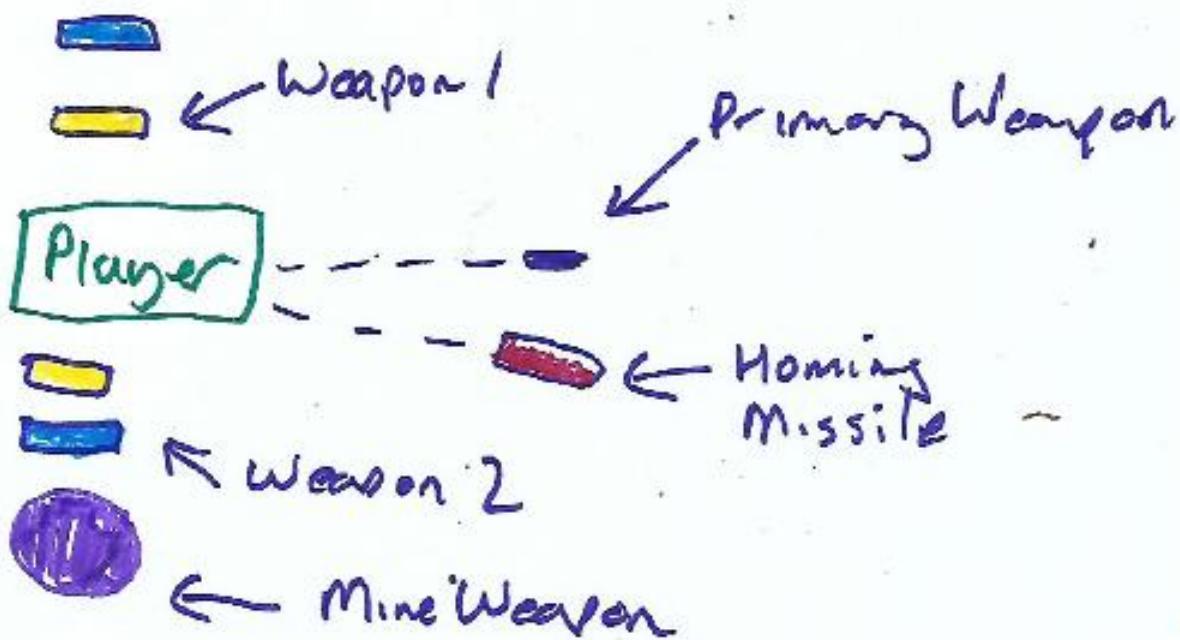
A big boss enemy that moves around the screen and shoots projectiles at the player. Have soft spot that the player must shoot at to damage it. *Sketch 1_6* shows the basic idea:



Sketch 1_6: Boss movement ideas

Multiple Weapons – Player can collect upgrades to their weapons

Have collectibles that the player can collect to add new weapons (both manual and automatic) and power upgrades. The player will start with a primary weapon, and by collecting orbs can add new weapons and upgrades. *Sketch 1_7* shows the initial idea:



Sketch 1_7: Player weapon ideas

Set Health – Player has set amount of health, game over when all lost

Have a health / hp system for the player. Player to lose health if hit by enemy or enemy bullet. Gameover when all health has been lost.

Highscore System – Save and display the player's best score

At end of each game, check player's score and against saved score and update if bigger. Display the highscore at game start.

Game Aim – Survive as long as possible and get the highest score

The aim of this game will be to survive as long as possible and get the highest score.

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 keypresses 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 middle 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.

Fire projectile on mouse left button press

Create a projectile at the player's position when left mouse is pressed. Shoot this projectile based on the player's image angle so it shoots in the direction the player is pointing.

Fire projectile on mouse right button press, if available

Create a projectile at the player's position when right mouse is pressed, and weapon is available. This will be a homing missile that locks on the nearest enemy when fired.

Auto fire weapon 3 if active

Use an alarm to fire this weapon.

Level up on collision with level up object

Will be something like:

1. Start with standard projectile
2. Add to shooter 1 top
3. Add to shooter 1 bottom
4. Add to shooter 2 top
5. Add to shooter 2 bottom
6. Triple shot
7. Mine 1
8. Mine 2
9. Weapon 3
10. Mine 3 and 4
11. Faster Shots
12. Faster Shots
13. Homing Weapon
14. Increased power on all weapons

Additional power ups will increase power.

Create a shield as active on collision with shield object

Activate the shield and reduce over time. We will set a flag to true or false whether the shield is active or not. If active we will draw the shield over the player and slowly fade its transparency, when totally faded the shield will no longer be active.

Damage self upon collision of enemy or enemy bullet (if shield is not active)

Reduce health to reflect damage.

Weapon 1

Automatically fire. Shoot 3 bullets if levelled up. Base the direction on the player's image angle. Increase shoot speed and power based on player's level.

Weapon 2

Auto fire when active. Base direction on the player's image angle. Base the direction on the player's image angle. Increase shoot speed and power based on player's level.

Mine Weapon

Circle around the player when active. Damage enemy / enemy projectiles upon collision. This can be done by affixing an offset sprite origin that stays attached to the player. Rotating the image angle will make it orbit the player. When adding extra mines, set the image angle based on current mines so they space evenly.

Homing Weapon

Create a homing weapon on right mouse clicked, when active. Home in to enemy nearest mouse position. Slowly move direction to the target. Look for new target if original target has been destroyed. Give this weapon a timed life.

Shield Bonus

Move across the screen, player to pick up. Activate shield and destroy upon collision with the player.

Power Up

Move across the screen, player to pick up. Upgrade weapons upon collision with the player, as noted on page 14.

Ground Enemy

Moves along the bottom of the screen. Player to destroy by hitting with projectiles or mines. This enemy does not fire any projectiles but will damage the player if the player collides with it. Setting movement with an initial horizontal speed should suffice. Remember to destroy after it has left the screen.

Flying Enemy

Flies in a circle motion and fires at the player. Do this by moving into the screen and then following a path. Have it shoot weapons, using an alarm system.

Boss Enemy

Flies around a set path and fires projectile at player. Has to be hit a certain area to take damage. The hit area could be another instance that moves relative to the boss enemy. A simple alarm system could be used to shoot projectiles. A nice feature would be if the boss has a weapon that always points towards the player, regardless of how the player moves, so that projectiles are fired to the player's current position.

Player Projectiles

Damages enemy if hit. Remember to destroy once outside the room.

Enemy Projectiles

Damages player if hit. Remember to destroy once outside the room.

Explosion Objects

A cool explosion effect when hp of enemy has gone. The resources I'm using from GameDeveloperStudio.com include all the sprite parts in a separate folder. This could be loaded up as sub images. The object instance that has just died can then create multiple instances of another object, setting a fixed sub image, and have these move out in various sections. I'm confident that this effect will look pretty cool.

Score Objects

When the enemy has been destroyed, we can create some kind of coin that player can pick up to increase their score. We'll set its value as the enemies starting hp value. An effect upon collection would be a good idea.

Control Objects

Various control objects to spawn enemies and draw player stats. For spawning enemies, a simple alarm system should suffice. The hp of the enemies will increase based on the level. Drawing HUD can be done with simple primitive shapes and formatted text.

Menu Objects

Allow player to select game, credits or how to play. Control using mouse middle wheel or keyboard. Click to select option. We could use a sprite with sub images for each option and display / rotate them based on the player's interaction. Take the player to appropriate room when selection is made.

Credits

Game credits displayed. We'll have the text move up the screen and gradually get smaller. We could do this by adding all the text to display into a list and take each and display. Scaling text can be simply done using transform text function.

How to Play

Explain how to play. Display a scrolling info that explains how to play the game. This can be a duplicate of the system used to display the game credits.

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**, and 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 looking to create your own audio, a budget of this size allows purchasing a DAW (such as Cubase for around \$70) or one of my favourites, Fruity Loops (available for around \$215)

A big budget (which I set at \$1000)

A budget of this size allows you to outsource parts of your game development to freelancers, whether that be audio, graphics or code. A budget of this size should also allow you some \$\$ for promoting your game, or upgrading GameMaker Studio with exports, allowing you to export your game to consoles, tablets and mobile devices. Outsourcing can be a great way to get artwork or other assets done exactly as you want them but does involve a bit of project management.

A word of caution – I have outsourced before several times, mostly for graphics. On several occasions the so-called freelancers had just copied graphics from various sites on the net. I would strongly suggest that you only hire freelancers who have a good track-record / good feedback.

Cost of software

Development

GameMaker Studio 2

The current cost of GMS 2 is:

Trial - \$0 But has limitations

Creator - \$39 – Great for making games for Windows

Developer:

Desktop \$99 – Publish to Windows, Mac & Ubuntu

Fire \$149 – Publish games to the Appstore

Web \$149 – Publish in HTML5

Mobile \$399 – Publish to Android, Amazon and iOS

UWP \$399 – Publish to Xbox One and UWP

Console:

UWP \$399 - Publish to Xbox One and UWP

Nintendo Switch \$799 – Publish to Nintendo Switch

PS4 \$799 – Publish to PS4

Xbox One \$799 – Publish to Xbox One

Ultimate \$1500 – Publish to all platforms

YoYo Games also offers an educational licence for schools and colleges. I do not have current prices, but this is available on their website.

Graphics

Free Graphics Software

GIMP

It is often used for making logos, making photographs bigger or smaller, changing colours, making many pictures part of one picture, making pictures nicer to look at, and changing file formats.

GIMP is often used as a free software alternative for the most popular Adobe Photoshop, but it is not made to be an Adobe Photoshop clone. GIMP's mascot is named Wilber.

GIMP was started in 1995 by Spencer Kimball and Peter Mattis and is now taken care of by a group of volunteers as part of the GNU Project. The newest version of GIMP is v.2.8 and it was available since March 2009. GIMP's license is the GNU General Public License, so GIMP is free software.

If you are looking to edit graphics resources from other sources, GIMP has all the tools you need.

Inkscape

Inkscape is a vector graphics (pictures made from lines instead of dots) drawing program published under the GNU General Public License. Its stated goal is to become a really good drawing tool while being able to fit in with standards for SVG graphics. Inkscape was first made for Linux, but now it is cross-platform and runs on Microsoft Windows, Mac OS X, and other Unix-like operating systems. As of 2007, Inkscape is actively being made better, and new features are added all the time.

If you are looking for free entry-level software for making your own game assets, Inkscape is an excellent place to start.

GameMaker Studio 2

GameMaker Studio 2 has its own sprite editor, Chapter 5 shows some of the cool features. If you are looking at making basic game features, or need to edit a sprite or two, this editor allows you to do so quickly.

Paid Graphics Software

Photoshop

Adobe Photoshop is a raster graphics editor developed and published by Adobe Inc. for macOS and Windows.

Photoshop was created in 1988 by Thomas and John Knoll. Since then, it has become the de facto industry standard in raster graphics editing, to the point that Photoshop has become a generic trademark leading to its use as a verb such as "to photoshop an image," "photoshopping" and "photoshop contest", though Adobe discourages such use. Photoshop can edit and compose raster images in multiple layers and supports masks, alpha compositing and several color models including RGB, CMYK, CIELAB, spot color and duotone. Photoshop uses its own PSD and PSB file formats to support these features.

In addition to raster graphics, it has limited abilities to edit or render text, vector graphics (especially through clipping path), 3D graphics and video. Photoshop's feature set can be expanded by Photoshop plug-ins, programs developed and distributed independently of Photoshop that can run inside it and offer new or enhanced features.

Photoshop's naming scheme was initially based on version numbers. However, in October 2002, following the introduction of Creative Suite branding, each new version of Photoshop was designated with "CS" plus a number; e.g., the eighth major version of Photoshop was Photoshop CS and the ninth major version was Photoshop CS2.

Photoshop CS3 through CS6 were also distributed in two different editions: Standard and Extended. In June 2013, with the introduction of Creative Cloud branding, Photoshop's licensing scheme was changed to that of software as a service rental model and the "CS" suffixes were replaced with "CC". Historically, Photoshop was bundled with additional software such as Adobe ImageReady, Adobe Fireworks, Adobe Bridge, Adobe Device Central and Adobe Camera RAW.

Alongside Photoshop, Adobe also develops and publishes Photoshop Elements, Photoshop Lightroom, Photoshop Express, Photoshop Fix, Photoshop Sketch and Photoshop Mix. Adobe also plans to launch a full-version of Photoshop for the iPad in 2019. Collectively, they are branded as "The Adobe Photoshop Family". It is currently a licensed software.

It is my understanding that Photoshop is one of the most widely used graphic editing and creation software.

The current price of the software is about \$155, though they do offer a student version, and the possibility to pay a monthly fee.

Illustrator

This software is also provided by Adobe, it is a vector-based graphics creation tool.

It can be acquired for a monthly fee.

If you are looking to make your own game graphics, then both above software is a great investment.

Note: You can get access to Photoshop and Illustrator for a single monthly fee. If you will be using both these pieces of software, you should consider this. For students and Teachers, the price is around just \$20 a month.

Audio

Free Audio Software

Audacity

Audacity is a free and open-source digital audio editor and recording application software, available for Windows, macOS/OS X and Unix-like operating systems. Audacity was started in the fall of 1999 by Dominic Mazzoni and Roger Dannenberg at Carnegie Mellon University and was released on May 28, 2000 as version 0.8.

As of October 10, 2011, it was the 11th most popular download from SourceForge, with 76.5 million downloads. Audacity won the SourceForge 2007 and 2009 Community Choice Award for Best Project for Multimedia. In March 2015 hosting was moved to FossHub and by September 2018 it had exceeded 62.5 million downloads there.

Features and usage

Audacity's main panel annotated. All the components that have been labelled are custom for Audacity.

In addition to recording audio from multiple sources, Audacity can be used for post-processing of all types of audio, including podcasts by adding effects such as normalization, trimming, and fading in and out. Audacity has also been used to record and mix entire albums, such as by Tune-Yards. It is also currently used in the UK OCR National Level 2 ICT course for the sound creation unit.

Audacity's features include:

- Four user-selectable themes enable the user to choose their preferred look and feel for the application (version 2.2.0 and later)
- Four user-selectable colorways for waveform display in audio tracks (version 2.2.1 and later)[16]
- Recording and playing back sounds
- Scrubbing (Version 2.1.1 and later)
- Timer Record [19] enables the user to schedule when a recording begins and ends to make an unattended recording
- MIDI playback is available (from version 2.2.0 onwards)
- Punch and Roll recording - for editing on-the-fly (from version 2.3.0 onwards)

- Editing via cut, copy, and paste, with unlimited levels of undo
- Features of modern multitrack audio software including navigation controls, zoom and single track edit, project pane and XY project navigation, non-destructive and destructive effect processing, audio file manipulation (cut, copy, paste)
- Amplitude envelope editing
- Precise adjustments to the audio speed (tempo) while maintaining pitch in order to synchronize it with video or run for a predetermined length of time
- Conversion of cassette tapes or records into digital tracks by splitting the audio source into multiple tracks based on silences in the source material
- Cross-platform operation — Audacity works on Windows, macOS/OS X, and Unix-like systems (including Linux and BSD)
- Audacity uses the wxWidgets software library to provide a similar graphical user interface on several different operating systems
- A large array of digital effects and plug-ins. Additional effects can be written with Nyquist, a Lisp dialect.
- Built-in LADSPA, VST(32-bit) and Nyquist plug-in support
- Noise Reduction based on sampling the noise to be minimized
- Vocal Reduction and Isolation for the creation of karaoke tracks and isolated vocal tracks
- Adjusting audio pitch while maintaining speed and adjusting audio speed while maintaining pitch
- LADSPA, VST (32-bit) and Audio Unit (macOS/OS X] effects now support real-time preview (from version 2.1.0 onwards). Note: Real-time preview does not yet support latency compensation
- Saving and loading of user presets for effect settings across sessions (from 2.1.0 onwards)
- Multitrack mixing
- Support for multi-channel modes with sampling rates up to 96 kHz with 32 bits per sample
- Audio spectrum analysis using the Fourier transform algorithm
- Importing and exporting of WAV, AIFF, MP3 (via the LAME encoder, downloaded separately), Ogg Vorbis, and all file formats supported by libsndfile library. Versions 1.3.2 and later supported Free Lossless Audio Codec (FLAC).[39] Version 1.3.6 and later also supported additional formats such as WMA, AAC, AMR and AC3 via the optional FFmpeg library
- Detection of dropout errors while recording with an overburdened CPU
- A full downloadable Manual[41] (or available online without downloading)

- Audacity supports the LV2 open standard for plugins and can therefore load software like Calf Studio Gear

I personally love this software, it is quick to use and its basic features require little skill or knowledge. I use it often for my projects.

Anvil Studio

Anvil Studio consists of a free core program with optional add-ons. The free version is a fully functional MIDI editor/sequencer which loads and saves standard MIDI-formatted files, and allows individual tracks to be edited with a:

- Staff editor
- Piano Roll editor
- Percussion editor
- TAB editor
- MIDI event list editor

The program uses the standard MIDI Sequencer-Specific event (FF 7F) to control items not specifically defined by the MIDI standard, such as:

- the font to use when rendering lyrics
- the position of notes or staff notation
- links to Pulse-code modulation formatted audio files for audio tracks

By default, Anvil Studio uses a General MIDI software synthesizer for playback, but also allows tracks to be assigned to VST instrument or external MIDI devices. It processes audio using Core Audio, ASIO, DirectX or WDM or enabled drivers.

If you are looking for entry point to make your own music, I recommend this software as a starting point.

Paid Audio Software

Fruity Loops (FL Workstation)

Fruit Loops is currently priced around from \$100 to over \$1000, depending on the edition you acquire.

It is a DAW (Digital Audio Workstation) developed by Belgian Company Image-Line. It features a GUI based around a pattern-based music sequencer. It's available for Windows & macOS.

I love this software, I have been using it for years, and I am impressed how quickly you can get some awesome music playing.

Cubase

Cubase is another great piece of audio software, priced at from about \$100 to about \$600.

I see this more suited for the professional music creator and editor. Although an extremely capable piece of software, it contains lots features that you won't need if you're just using it for basic audio editing.

Pre-Made Graphics

Free Graphics

GameDeveloperStudio.com

One of my favourite graphic sites!

Although the website offers a lot of free graphics, most are paid. See the ***Paid Graphics*** section later in this chapter for my thoughts on this site.

OpenGameArt.org

A great site with thousands of images for game developers. The majority of the sprites are CC-0 or CC-BY. The best thing about the site is that all the images are free. There are however a few downsides, mostly that some sprites need editor or backgrounds removed, and that the graphics are in many styles – so trying to find matching styles for your game can take a bit of work.

Paid Graphics

GameDeveloperStudio.com

I just love this site!

The game made in this book was made exclusively using graphical assets from this site.

What I like about this site:

- Easy to navigate
- Huge range of graphics
- Fairly priced
- Most graphics in the same style, so multiple assets can be combined
- Downloads are of high quality
- Provided in PNG format
- New assets added regularly
- Easy to understand licence

If you're making a game, or just want some free assets to play with, then checkout Useful Links in Chapter 11.

Ways of raising funds

Unless you are making a project on \$0, you are going to need same funds for software assets. You could of course use your own hard saved cash, but fortunately there are many gamers out there will to support your projects. I have seen many projects successfully funded using some of the methods on the following page. Which method you use will depend on your game and features.

Don't expect your project to magically fund itself, it is not enough to create a funding and raise funds overnight – it does require effort on your behalf.

Crowd Funding

Put simply, crowd funding is getting small amounts of money from lots of people. It is usually done via the internet, and is a great and growing way to fund small (and sometimes) large projects.

KickStarter

Kickstarter is a great site for raising funds for your game project. Basically, you offer rewards at set price points, and people pledge money in advance of release of your game.

IndieGoGo

Similar to Kickstarter, though perhaps more suited to indie game projects.

Patreon

A great way to raise funds every month. Users make a monthly pledge (from as little as \$1) to access your posts, blogs, etc.

A great way to make use of this platform is to make a game that updates each month (for example a bonus level) and your patreons get this update (setting this up is easier than it sounds). \$1 a time doesn't sound a lot, but as soon as you start getting hundreds of patreons, this can stack up quite quickly.

Social Media

Although social media cannot raise funds itself (OK, there are exceptions), it is a great way to make people aware of your game, and crowdfunding / patron projects. If you put good content out there, you will get followers. Keep your followers aware of your projects and progress.

Steam Early Access

Another option is Steam's Early Access, you allow players to test and play various stages of your game throughout production. It has the added bonus of being able to get feedback early on and address any issues as they arrive. A great option all-round.

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 your game theme and other assets?
- Have you purchased the correct licence?
- Have you 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 5 Refining Resources

Off the shelf assets are unlikely to be game ready. This chapter shows some methods to prepare assets ready for use. Graphics are done in GameMaker Studio'2 built in editor, audio is done with Audacity.

Graphics

For the purpose of this chapter I'll be using graphics from the excellent site:
GameDeveloperStudio.com

Note: I like to use PNG files for sprites, as they can be set to be transparent. JPG files do not have this, so would need editing prior to use.

Sprites from this site come in two formats, as sprite sheets and as separate images.

Figure 5_1 shows as sprite sheets, and *Figure 5_2* as separate image.

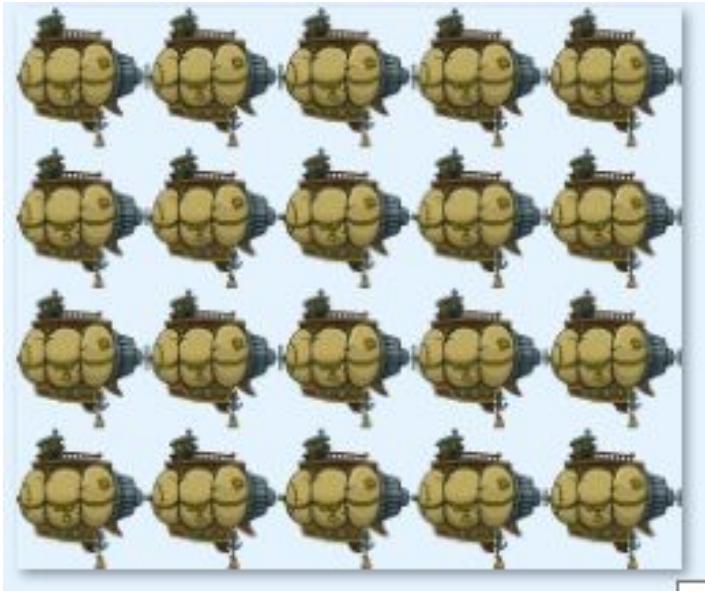


Figure 5_1: Sprite sheet example.

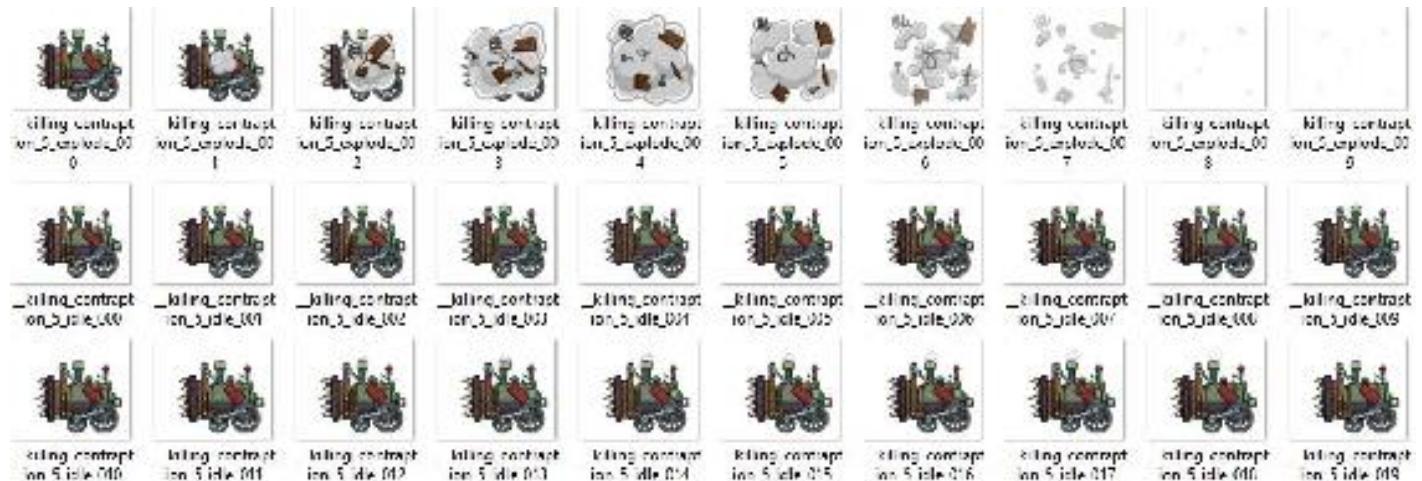


Figure 5_2: Showing as separate images

The figures below show the steps for importing and processing a sprite sheet. First right-click on **Sprites** in the resources tree:

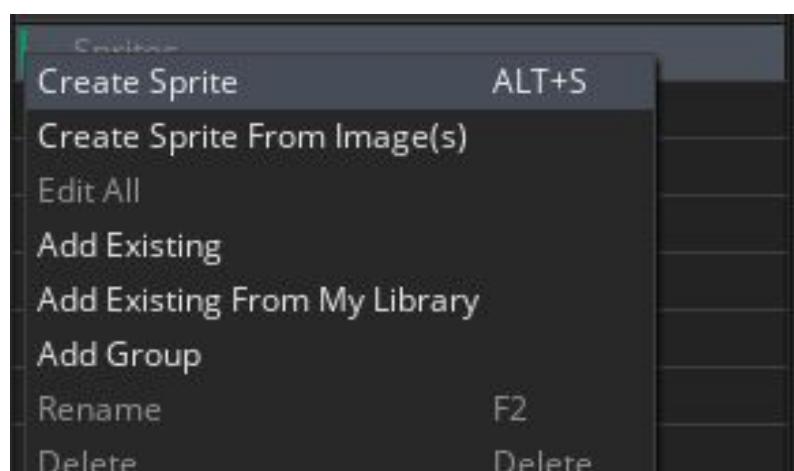


Figure 5_3: Right click and create a sprite

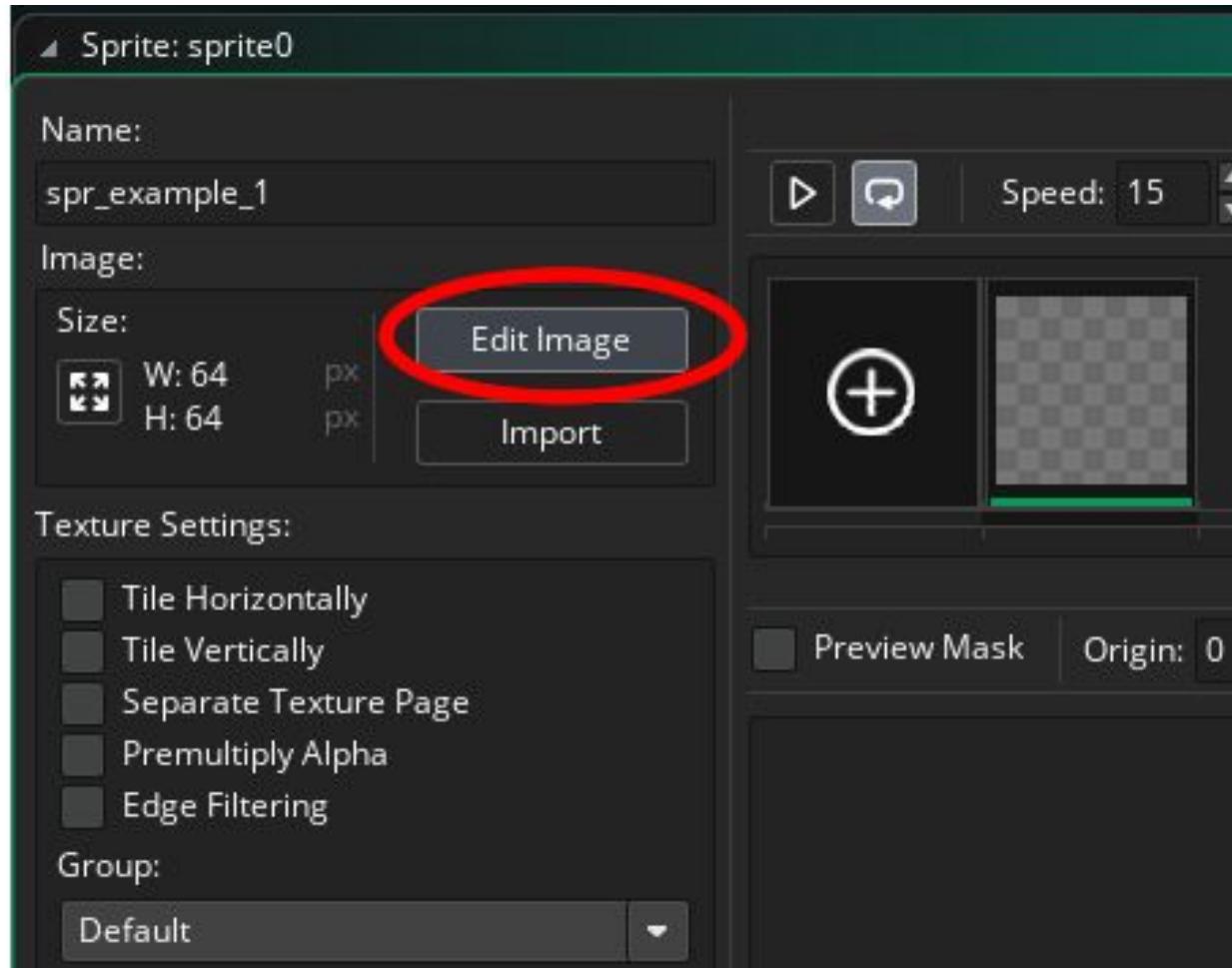


Figure 5_4: Selecting edit image

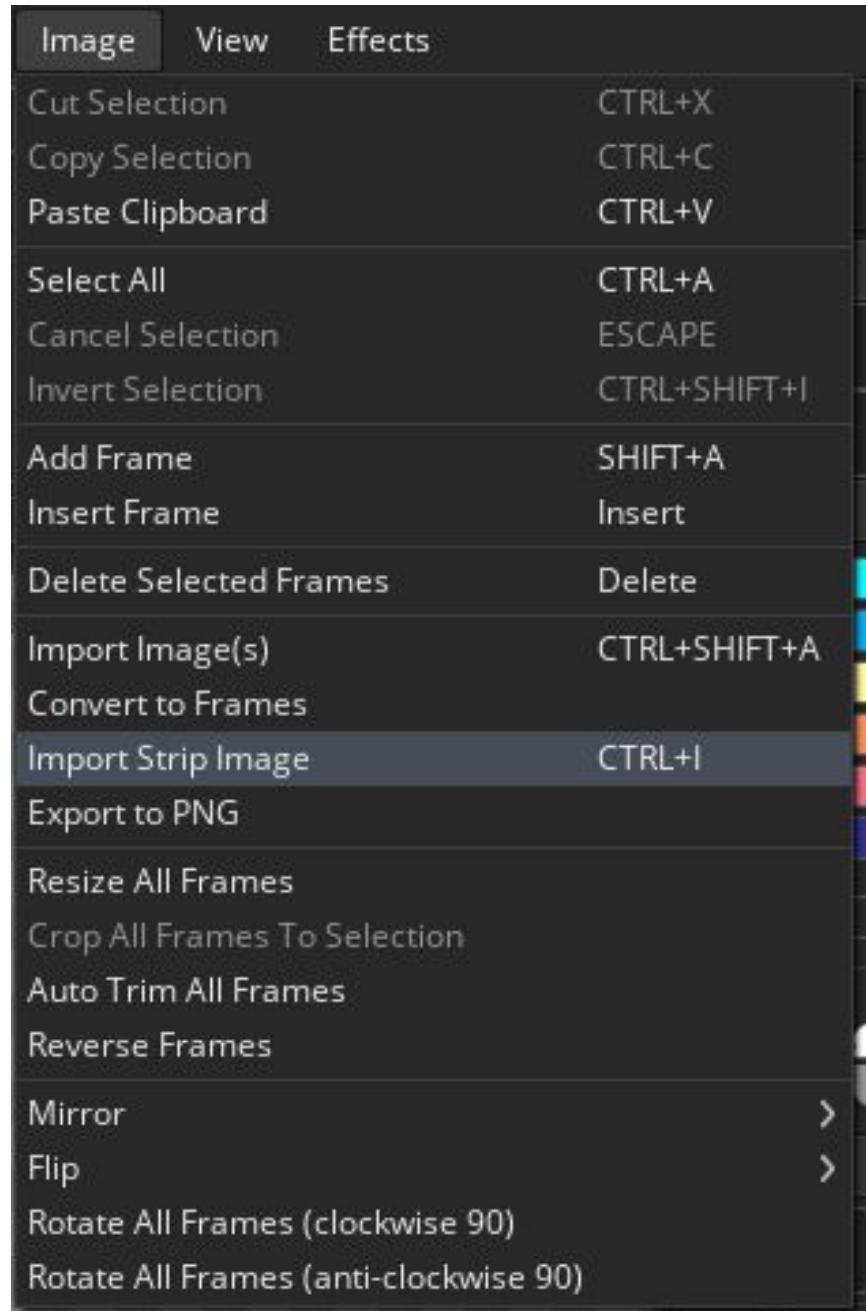


Figure 5_5: Selecting import strip image

If you look at *Figure 5_1* you will see the sprite sheet is 5 frames wide and 4 deep. If you open the folder that has the image in (do this in Windows not GameMaker Studio 2), and hover the mouse over the image you'll see a popup showing the whole dimensions of the sprite, as shown in *Figure 5_6* :

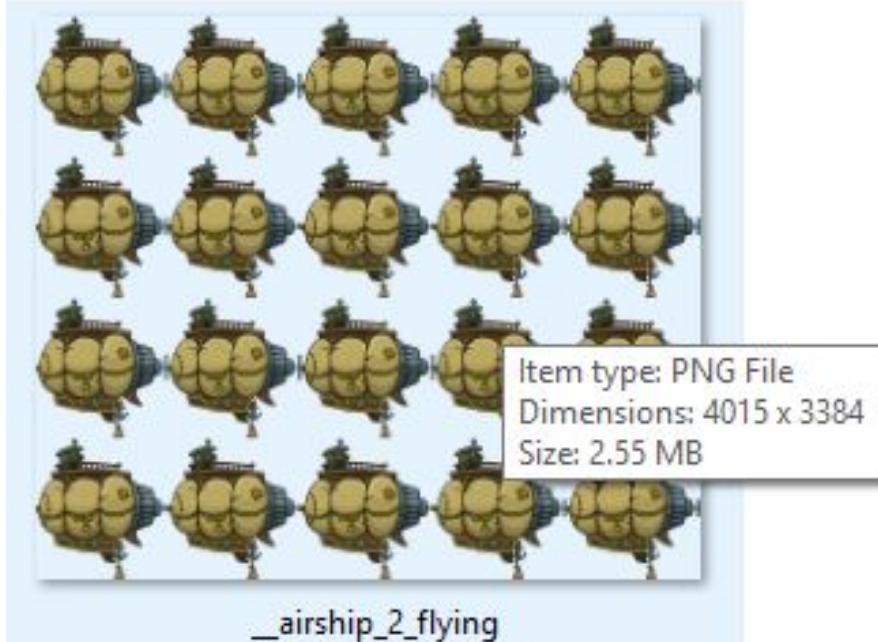


Figure 5_6: Showing image dimensions

Using a bit of math we can figure out that each subimage is 803 pixels wide and 846 high, and there are 20 images.

If we plug these values to the importer, as shown in *Figure 5_7* :

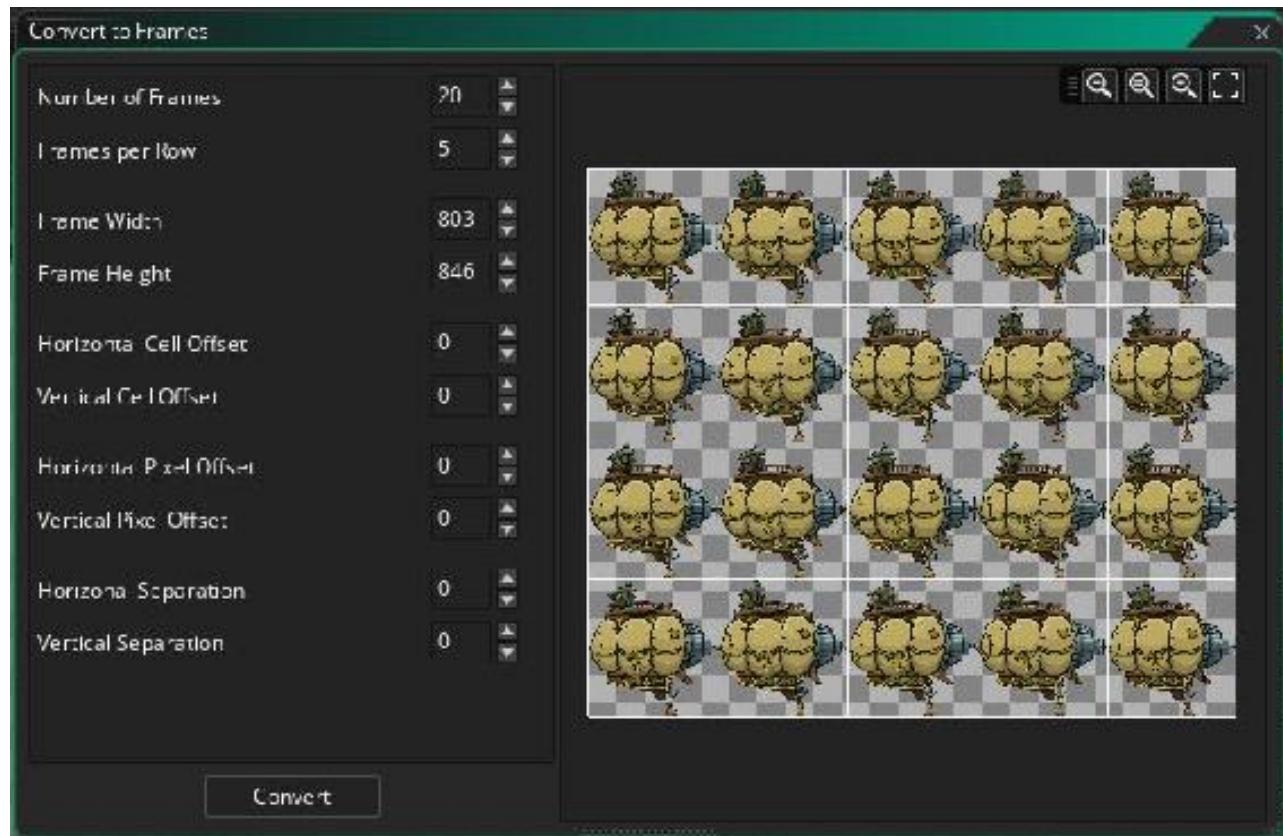


Figure 5_7: Showing import settings

We then end up with the sub images added, as shown in *Figure 5_8* :



Figure 5_8: Showing sprite after import conversion
You now have sprites that are nearly ready for your game.

The Figure's below show the process for importing multiple sub images. First create a new sprite, name is **spr_example_2** for example:

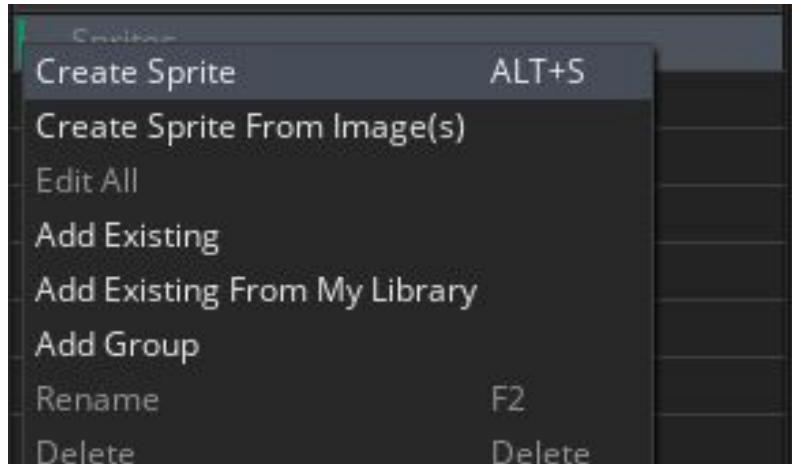


Figure 5_9: Create a new sprite

Next select Import Image:

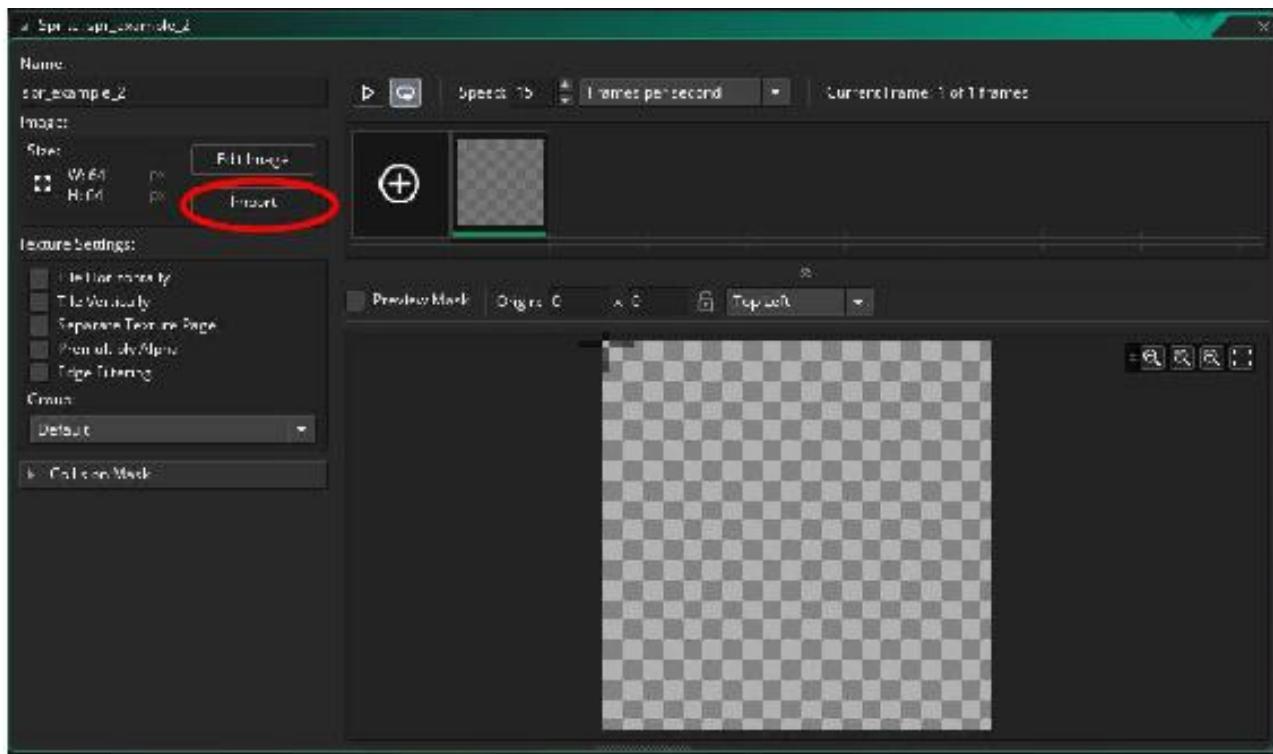


Figure 5_10: Selecting import option

Next select which sprites you want to import, you can use control and shift to select multiple images. This is shown in *Figure 5_11* :

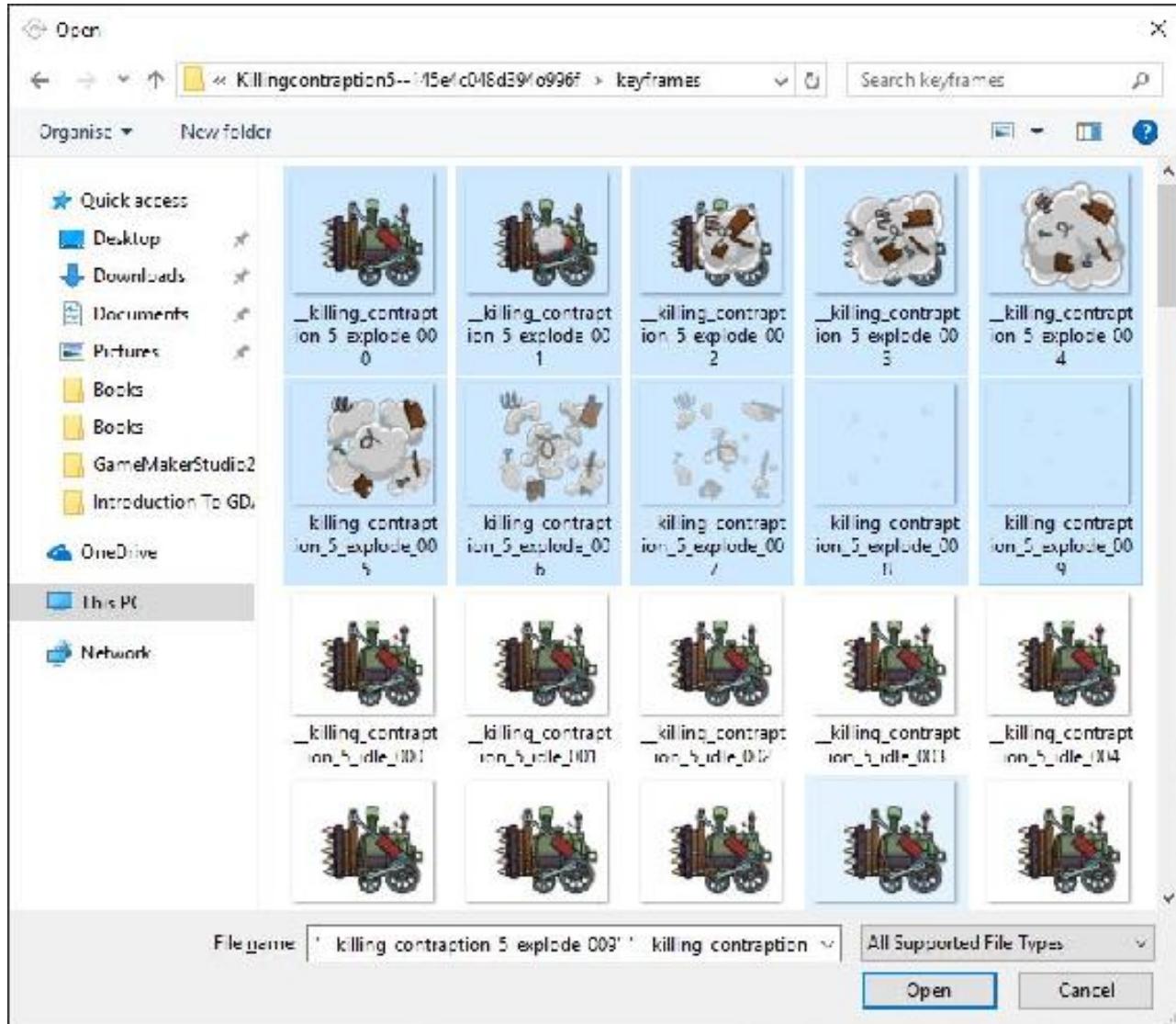


Figure 5_11: Selecting multiple images to import

Figure 5_12 below shows the sprites successfully imported:

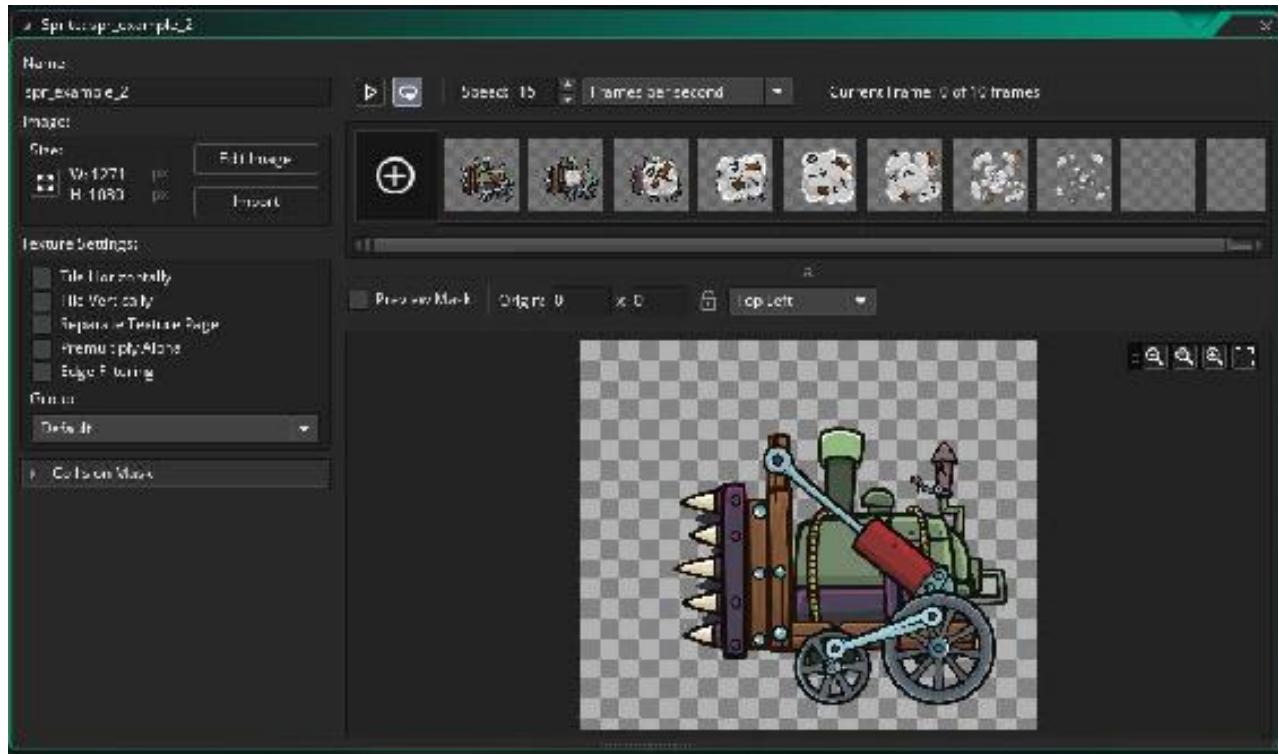


Figure 5_12: Showing sprite images imported

The sprites for **GameDeveloperStudio.com** are of a very high quality and of a high resolution, so in most case you will want to change the size to suite your game.

For example, spr_example_2 has a size of 1271 by 1080 pixels, see *Figure 5_13*:

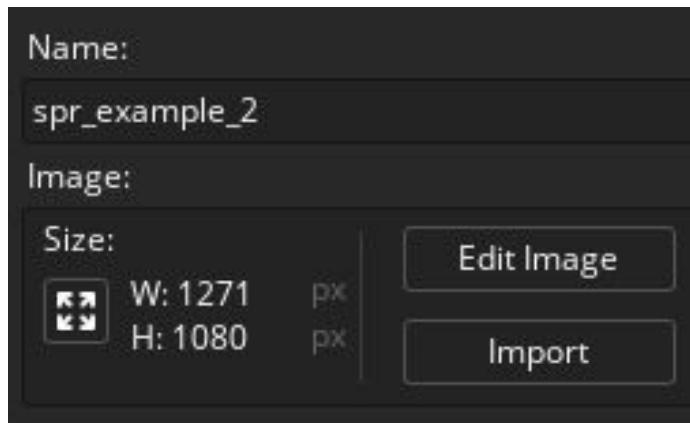


Figure 5_13: Showing sprite size

It's a quick and easy process to resize sprites, regardless of 1 or multiple frames. Open up the editor by clicking on Edit Image. As shown in *Figure 5_14* :

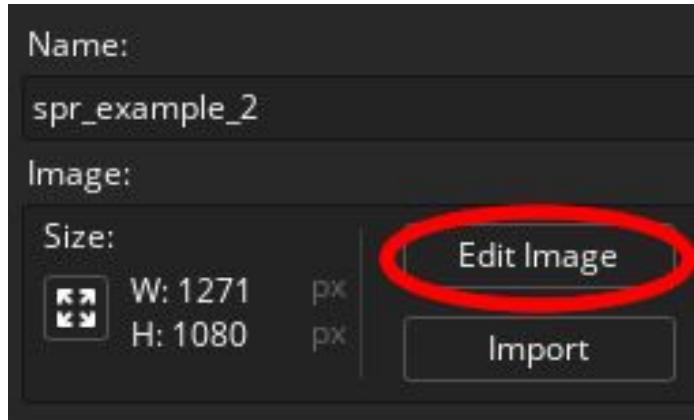


Figure 5_14: Selecting edit image option

Next under image, choose resize all frames:

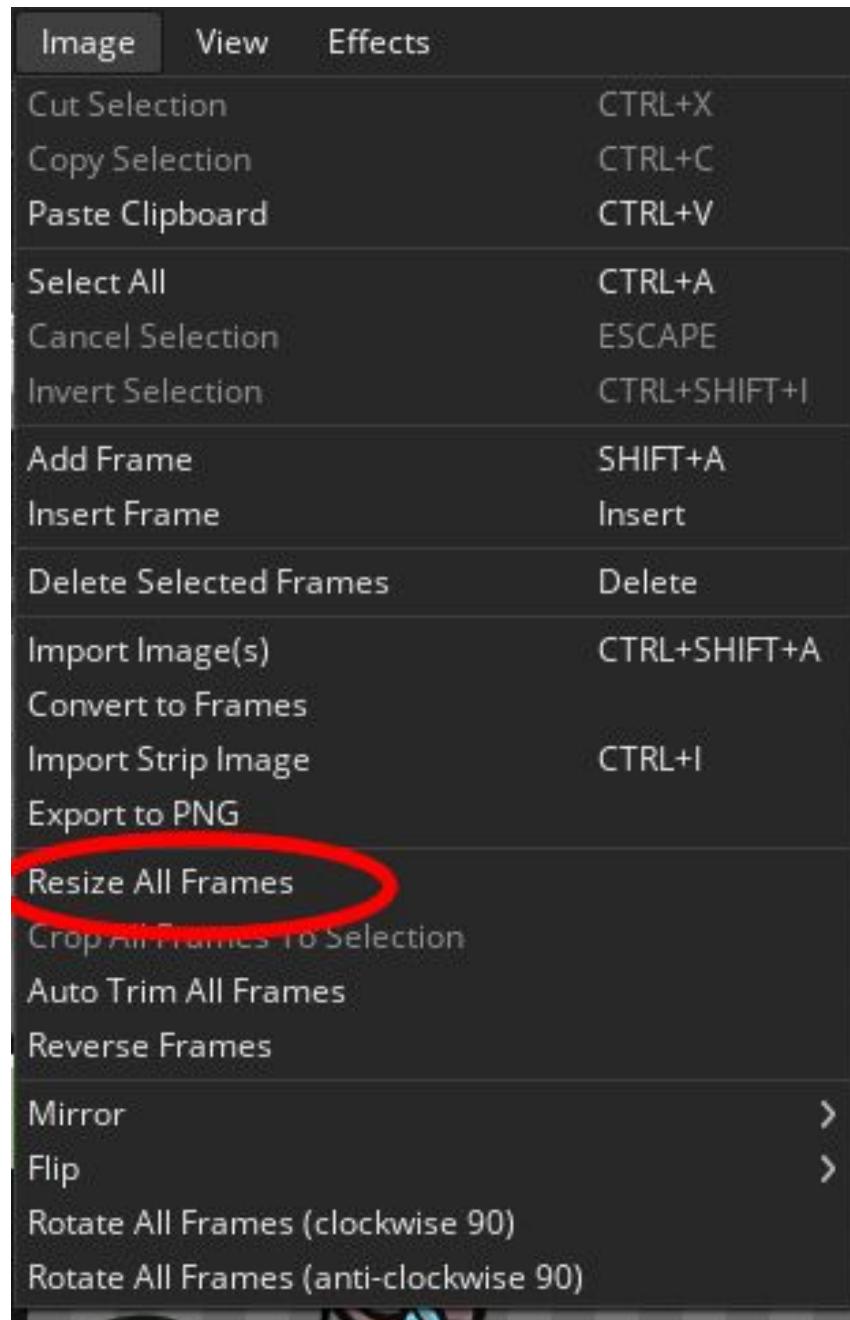


Figure 5_14: *resize all frames* option

Figure 5_15 shows an example setting for resizing all frames:

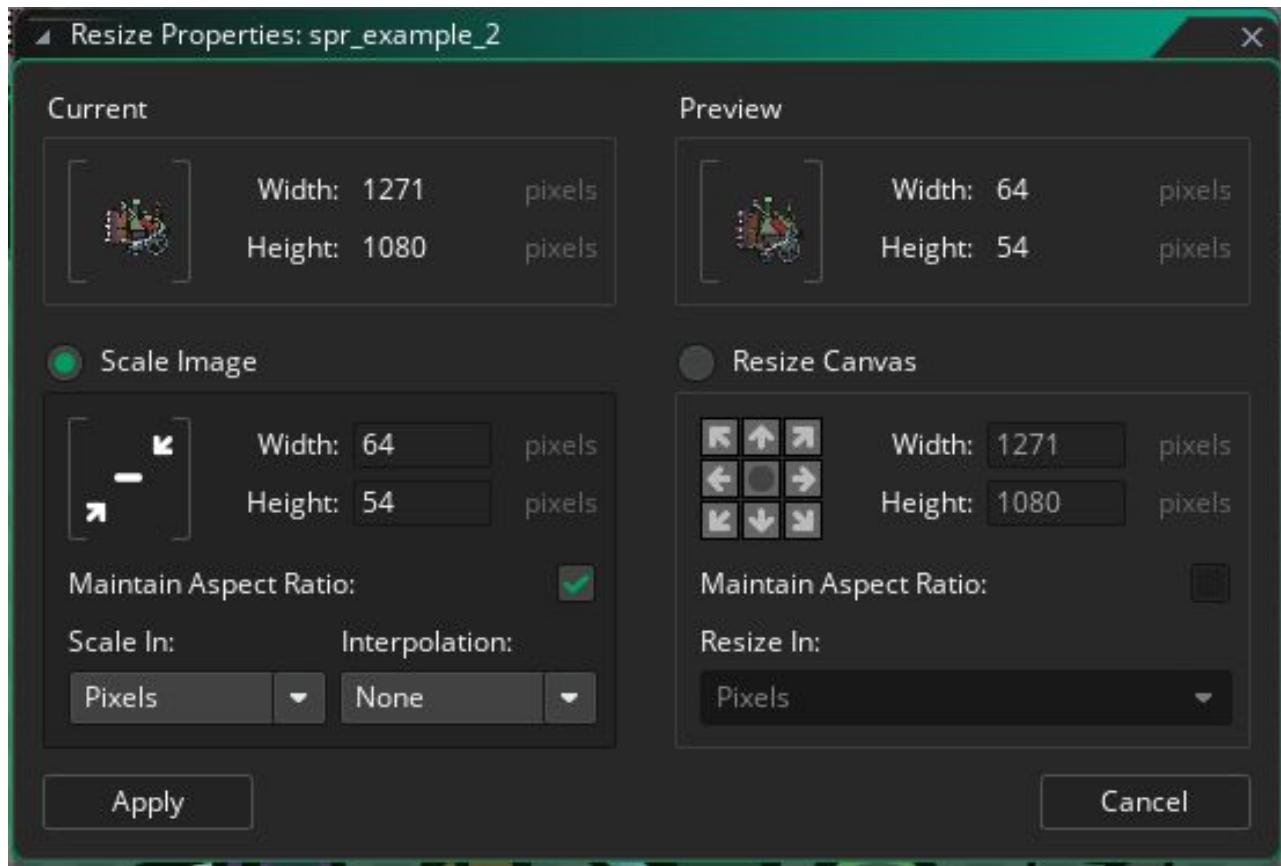


Figure 5_15: Resizing frames

Other things you might want to do with a sprite, using GameMaker Studio 2's built in editor. These are actions I use often in my game projects. *Figure 5_16* below shows a sprite without any changes:



Figure 5_16: Showing sprite without changes

If you want an object moving in the opposite direction you can mirror:

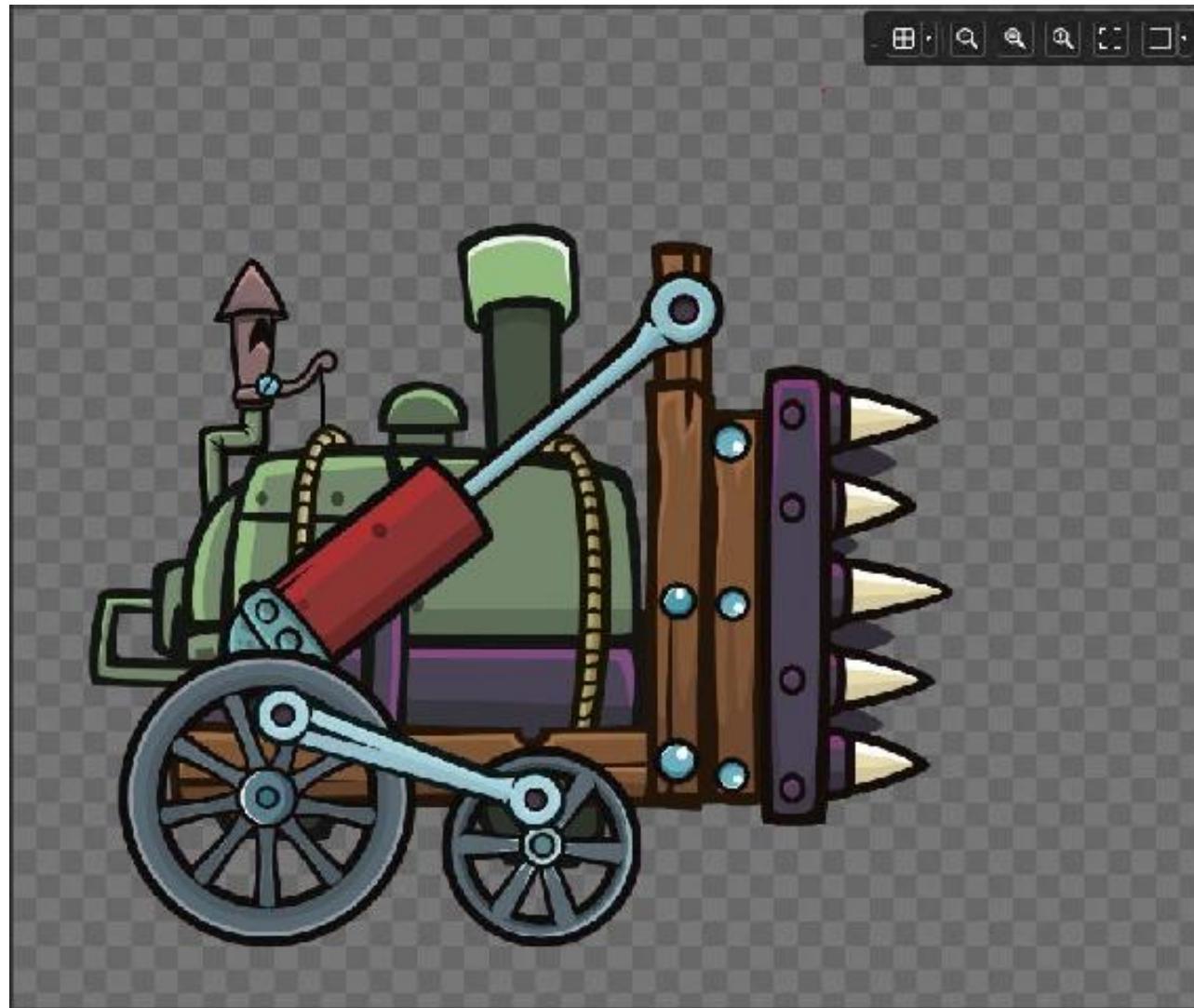


Figure 5_16: Showing sprite mirrored

You may want to gray out a sprite, for example when an upgrade is not available:

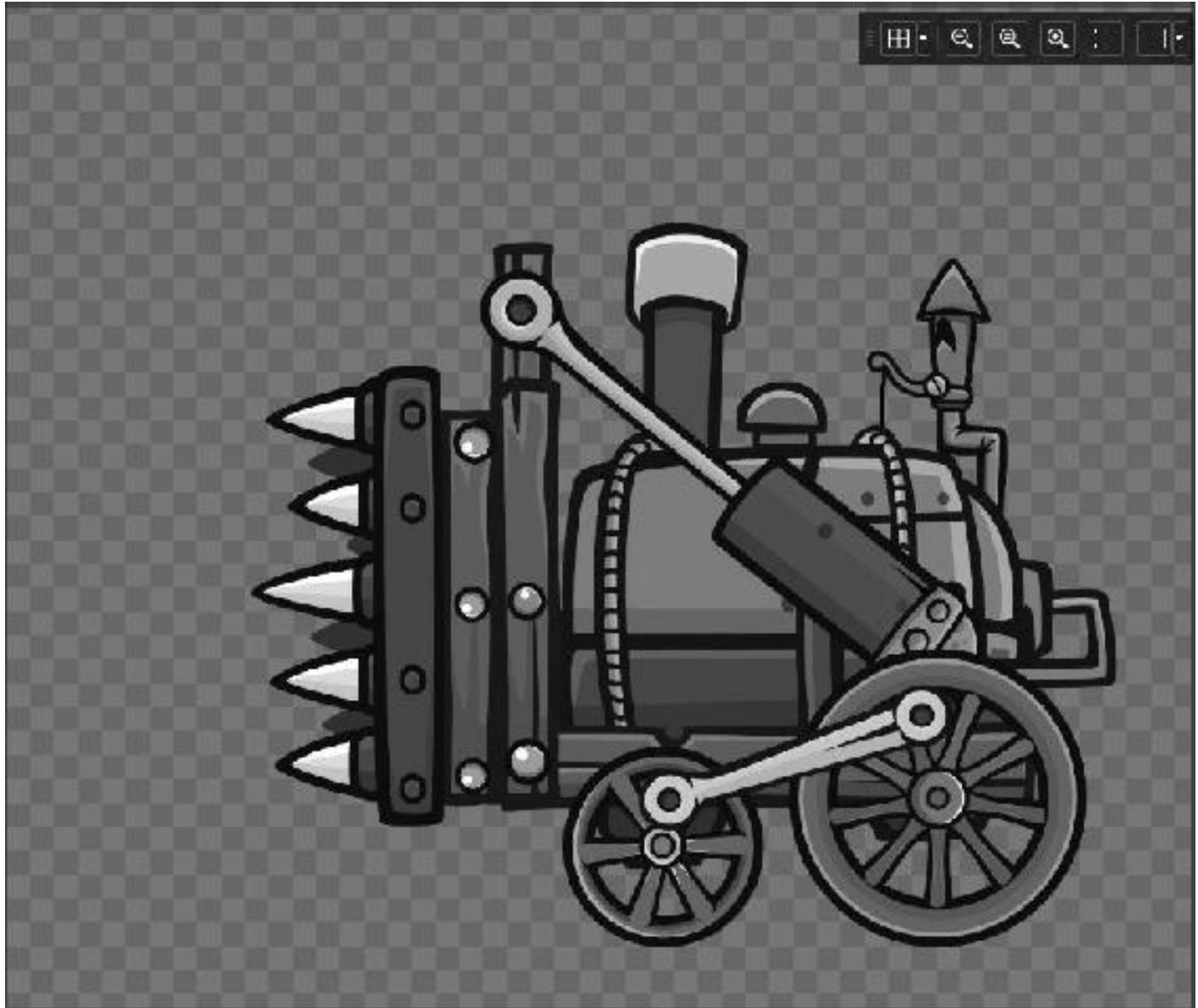


Figure 5_17: Sprite with gray scale

Motion blur can create a cool effect:

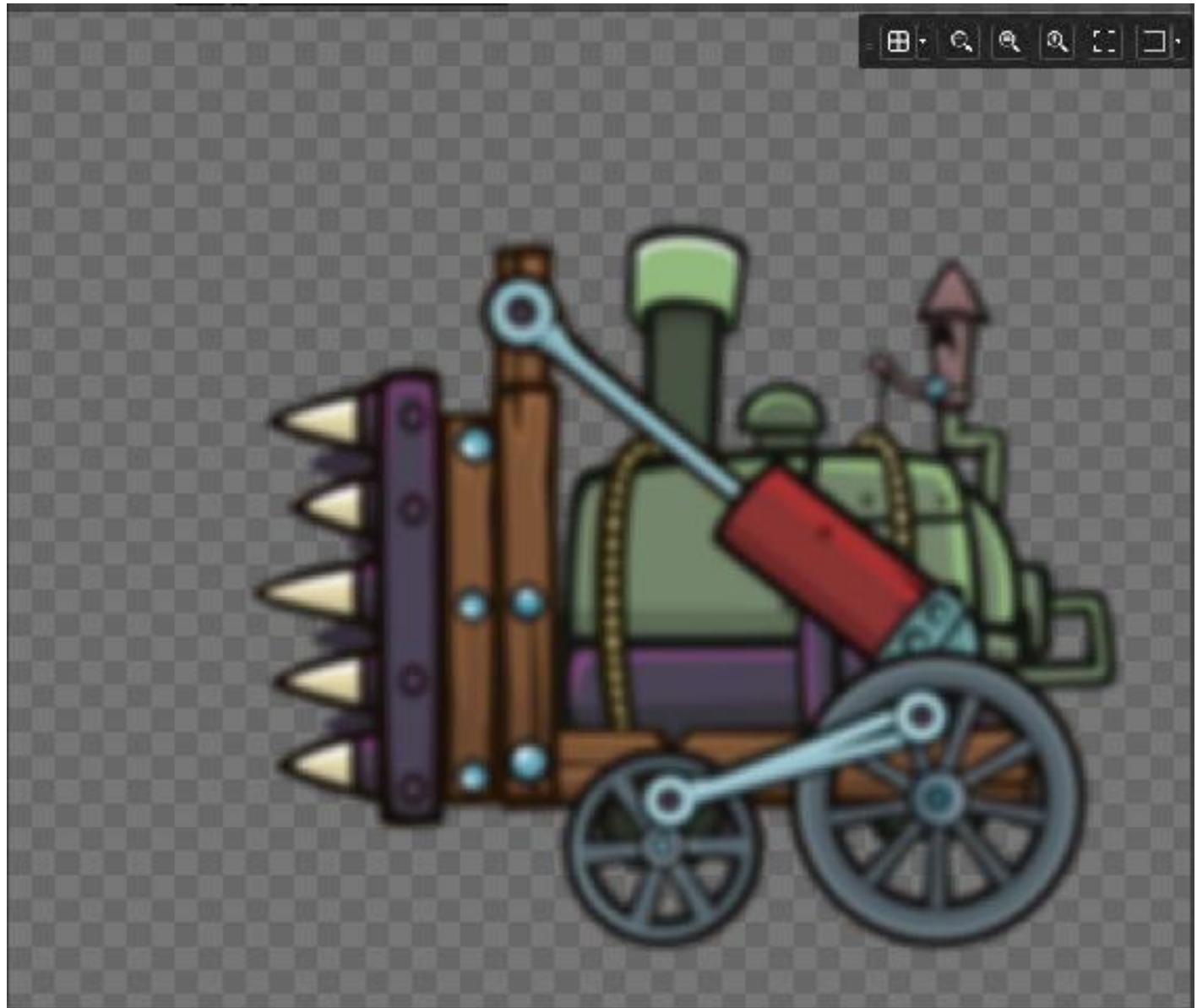


Figure 5_18: Showing a blur effect

If you have multiple similar objects, you want to change some colours so the player can see the difference, as shown in *Figure 5_19*:

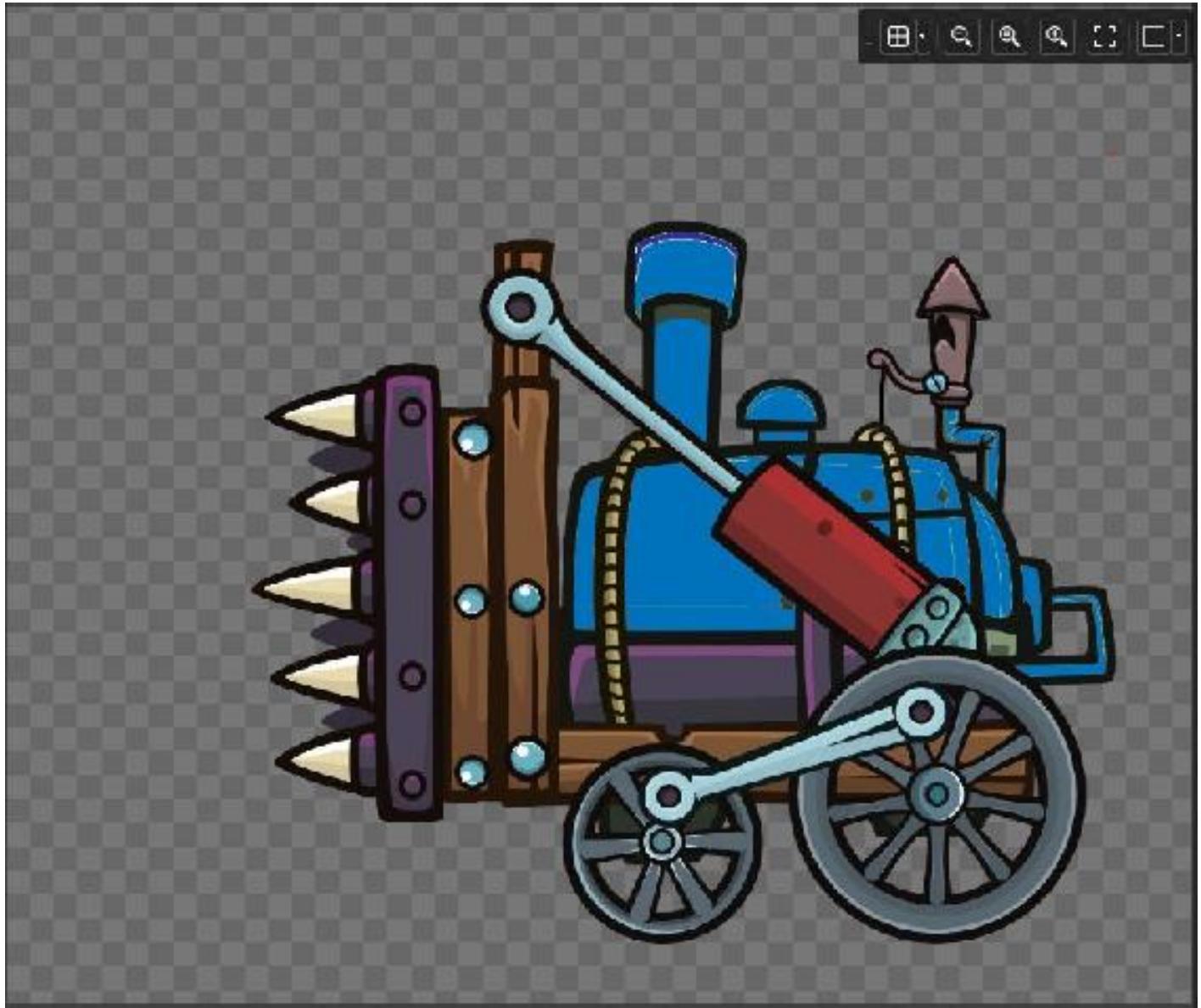


Figure 5_19: Showing some colour changed

Audio

As with graphics, audio will sometimes need a bit of tweaking. For this chapter I have used audio from SoundImage.org and am using Audacity as the audio editor (see Chapter 3 for my thoughts on Audacity).

Note: The wav file format should be used for sound effects, as it plays almost instantaneous. MP3 or OGG format should be used for audio such as music, as it requires some extra processing.

Sometimes when you get a sound asset there is a short bit of silence before the actual sound playing, as shown in *Figure 5_20*. If this was sound effect used in game, this short bit of silence would mean a delay, so would not sync with something else in your game, such as a sound effect.

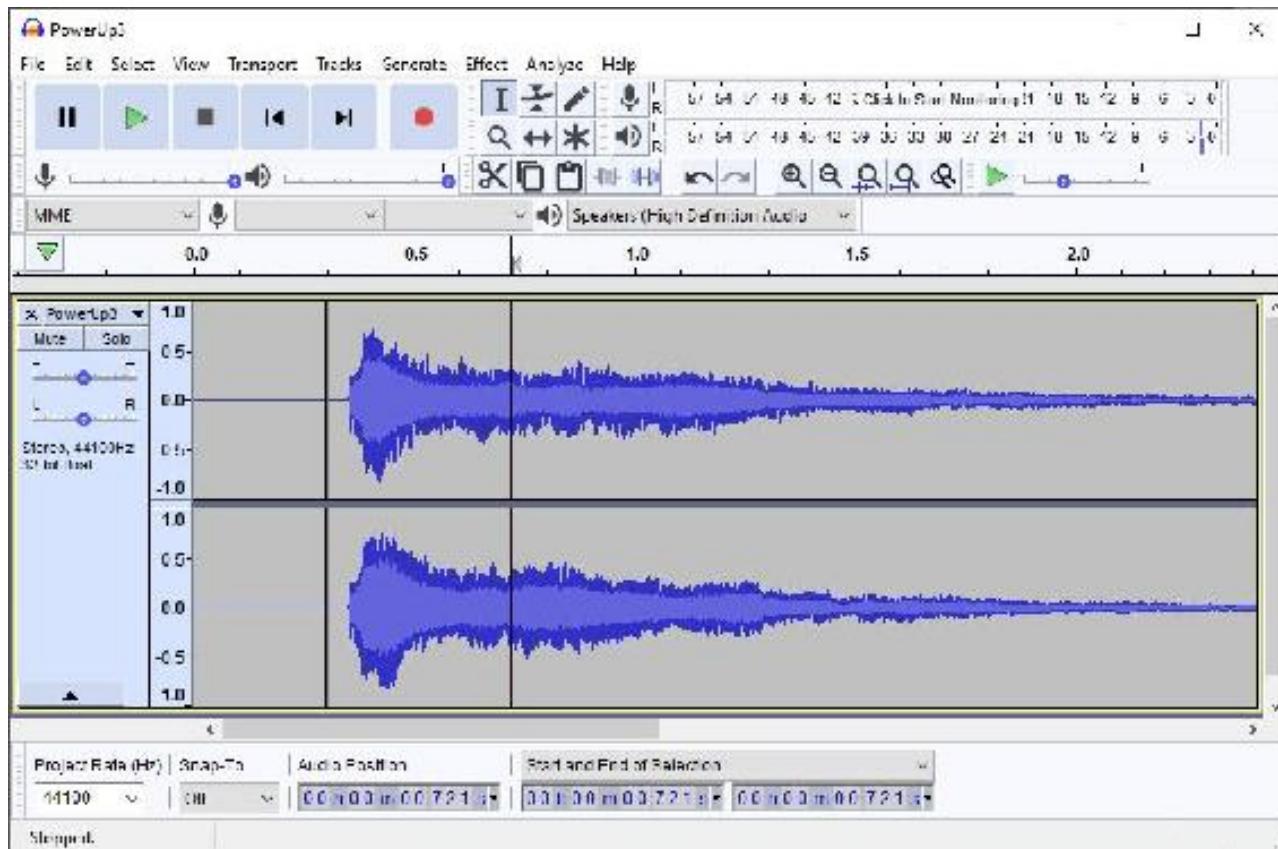


Figure 5_20: Showing silence before sound

This can easily be fixed by selecting the silent area, as shown in *Figure 5_21*:

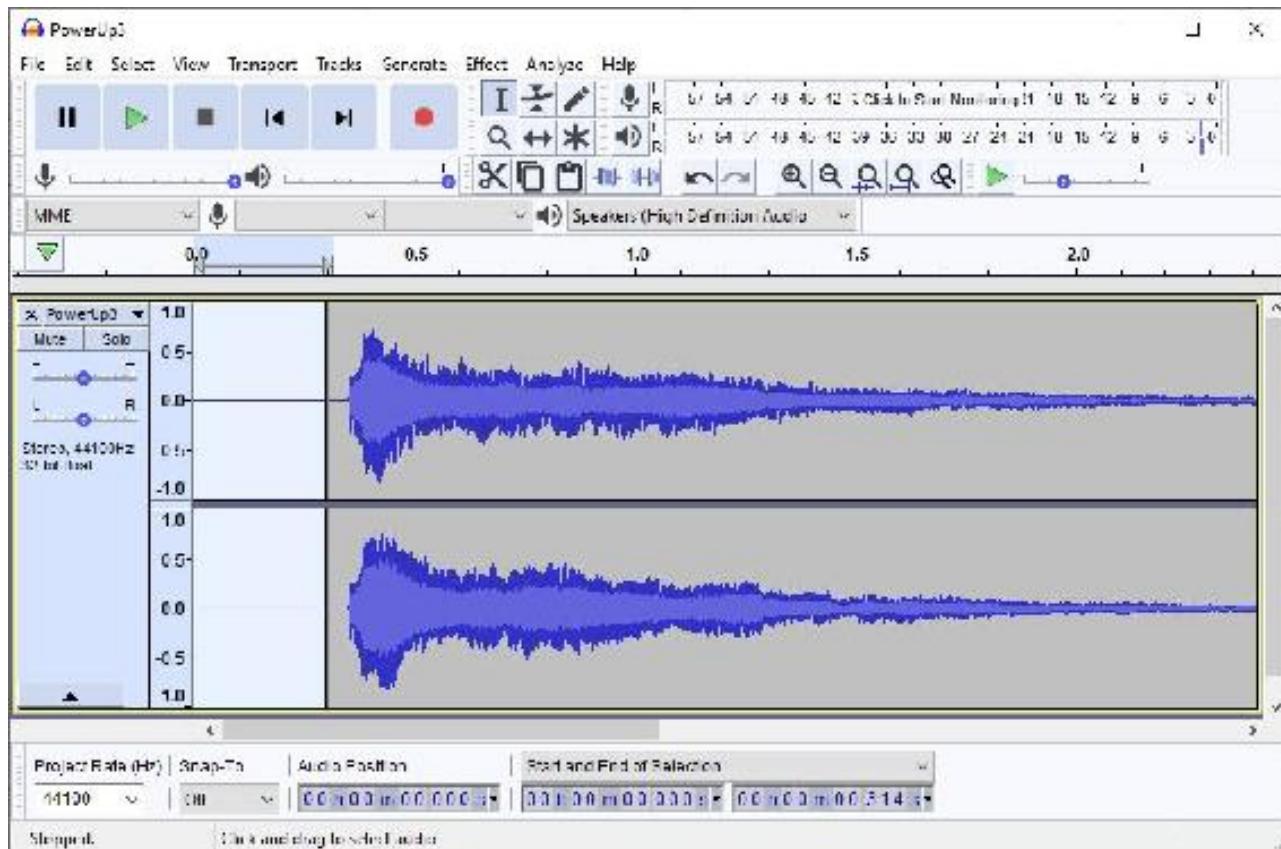


Figure 5_21: Selecting part of the audio

Just hit delete and it is magically removed:

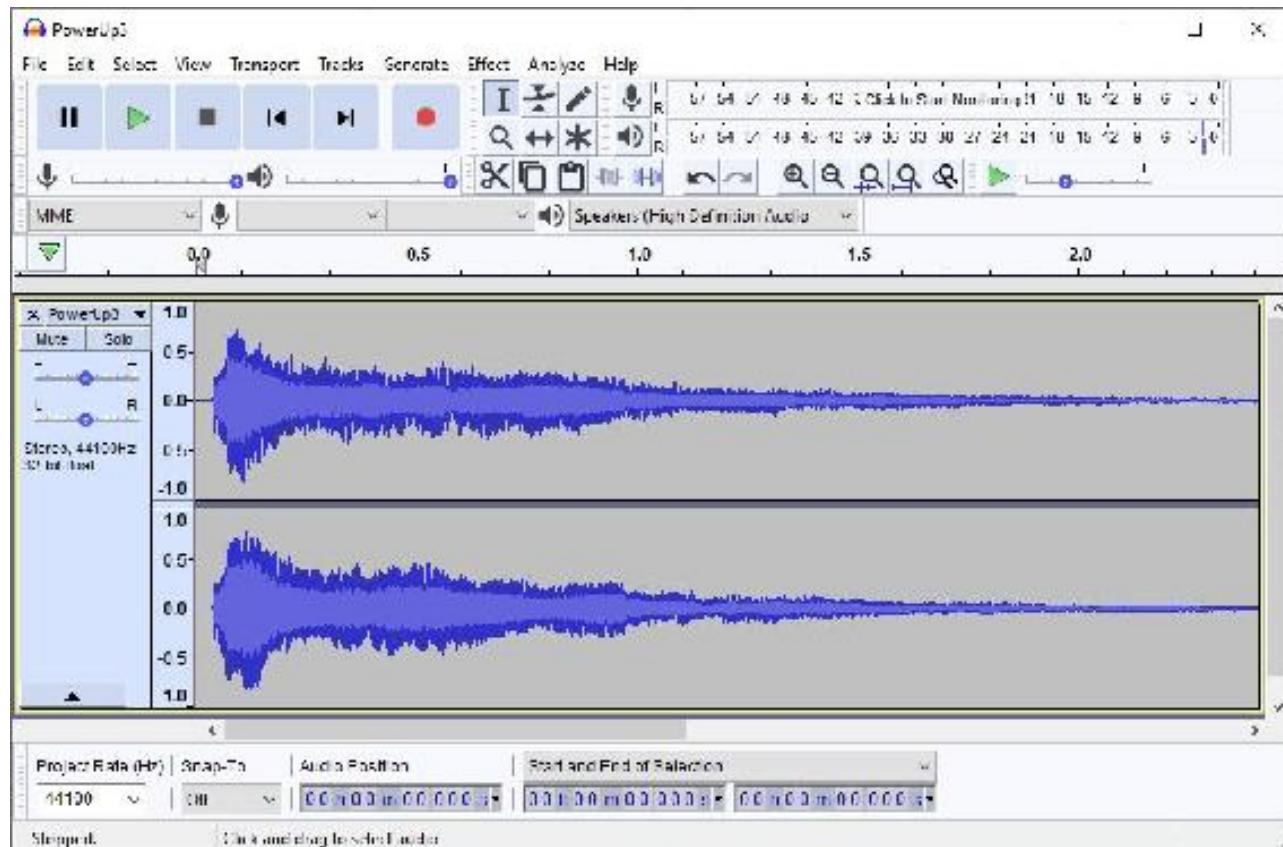


Figure 5_22: Showing edited sound

Figure 5_23 below shows some music audio:

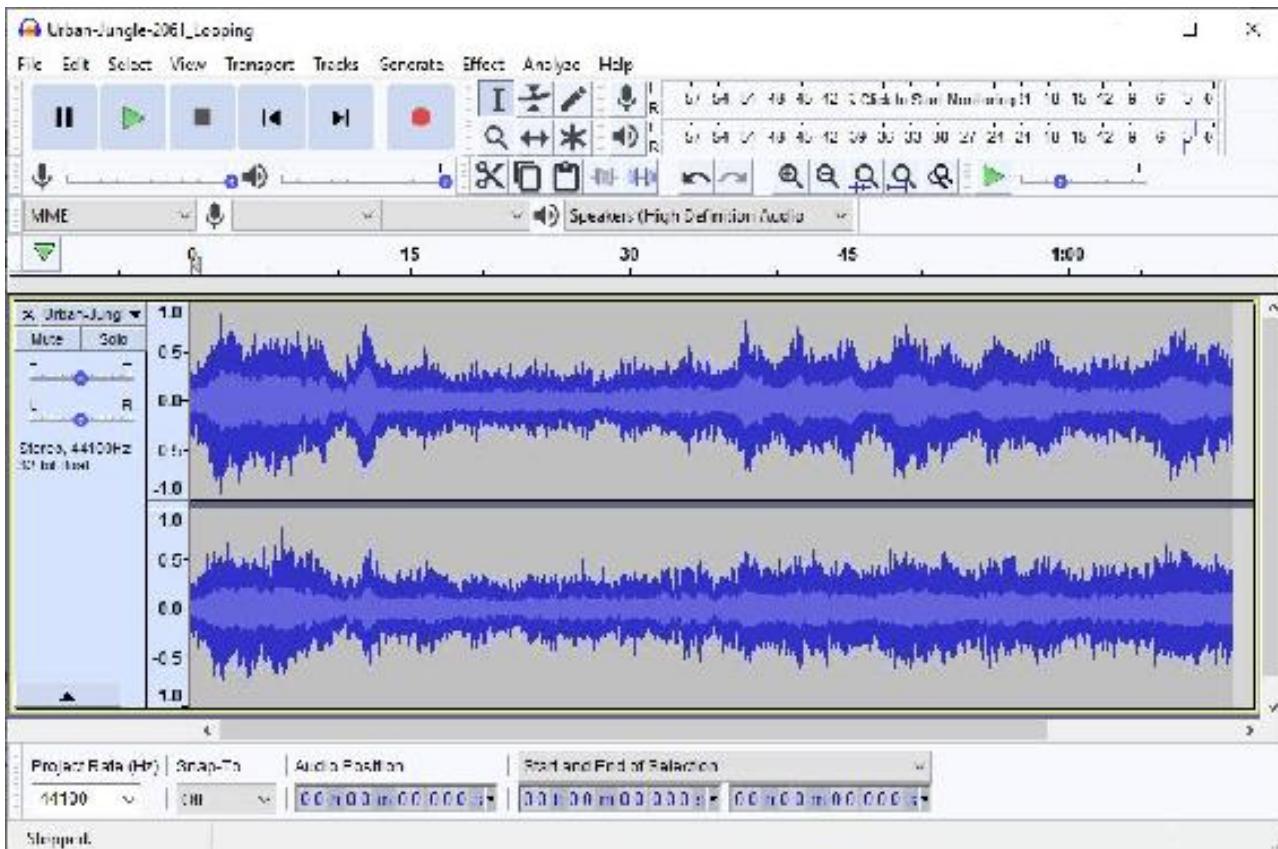


Figure 5_23: Showing an audio track

Sometimes music may be too long, so can select and delete part of the track if you wish:

Figure 5_24 shows the track clipped:

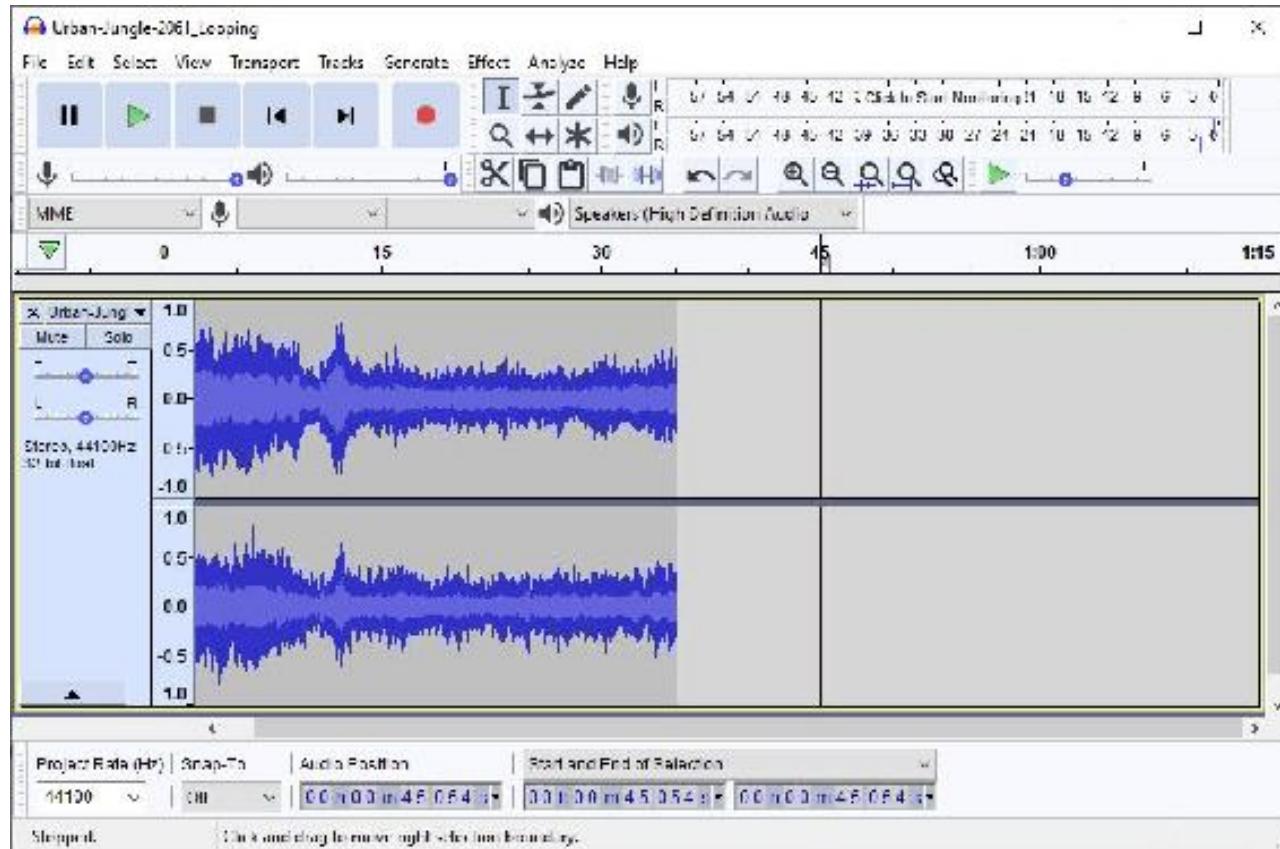


Figure 5_24: Track with part removed

This ending of this music will end quite abruptly. You can select part of the track, then fade out from the effects option, this is how the above would look after fading:

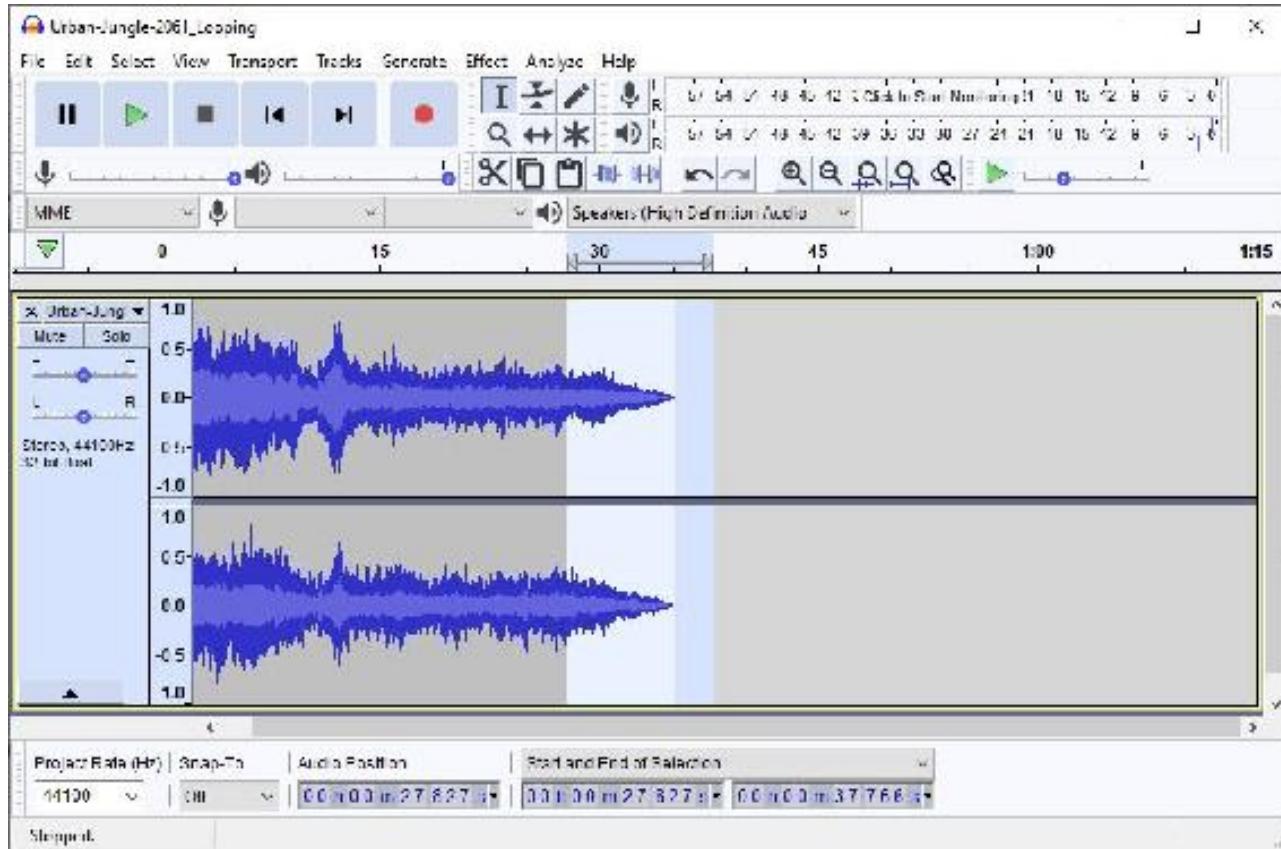


Figure 5_25: Showing clipped track with fading

Audacity has quite an assortment of effects you can apply to your music or audio. My personal favourites are echo and distortion

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 unlikely to go error free on the first attempt. Beta testing allows you to find errors, issues and problems and act upon 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.

Here is screen shot of the game room before changes have been made:



Figure 6_1: Showing game before beta testing feedback

This is a summary of the feedback I received from my Beta Testers:

- Graphics are too large
- Not responsive enough
- Too easy
- Player weapons are too slow
- HUD is not in keeping with the rest of the game
- Aspect ratio should be changed
- Collision masks are imprecise
- GUI life and dots not in keeping with game style
- To visually show damage to player

The following shows how I worked on and addressed the issues above. It's not totally inclusive, but does show the process of working with the given feedback.

Graphics are too large

One piece of feedback was that sprites are too large. Let's address that.

Figure 6_2 below shows the main player sprite setting:

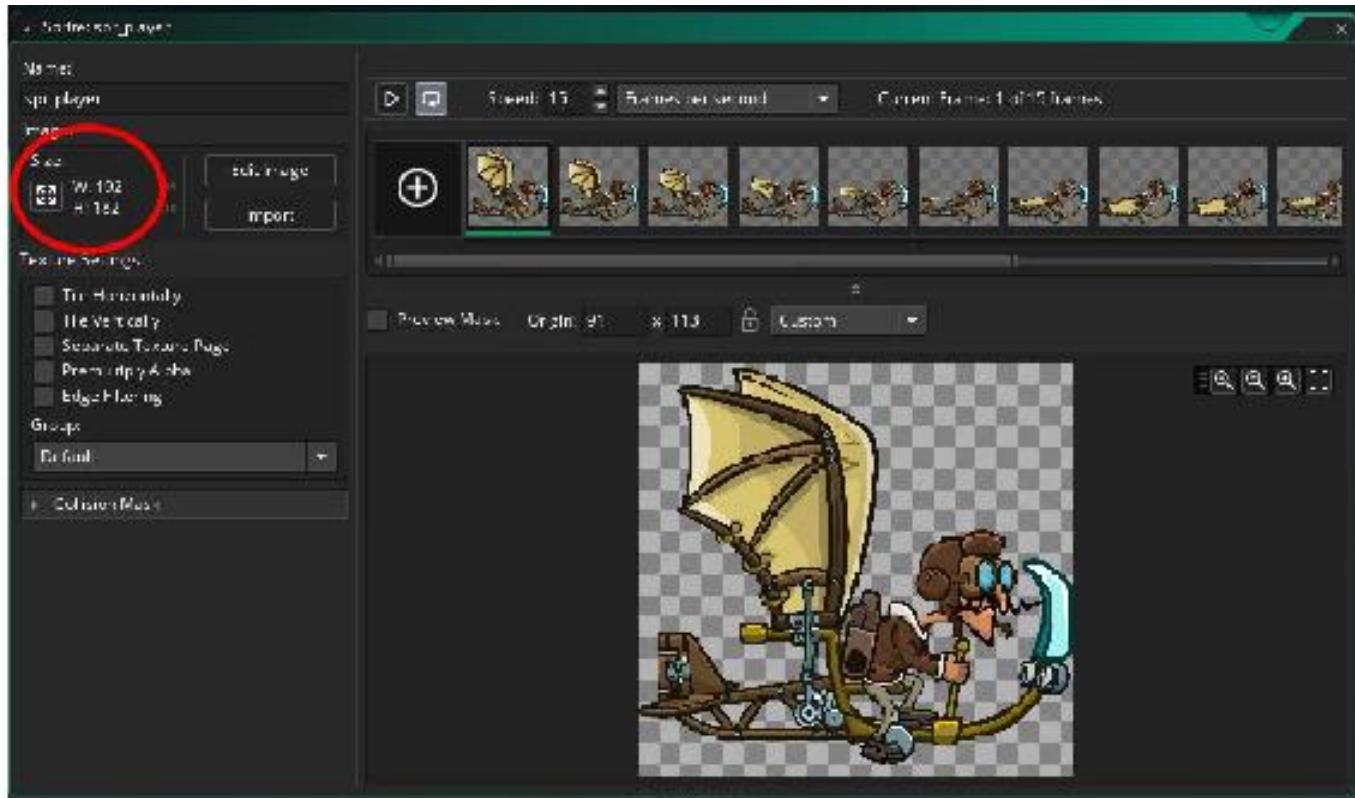


Figure 6_2: Showing current player sprite

The current size is 192x182. We'll reduce the size by 50%. We can click Edit Image and Resize All Frames, as shown in *Figure 6_3*:

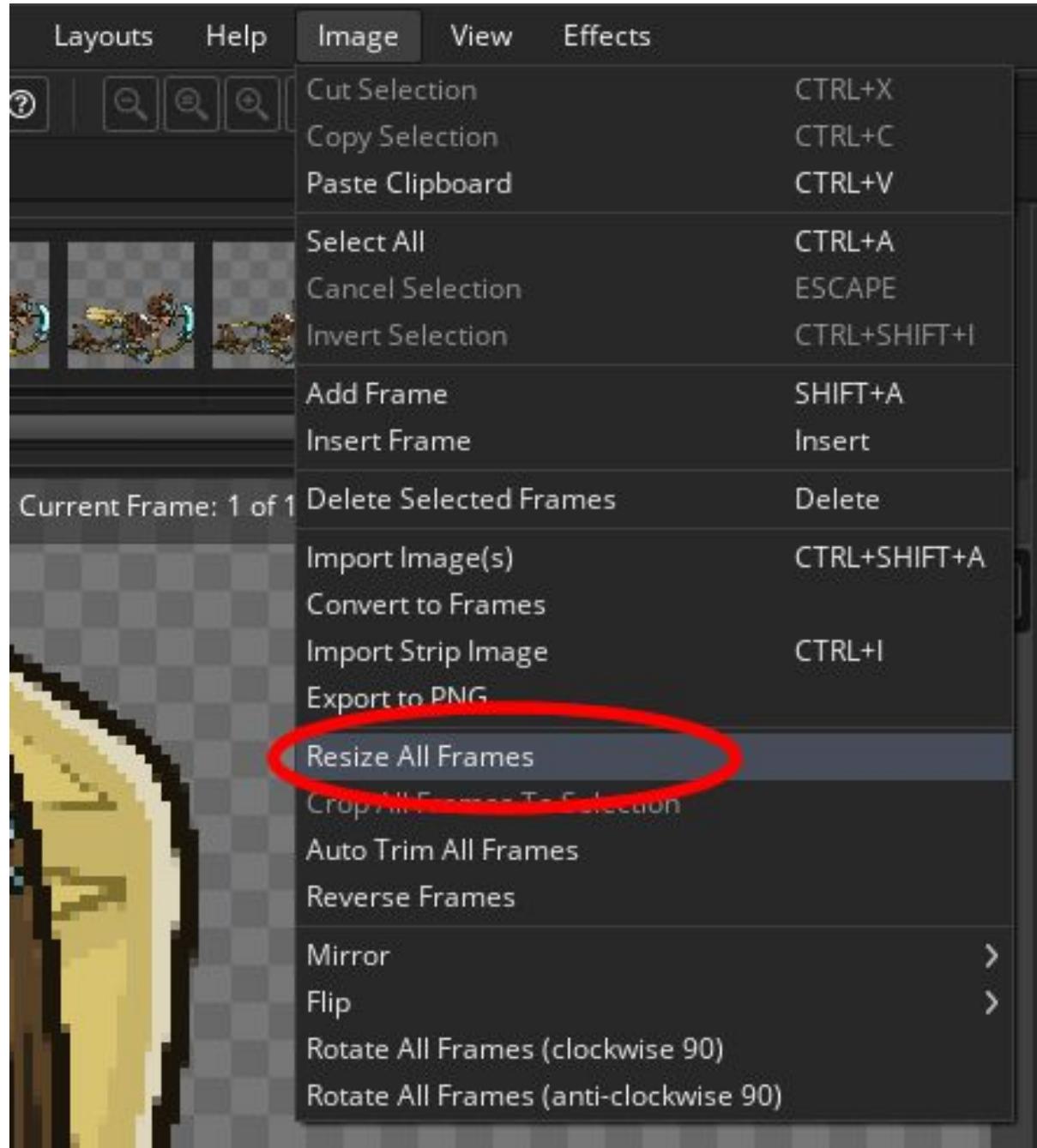


Figure 6_3: Resize all frames

Figure 6_4 shows the settings needed to reduce the size by 50%:

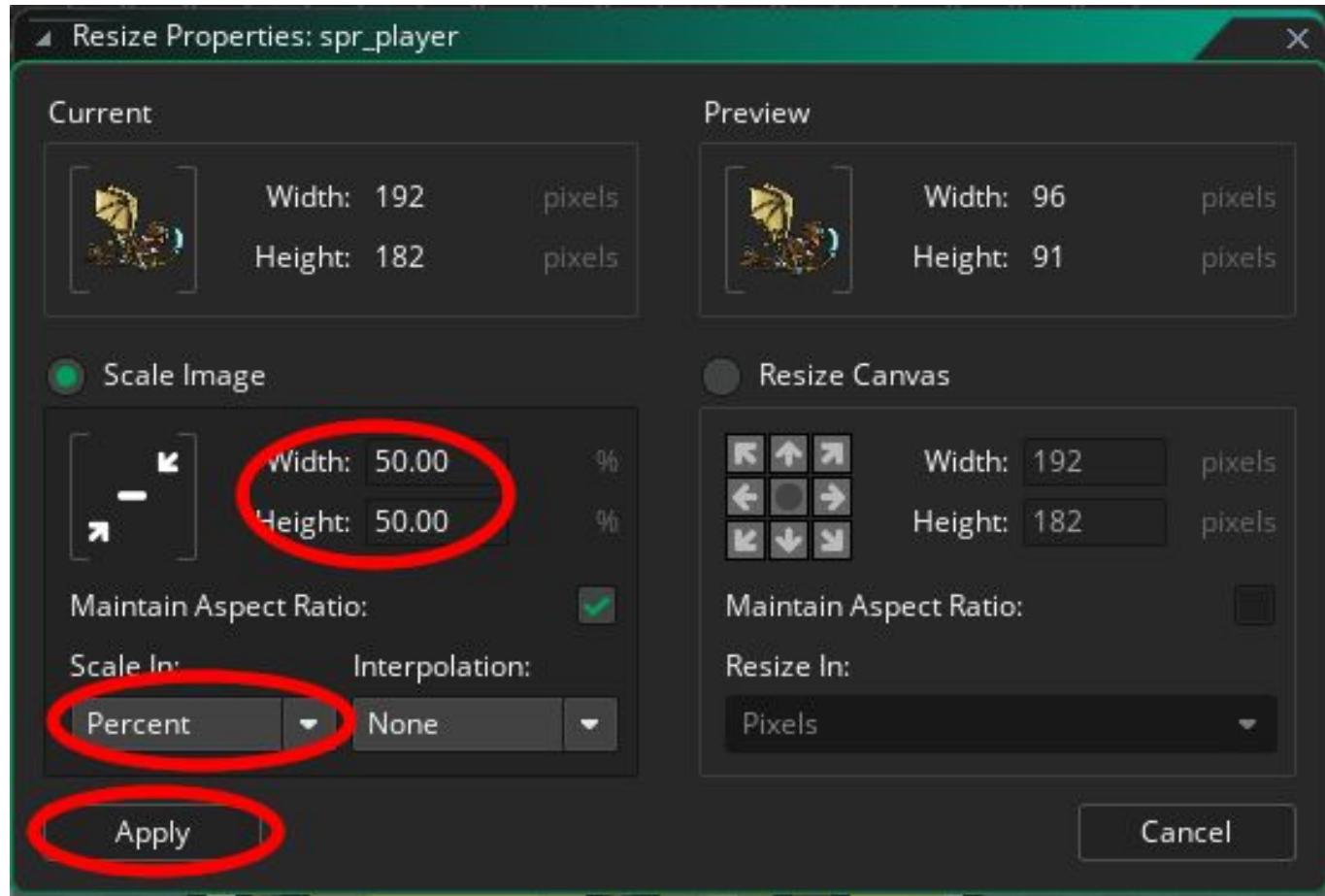


Figure 6_4: Reducing size by 50%

We will repeat the process with all the game level sprites.

Because the sprites have changed size, we will also update the sprite origin of all sprites as we go.

You should now test the game and ensure everything works as planned. You will probably find that a few tweaks are needed here and there, namely check that the collision masks cover the correct location and size, and that spawning of weapons takes into the reduced sizes.

Figure 6_5 shows the game with all sprites reduced by 50%:



Figure 6_5: Showing game with updated sprite sizes

Not responsive enough

Another piece of feedback was that player movement was a bit sluggish.

The current movement code is shown below:

```
//keep in screen
y=clamp(y,80,room_height-80);
//move
//up down
nokey=keyboard_check(ord("W"))+keyboard_check(or
d("S"));
if nokey==0
{
    if y<flying_level y+=2;
    if y>flying_level y-=2;
}
//forward back
if keyboard_check(ord("D"))
{
    x+=4;
}
if keyboard_check(ord("A"))
{
    x-=4;
}
x=clamp(x,xx-40,xx+180);
nokey=keyboard_check(ord("A"))+keyboard_check(or
d("D"));
if nokey==0
```

```

{
  if x<xx x+=2;
  if x>xx x-=2;
}

if keyboard_check(ord("W"))
{
  y-=4;
}
if keyboard_check(ord("S"))
{
  y+=4;
}

```

We'll make it 50% more responsive by changing the code to that below. **Code in red** has been changed:

```

//keep in screen
y=clamp(y,80,room_height-80);
//move
//up down
nokey=keyboard_check(ord("W"))+keyboard_check(ord("S"));
if nokey==0
{
  if y<flying_level y+=3;
  if y>flying_level y-=3;
}
//forward back
if keyboard_check(ord("D"))

```

```
{  
    x+=6;  
}  
if keyboard_check(ord("A"))  
{  
    x=6;  
}  
x=clamp(x,xx-40,xx+180);  
nokey=keyboard_check(ord("A"))+keyboard_check(or  
d("D"));  
if nokey==0  
{  
    if x<xx x+=3;  
    if x>xx x-=3;  
}  
  
if keyboard_check(ord("W"))  
{  
    y=6;  
}  
if keyboard_check(ord("S"))  
{  
    y+=6;  
}
```

A quick test shows a marked improvement in player's control.

Too easy

Other feedback was that the game was too easy. Let's address the underlying reasons for that:

- Enemy hp too low
- Enemy projectiles too slow

The original code for setting the ground enemy hp was:

`start_hp=global.level;`

Let's change that to:

`start_hp=global.level*global.level;`

and change the code for the flying enemy to:

`start_hp=global.level*global.level*2;`

and for the boss enemy to:

`start_hp=global.level*global.level*5;`

We'll make the enemy weapons move and fire faster.

We'll start with the flying enemy, the code in the Alarm 0 Event was:

```
alarm[0]=room_speed*4;  
arrow=instance_create_layer(x,y,"Enemy_Missile",obj_enemy_arrow_1);  
ang=point_direction(x,y,obj_player.x,obj_player.y);  
arrow.direction=ang;  
arrow.image_angle=ang;  
arrow.speed=10;  
arrow.strength=global.level;
```

We'll change that to:

`alarm[0]=room_speed*3;`

```
arrow=instance_create_layer(x,y,"Enemy_Missile",obj_
enemy_arrow_1);
ang=point_direction(x,y,obj_player.x,obj_player.y);
arrow.direction=ang;
arrow.image_angle=ang;
arrow.speed=12;
arrow.strength=global.level;
```

This is increase the shooting frequency and projectile speed.

Similarly we'll change the shooting code for the boss enemy from:

```
alarm[0]=room_speed*3;
ball=instance_create_layer(x,y,"Enemy_Ball",obj_ball)
;
ball.direction=image_angle;
ball.speed=8;
```

to:

```
alarm[0]=room_speed*2;
ball=instance_create_layer(x,y,"Enemy_Ball",obj_ball)
;
ball.direction=image_angle;
ball.speed=12;
```

A quick play through of a few levels, shows a marked increase in difficulty.

Player weapons are too slow

We can increase the shoot rate and projectile speed with just a few code changes.

The current code for Global Mouse Left Pressed is:

```
if can_shoot_1==false exit;  
var xx = x + lengthdir_x(64, image_angle);  
var yy = y + lengthdir_y(64, image_angle);  
missile=instance_create_layer(xx,yy,"Missiles",obj_player_rocket);  
missile.image_angle=image_angle;  
missile.direction=image_angle;  
missile.speed=4;  
can_shoot_1=false;  
alarm[1]=room_speed*2;
```

We'll change that to:

```
if can_shoot_1==false exit;  
var xx = x + lengthdir_x(64, image_angle);  
var yy = y + lengthdir_y(64, image_angle);  
missile=instance_create_layer(xx,yy,"Missiles",obj_player_rocket);  
missile.image_angle=image_angle;  
missile.direction=image_angle;  
missile.speed=6;  
can_shoot_1=false;  
alarm[1]=room_speed*1.5;
```

And the code for Global Right Mouse Pressed from:

```
if global.level<13 exit;  
if can_shoot_2==false exit;
```

```
alarm[1]=room_speed;
show_debug_message("Missile 2");
var xx = x + lengthdir_x(64, image_angle);
var yy = y + lengthdir_y(64, image_angle);
missile=instance_create_layer(xx,yy,"Missiles",obj_player_missile);
missile.image_angle=image_angle;
missile.direction=image_angle;
missile.speed=8;
can_shoot_2=false;
alarm[2]=room_speed*2;
```

to:

```
if global.level<13 exit;
if can_shoot_2==false exit;
alarm[1]=room_speed;
show_debug_message("Missile 2");
var xx = x + lengthdir_x(64, image_angle);
var yy = y + lengthdir_y(64, image_angle);
missile=instance_create_layer(xx,yy,"Missiles",obj_player_missile);
missile.image_angle=image_angle;
missile.direction=image_angle;
missile.speed=10;
can_shoot_2=false;
alarm[2]=room_speed*1.5;
```

The code for weapon 1 and 2 addon for top and bottom Alarm Event code is:

```
alarm[0]=
(room_speed*3)+room_speed*5/global.shoot_speed;
```

```
ang=image_angle;
xx = x + lengthdir_x(32, ang);
yy = y + lengthdir_y(32, ang);

bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
bullet.direction=ang;
bullet.speed=5;
bullet.image_angle=ang;
if global.level<6 exit;

bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
bullet.direction=ang-30;
bullet.speed=5;
bullet.image_angle=ang-30;

bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
bullet.direction=ang+30;
bullet.speed=5;
bullet.image_angle=ang+30;
```

We'll change that to:

```
alarm[0]=
(room_speed*1.5)+room_speed*5/global.shoot_speed;
ang=image_angle;
xx = x + lengthdir_x(32, ang);
yy = y + lengthdir_y(32, ang);
```

```
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
bullet.direction=ang;
bullet.speed=7;
bullet.image_angle=ang;
if global.level<6 exit;
```

```
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
bullet.direction=ang-30;
bullet.speed=7;
bullet.image_angle=ang-30;
```

```
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
bullet.direction=ang+30;
bullet.speed=7;
bullet.image_angle=ang+30;
```

Similary we'll change the code for the second weapon, top and bottom from:

```
/// @description Calculate Position
alarm[0]=(room_speed*3)+room_speed*4;
ang=image_angle;
xx = x + lengthdir_x(32, ang);
yy = y + lengthdir_y(32, ang);
```

```
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_2);
```

```
bullet.direction=ang;  
bullet.speed=5;  
bullet.image_angle=ang;
```

to:

```
alarm[0]=room_speed*4;  
ang=image_angle;  
xx = x + lengthdir_x(32, ang);  
yy = y + lengthdir_y(32, ang);
```

```
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_2);  
bullet.direction=ang;  
bullet.speed=7;  
bullet.image_angle=ang;
```

A quick play through for a few minutes, shows a marked improvement.

HUD is not in keeping with the rest of the game

We'll add a background for the HUD and change the font used for HUD.

A quick look on **GameDeveloperStudio.com** finds a suitable sprite, which matches in with the menu room.

I found suitable sprite, Papyrus

The updated HUD code now looks like this:

```
/// @description Draw HUD
draw_set_font(font_menu);
draw_set_colour(c_black);
draw_set_halign(fa_center);
draw_set_valign(fa_middle);
draw_set_alpha(0.5);
draw_sprite(spr_hud,0,0,0);
draw_text(200,40,"Score "+string(score));
draw_text(200,80,"Highscore
"+string(global.highscore));

draw_text(500,40,"Level "+string(global.level));
draw_text(500,80,"Power "+string(global.pow));

if obj_player.shield>0
{
    draw_text(800,40,"Shield Active");
}
else
{
```

```
    draw_text(800,40,"No Shield");  
}  
  
draw_text(800,80,"Health "+string(health));  
  
draw_set_alpha(1);
```

The game after applying, makes the game level look like the screen shot shown in *Figure 6_6*:



Figure 6_6: Showing updated HUD

Aspect ratio should be changed

Another bit of feedback from my beta testers was the screen ratio.

We'll make some changes to make the game more widescreen. I quick search on the net told me that the average minimum screen resolution width is 1366, so we'll change the window size to 1300 to accommodate this.

The steps for doing this:

Change the room size for all rooms, so they have a width of 1300

Resize sprite for in game HUD

Change drawing locations of text for in game HUD

After updates, the game room looks like that in *Figure 6_7*:

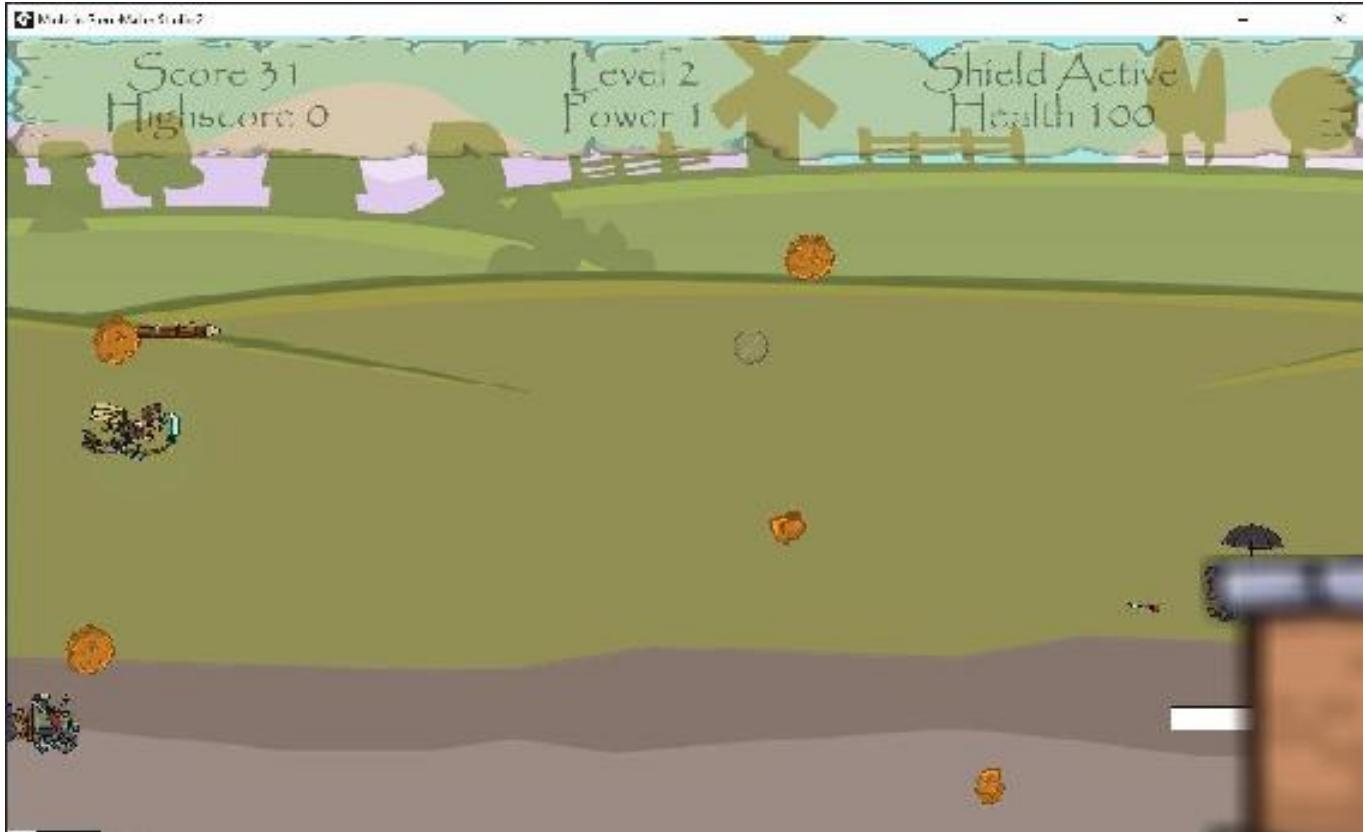


Figure 6_7: Showing game with new screen ratio

Collision masks are imprecise

Figure 6_8 shows a sprite collision mask before changes:

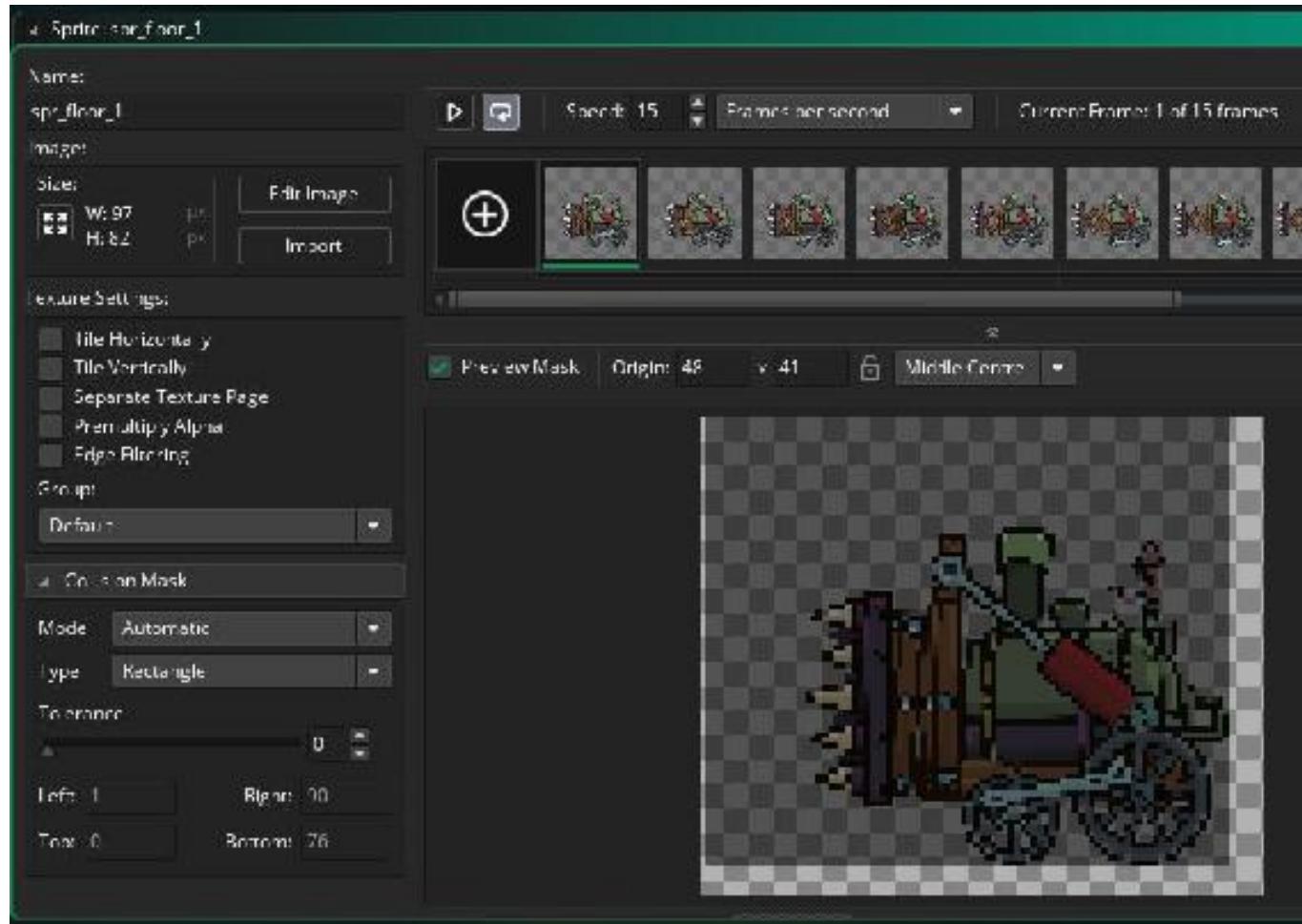


Figure 6_8: Showing default mask settings

The problem with this is obvious, it will detect a collision well to the left of the sprite's actual image. Due to the fact the sprite changes size with each subimage, we'll change the collision setting to use precise per frame setting as shown in *Figure 6_9*:

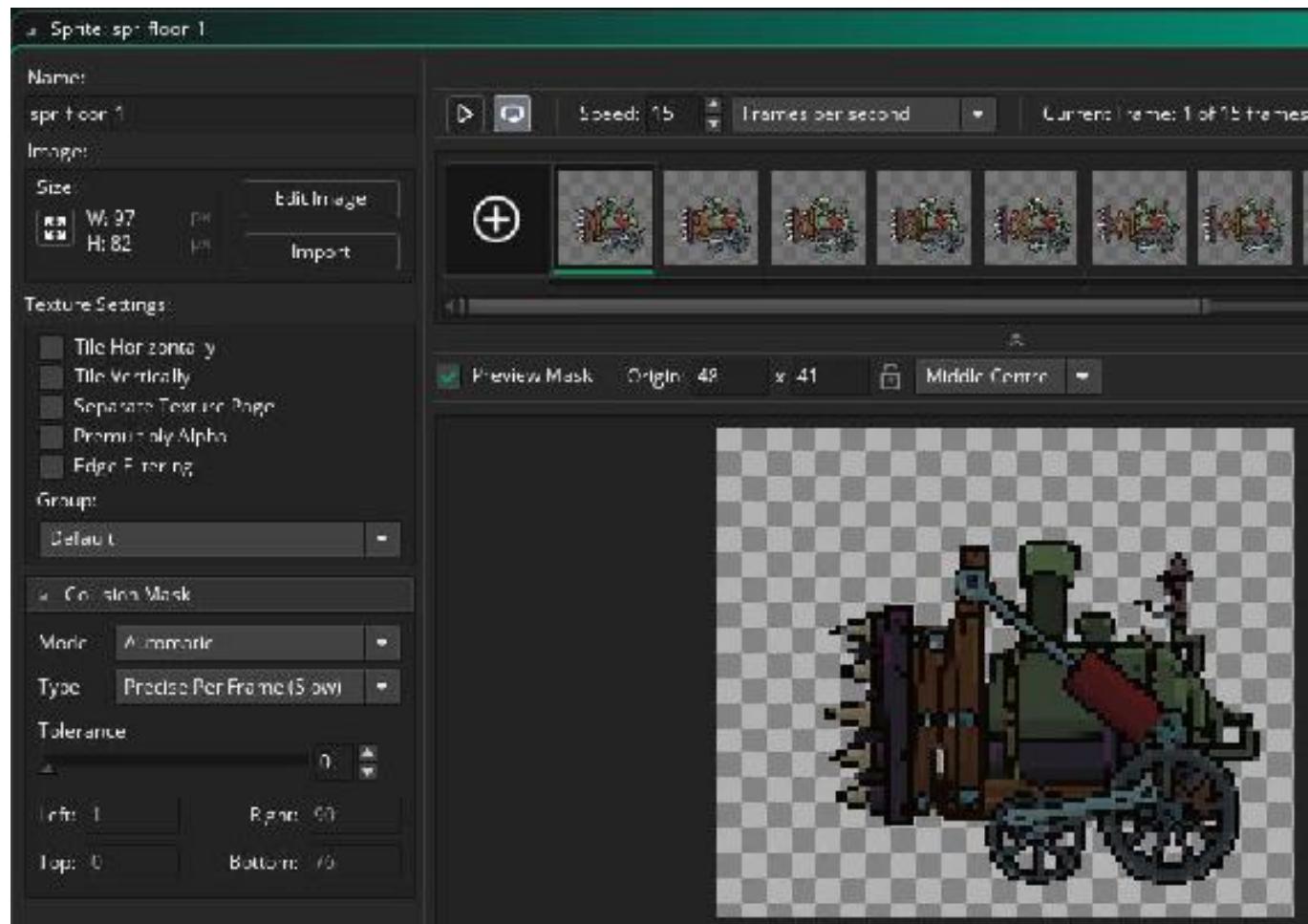


Figure 6_9: Showing new mask settings

Using this collision masking too often can slow down your game, so should only be used when needed. We'll also use this for the player's sprite mask, as it also changes size each frame.

Figure 6_10 shows an updated mask, using a rectangle, which is better than the above option, when you want to set a a collision that doesn't change too much:

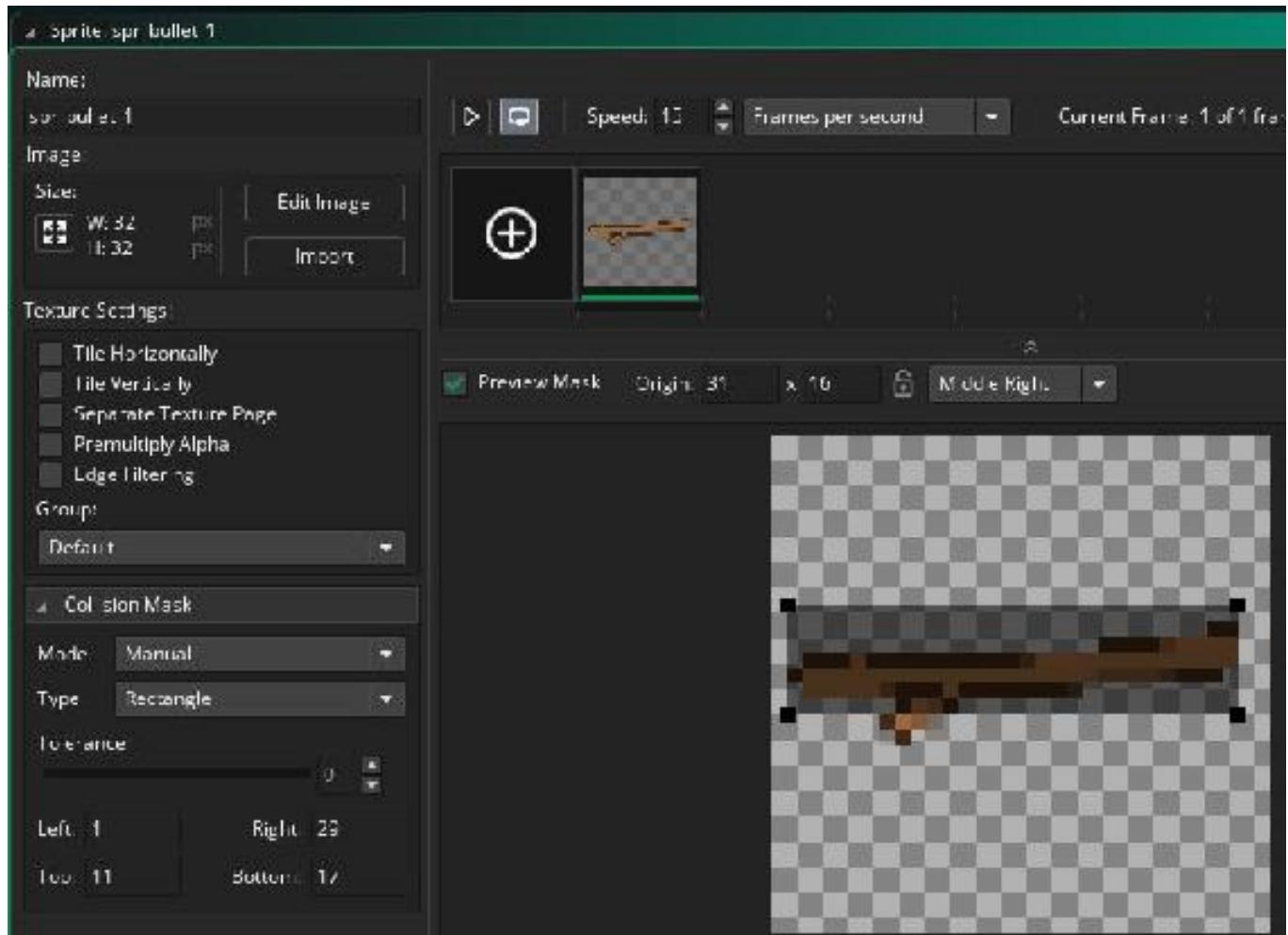


Figure 6_10: Showing a manual rectangle mask
I would now go through and check the masks for all sprites used in the game.

GUI life and dots not in keeping with game style

We'll change the colour of the health bar, and position it in the middle above the enemy. We'll also add a player's health bar at the bottom of the game window. See the programming section for the relevant code. The dot, used for detecting player hitting target area of the boss enemy, can just be made invisible.

To visually show damage to player

We'll make the player flash red when it receives any damage. We'll use a similar system to that which was used for the boss enemy.

Figure 6_11 shows game room with changes made:



Figure 6_11: Showing changes applied from beta testing

Once you've made all the changes based on your feedback, I suggest one further round of beta testing, just to check no bugs remain.

The download has a file **Game_Final**, you can play this to see the finished game.

Debugging

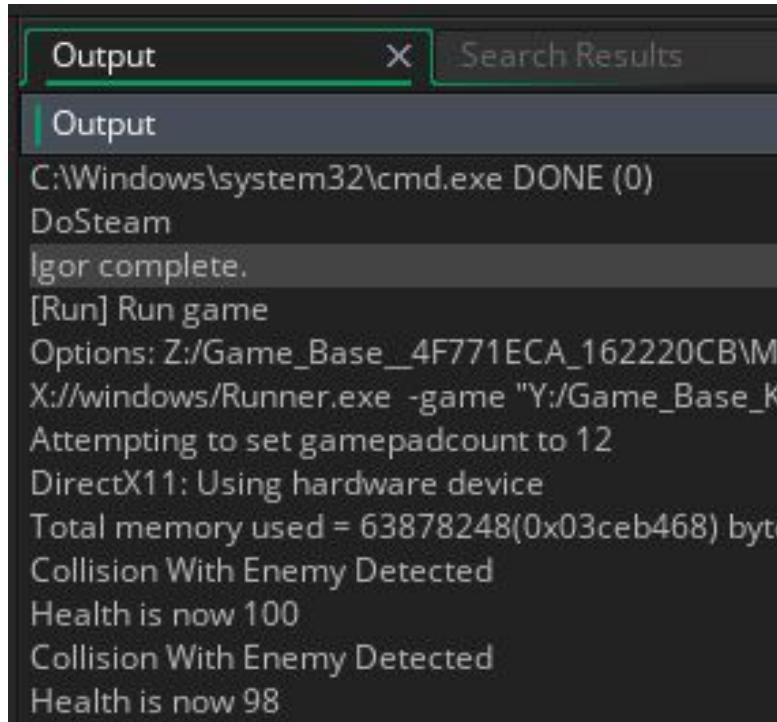
Another stage of development is debugging, which goes hand in hand with writing the code. You can add (and then remove when fixed) code that will display messages in the console window. This could include simple messages such as, “Collision With Enemy Detected”, which the code would be:

```
show_debug_message("Collision With Enemy  
Detected");
```

Or include values or strings of variables, for example the code below:

```
show_debug_message("Health is now "+string(health));
```

Figure 6_12 shows the effect when added to a collision event with an enemy projectile, console window output is shown, the first with shield active so taking no damage, then again without shield showing damage taken:



The screenshot shows the 'Output' tab of the GameMaker Studio 2 interface. The console window displays the following text:

```
C:\Windows\system32\cmd.exe DONE (0)
DoSteam
Igor complete.
[Run] Run game
Options: Z:/Game_Base_4F771ECA_162220CB\M
X://windows/Runner.exe -game "Y:/Game_Base_K
Attempting to set gamepadcount to 12
DirectX11: Using hardware device
Total memory used = 63878248(0x03ceb468) bytes
Collision With Enemy Detected
Health is now 100
Collision With Enemy Detected
Health is now 98
```

Figure 6_12: Showing console window

This is great to use if only test the occasional object instance or event that happens every so often, but if you try to debug lots of things at once, the console window would fill up quicker than you could read it. Fortunately, GameMaker Studio 2 come with quite an impressive debugger. I suggest you spend some time learning the basics of it.

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 magically 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 won't win any awards, but does serve as great introduction. The sketch below shows the ideas for the menu room and game room:

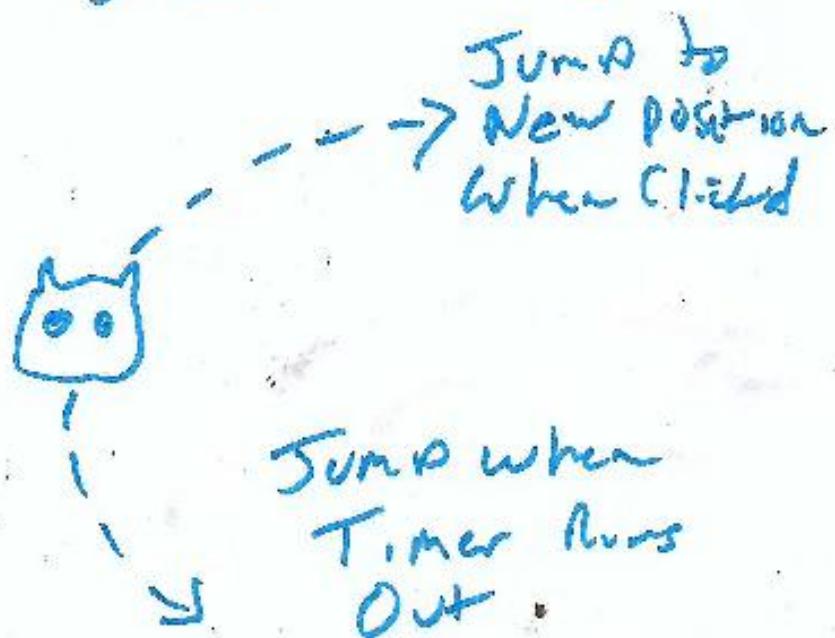
SKETCH A

Highscore Here



Sketch B

Score: 7 Highscore 12 ❤



Resources

The resources for this mini game are in the download you got with this book. See the acknowledgement page at the front of the book for the source of these assets.

Sprites

Sprites are images that will be used in your game. You'll use them to display the player, enemy, and other graphics in your game. Images needed for this

Sprites will generally be drawn by referencing or assigning them to objects. Next, load in the sprites for this game.

There are five of them, **spr_logo**, **spr_start**, **spr_target**, **spr_exit**, and **spr_lives**.

You can create a new sprite by clicking the right-clicking where shown in *Figure 7_1* :

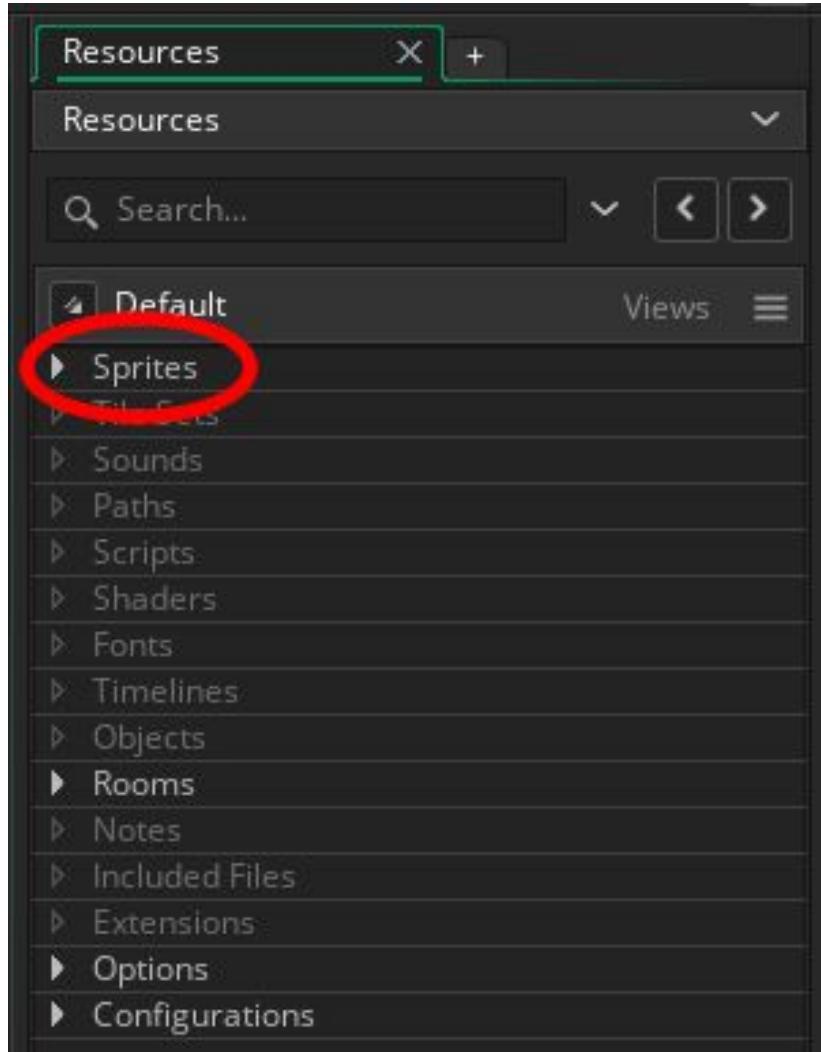


Figure 7_1. Showing where to click to create a sprite
Then select Create Sprite as shown in *Figure 7_2* :

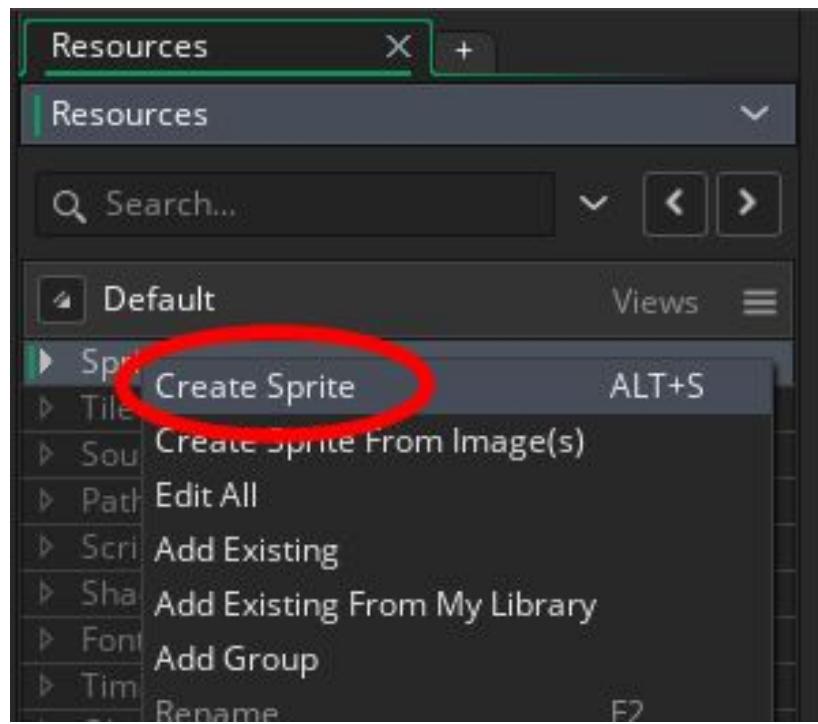


Figure 7_2: Creating a new sprite

Name the sprite spr_logo and click the import button, as shown in *Figure 7_3* :

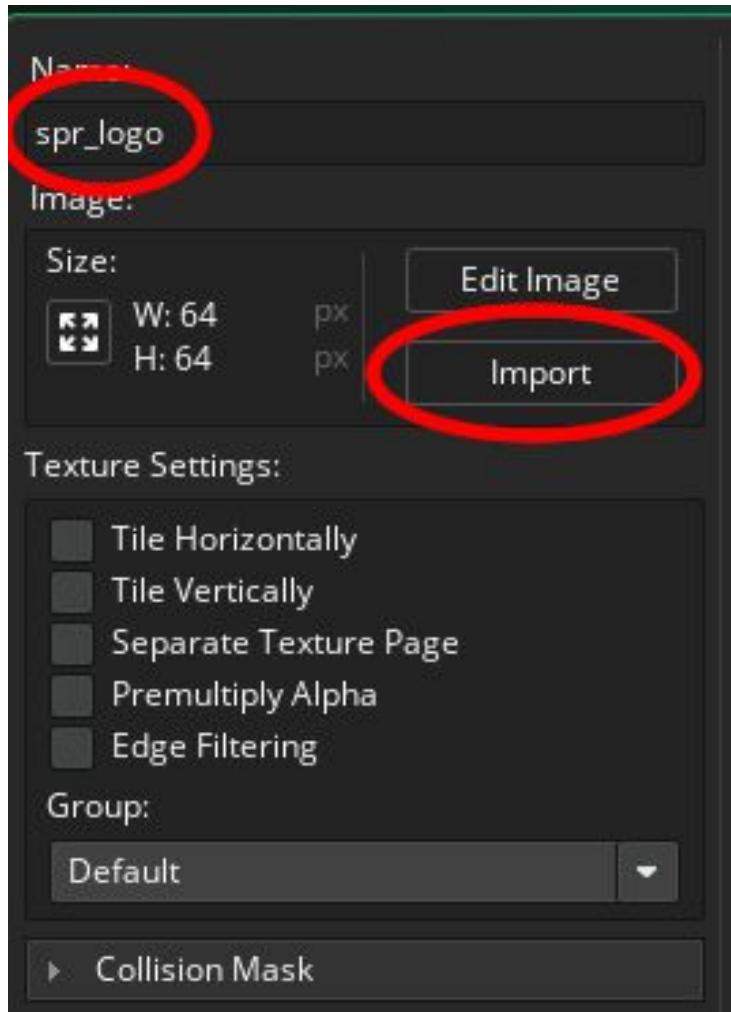


Figure 7_3: Naming sprite and clicking import

Select spr_logo, shown in Figure 7_4:

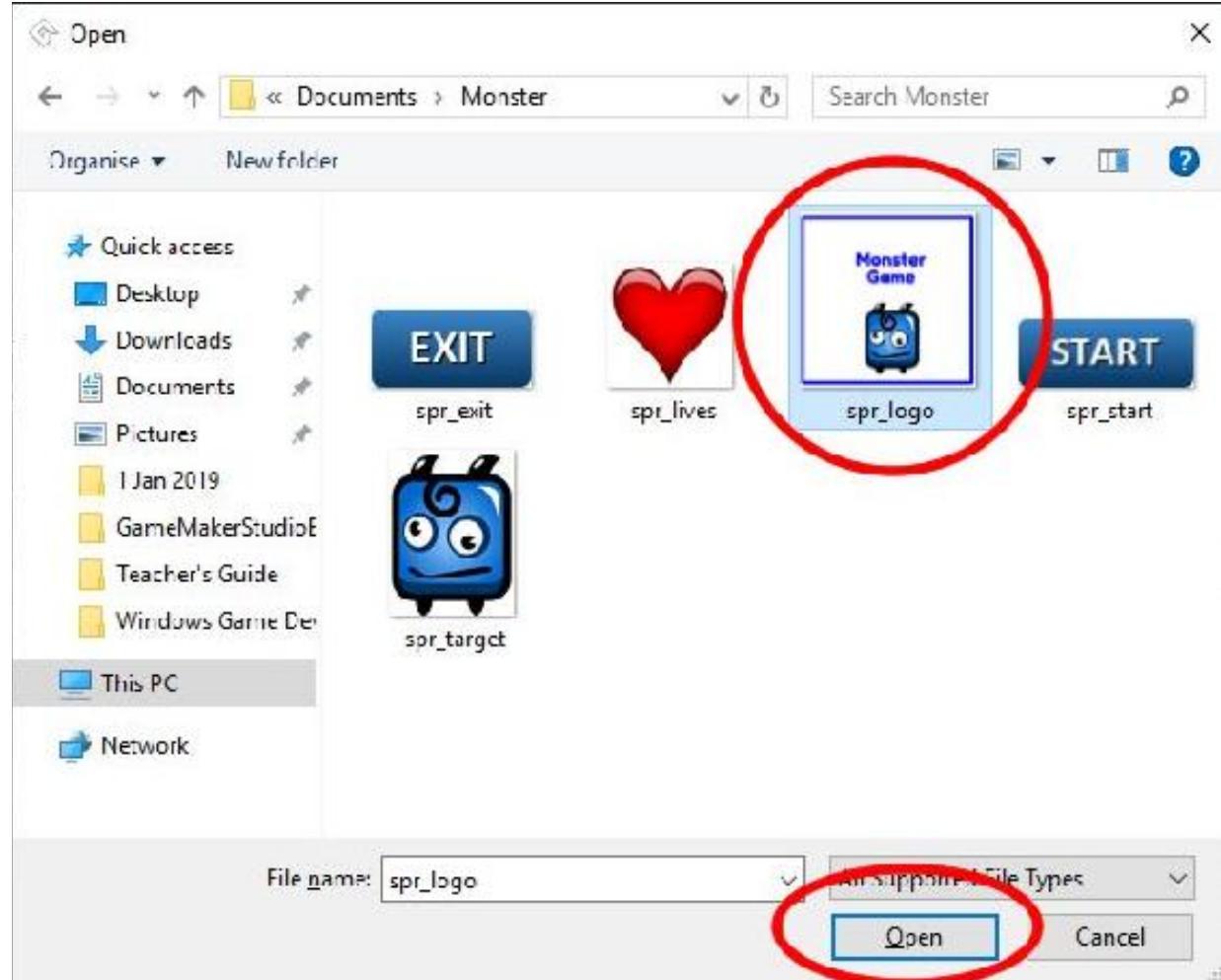


Figure 7_4: Selecting sprite image

Name the sprite **spr_logo** and click Load Sprite, select the file shown below, then Open, set the Origin to the center, the setting for this is shown in *Figure 7_5* :

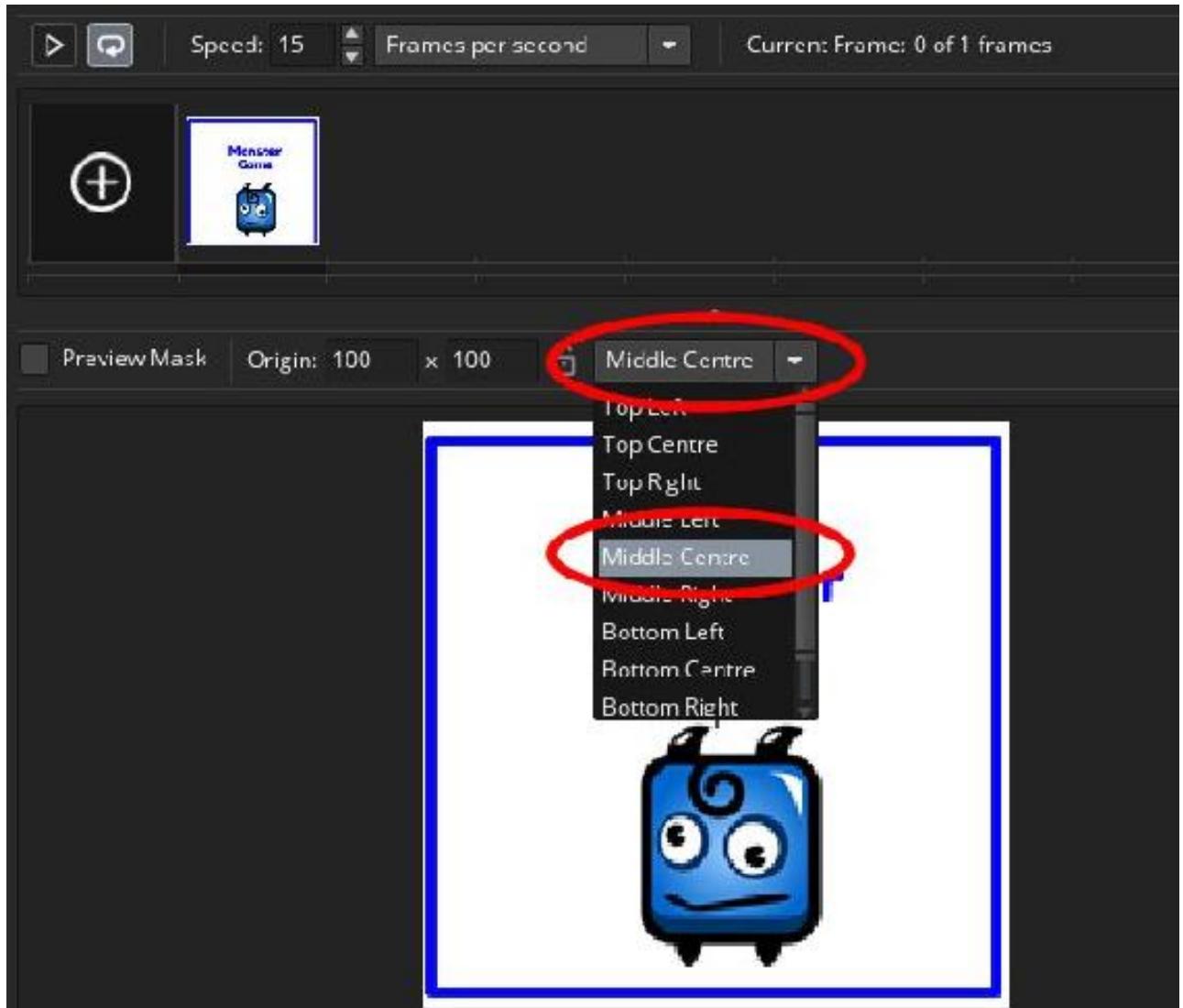


Figure 7_5: Setting the origin to middle center

Repeat this for the remaining sprites, naming them **spr_exit**, **spr_lives**, **spr_start**, and **spr_target**. Set the origin of all sprites to middle center. The origin is point in the sprite where it will be positioned in the room.

When done, the resources tree will look like that in *Figure 7_6*. You can change the order if you want to by clicking and dragging an asset.

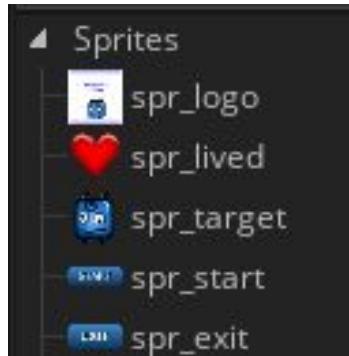


Figure 7_6: Showing sprite resources

This game only uses a few sprites, though some games can run into many hundreds. You can organize sprites (or other assets) by right-clicking in the resource's tree and creating a folder. You can then drag the assets to the created folder. You can sort by the room, or the type or any other way you want.

Rooms

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

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

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

You can do this by clicking the right clicking where shown in *Figure 7_7* and selecting Create Room:

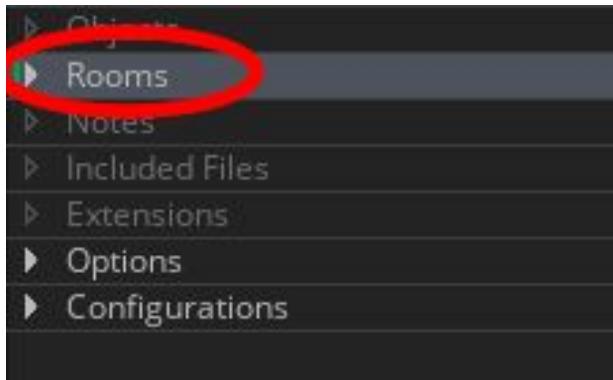


Figure 7_7: Creating a room

Right click on the room **room0** in the resource tree and select rename, rename as **room_menu**. Set the room dimensions as shown in *Figure 7_8*:

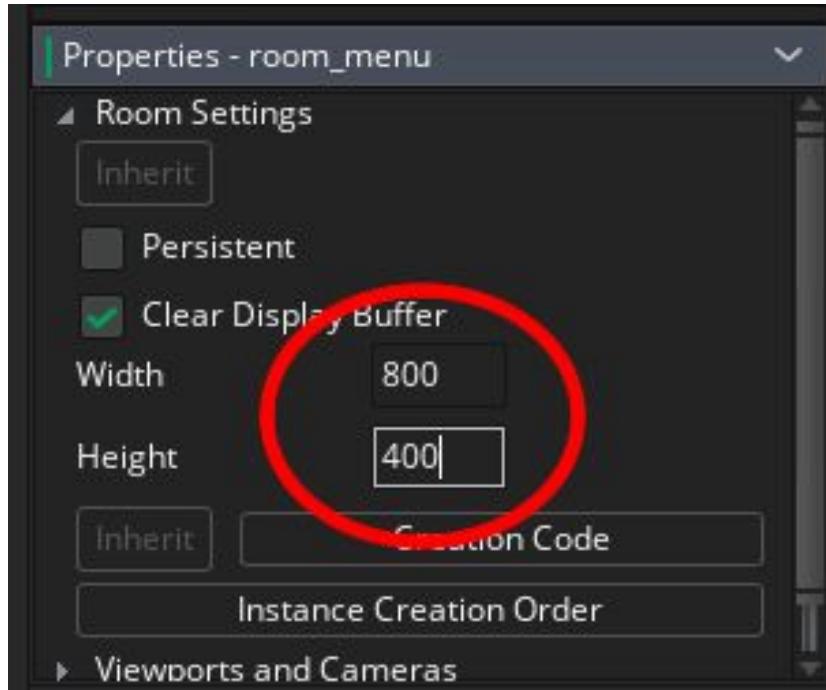


Figure 7_8: Setting room dimensions

Repeat this process for **room_game**, again setting the dimensions to 800 by 400.

The room at the top of the section of rooms in the resources tree will be the first room to run when the game starts. Ensure that **room_menu** is the first room to run.

Sounds

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

The example uses two sounds: **snd_click** and **snd_miss**.

You can do this by right clicking and selecting the **Create a sound** option, as shown in *Figure 7_9*:

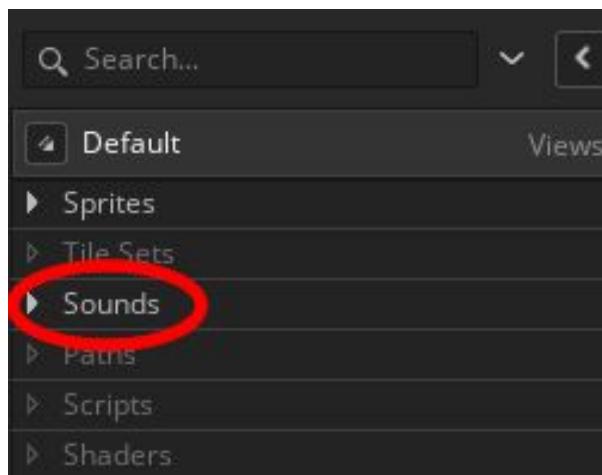


Figure 7_9. Create a new sound

Select the appropriate sound from the resources folder as shown in *Figure 7_10*, and name as **snd_click**:

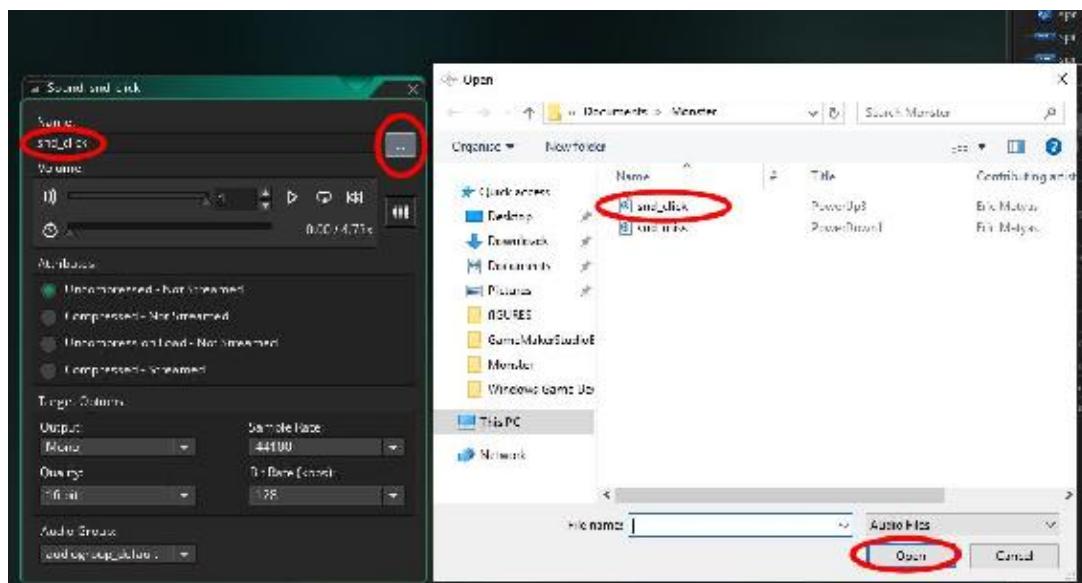


Figure 7_10: Loading in a sound resource

Repeat this with the sound file **snd_miss**.

Fonts

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

Figure i-11. Creating a font

Set the font name as **font_hud** and the size as 20 **Arial** as shown in *Figure 6_12* :

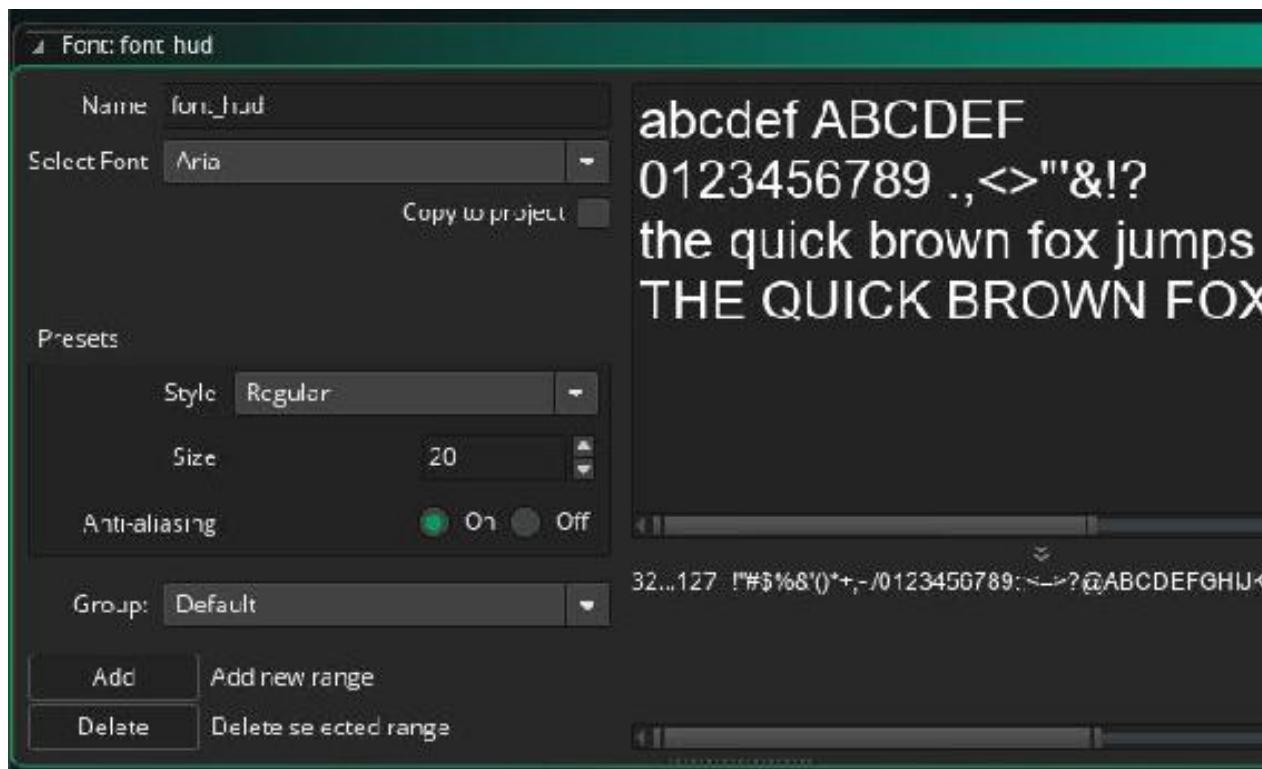


Figure 7_12: Naming and setting a font

Objects

Objects are the life blood of GameMaker Studio 2. Objects are used for displaying sprites, playing sounds, drawing text, detecting movement, processing functions, performing math calculations, and more.

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

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

This can be done by right clicking on Objects in the resource tree and selecting **Create Object**.

Name this object as **obj_logo**, as shown in *Figure 7_13* :

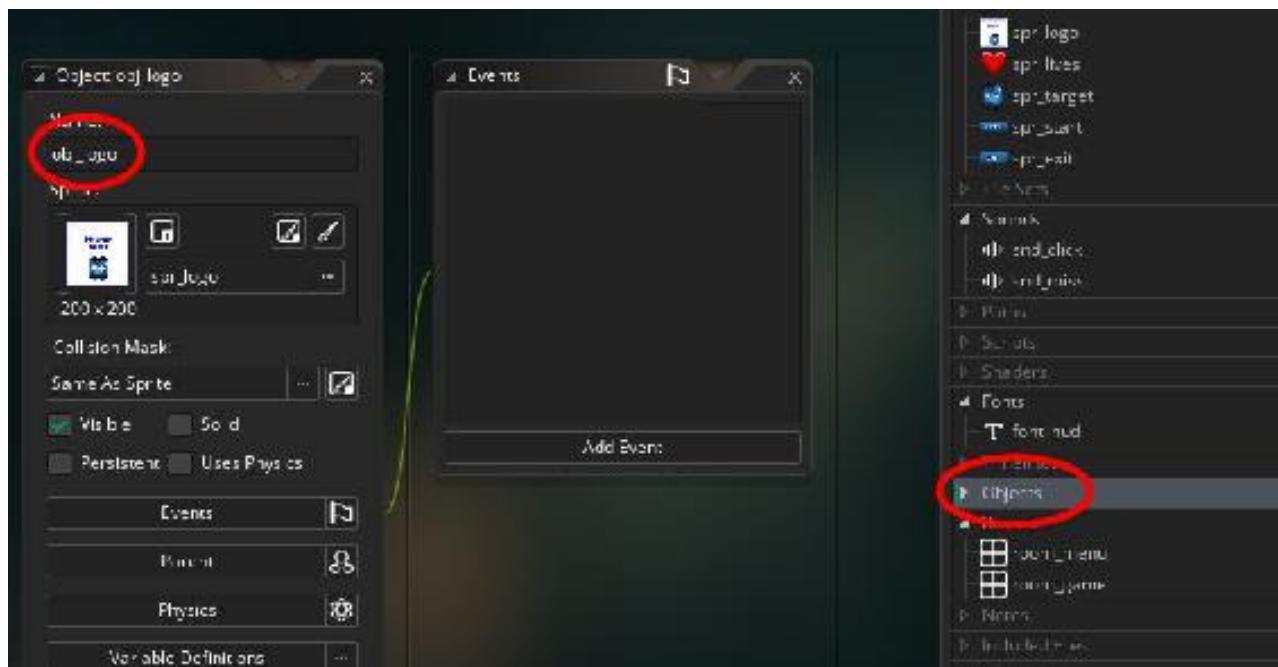


Figure 7_13: Creating a new object

Next is to assign a sprite to this object, assign the sprite **spr_logo** as shown in *Figure 7_14*:

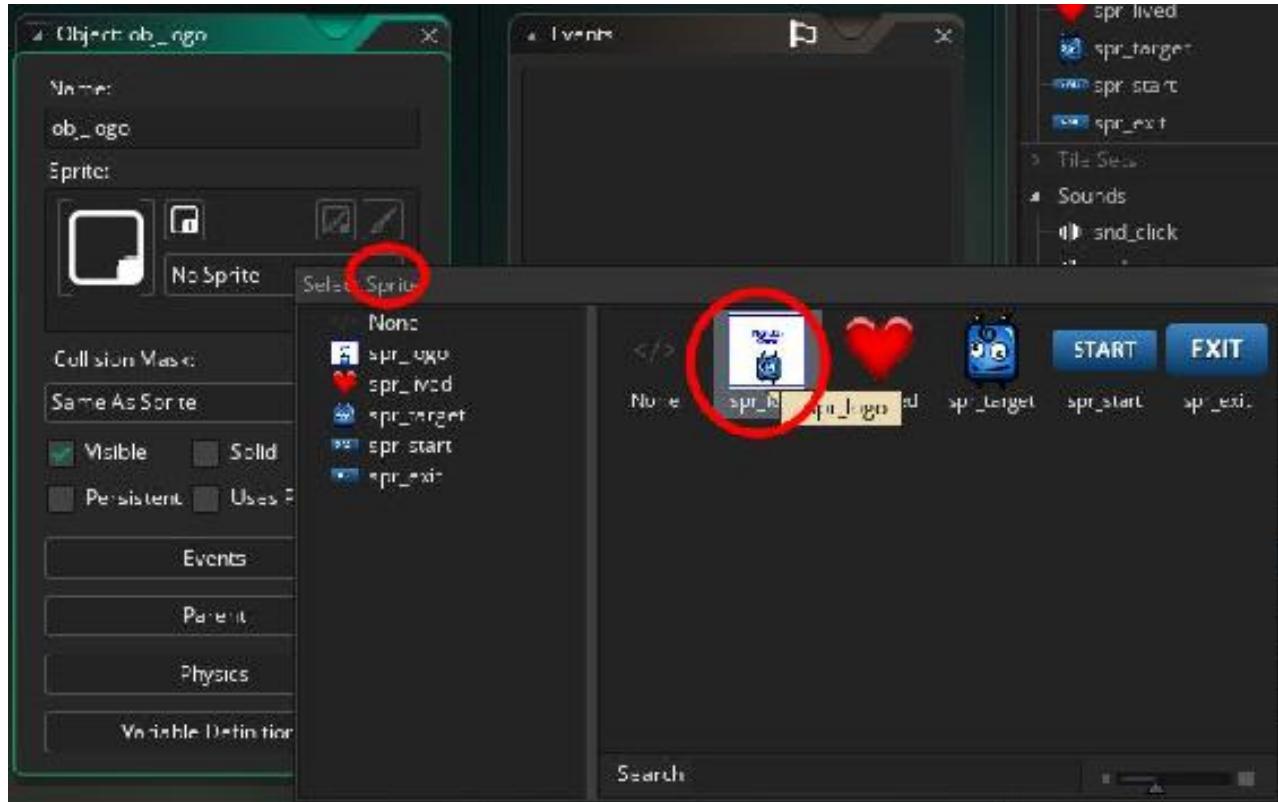


Figure 7_14: Assigning a sprite to an object

Then click ok.

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

The next step is to program some **Events**. Events are things that happen. The events you'll use most are the **Create Event**, **Step Event**, **Alarm Event**, and **Draw Event**. These can be set up using GameMaker Studio 2's built-in GUI.

Do this by clicking **Add Event** then **Create Event**, as shown in *Figure 7_15* :

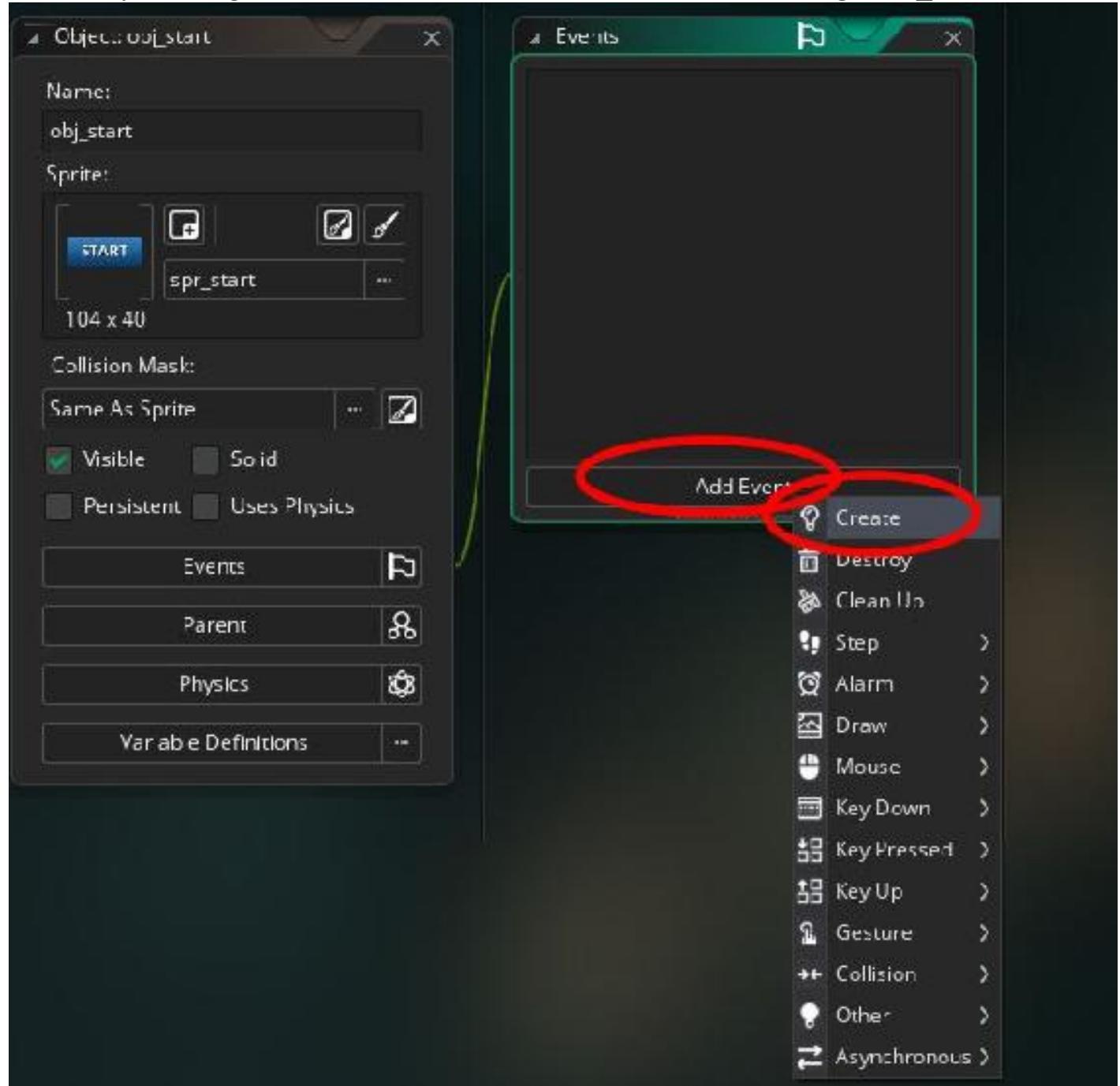


Figure 7_15: Making a create event

Drag the windows around using the Middle Mouse Button, so the window looks like that shown in *Figure 7_16*:

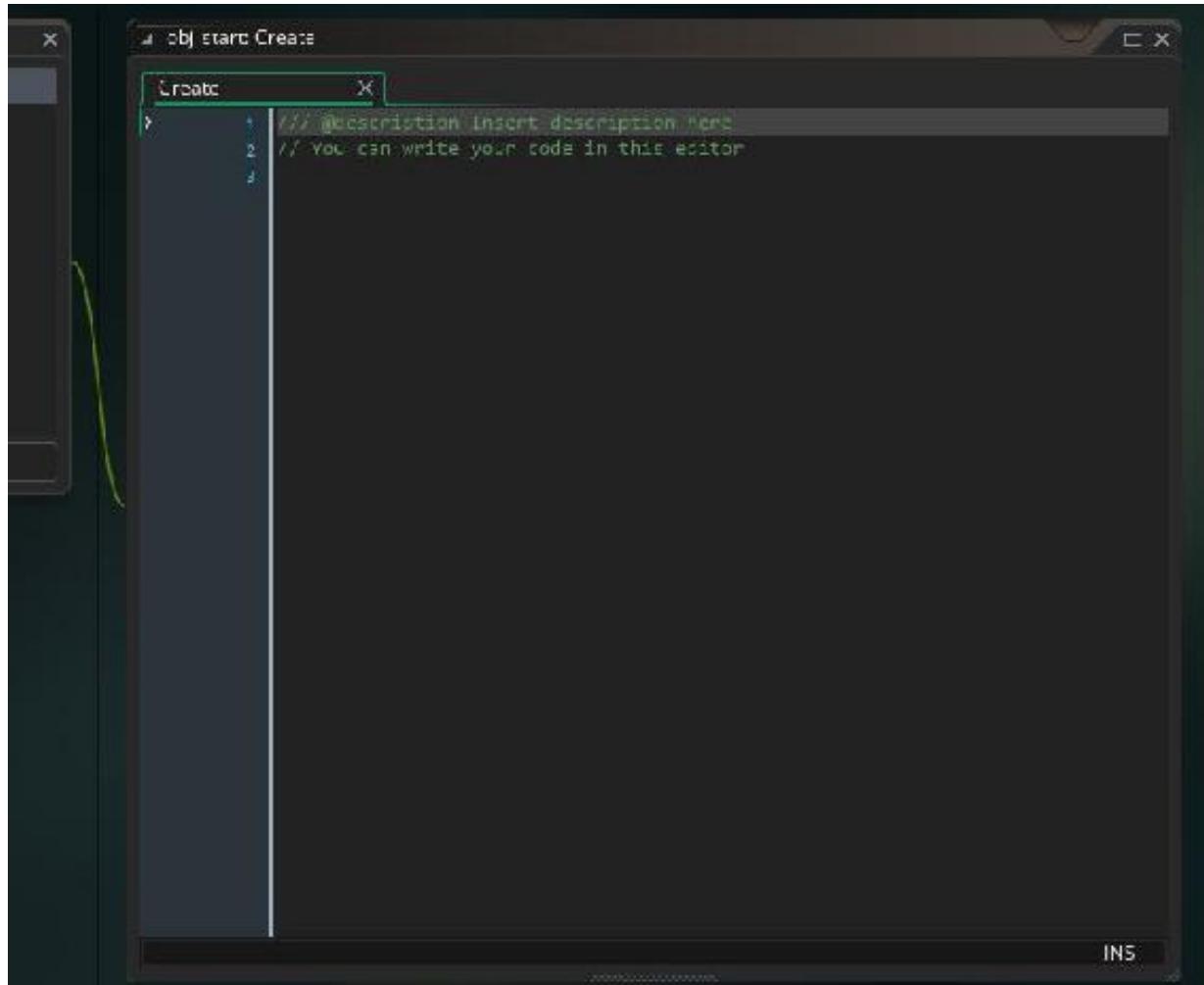


Figure 7_16: Showing window with create code in focus
In the open window, enter the following code:

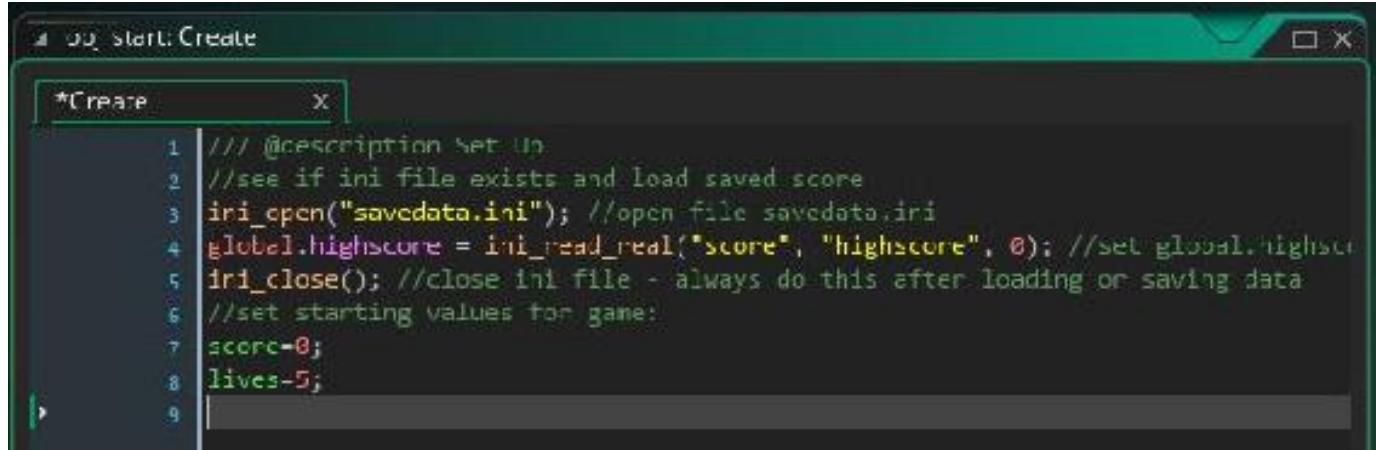
```
/// @description Set Up
//see if ini file exists and load saved score
ini_open("savedata.ini"); //open file savedata.ini
global.highscore = ini_read_real("score", "highscore",
0); //set global.highscore to value or set as 0 if no value
present
ini_close(); //close ini file - always do this after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

```
score=0;
```

```
lives=5;
```

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

When you've added the code, the open window will look as shown in *Figure 7_17*.



The screenshot shows a code editor window titled "start Create". The code in the editor is:

```
*Create x
1 // Description set up
2 // see if ini file exists and load saved score
3 iri_open("savedata.ini"); //open file savedata.ini
4 global.highscore = iri_read_real("score", "highscore", 0); //set global.highscore
5 iri_close(); //close ini file - always do this after loading or saving data
6 //set starting values for game:
7 score=0;
8 lives=5;
```

Figure 7_17. Showing code added

Next create a new event, a **Mouse Left Button Pressed Event** as shown in *Figure 7_18*:

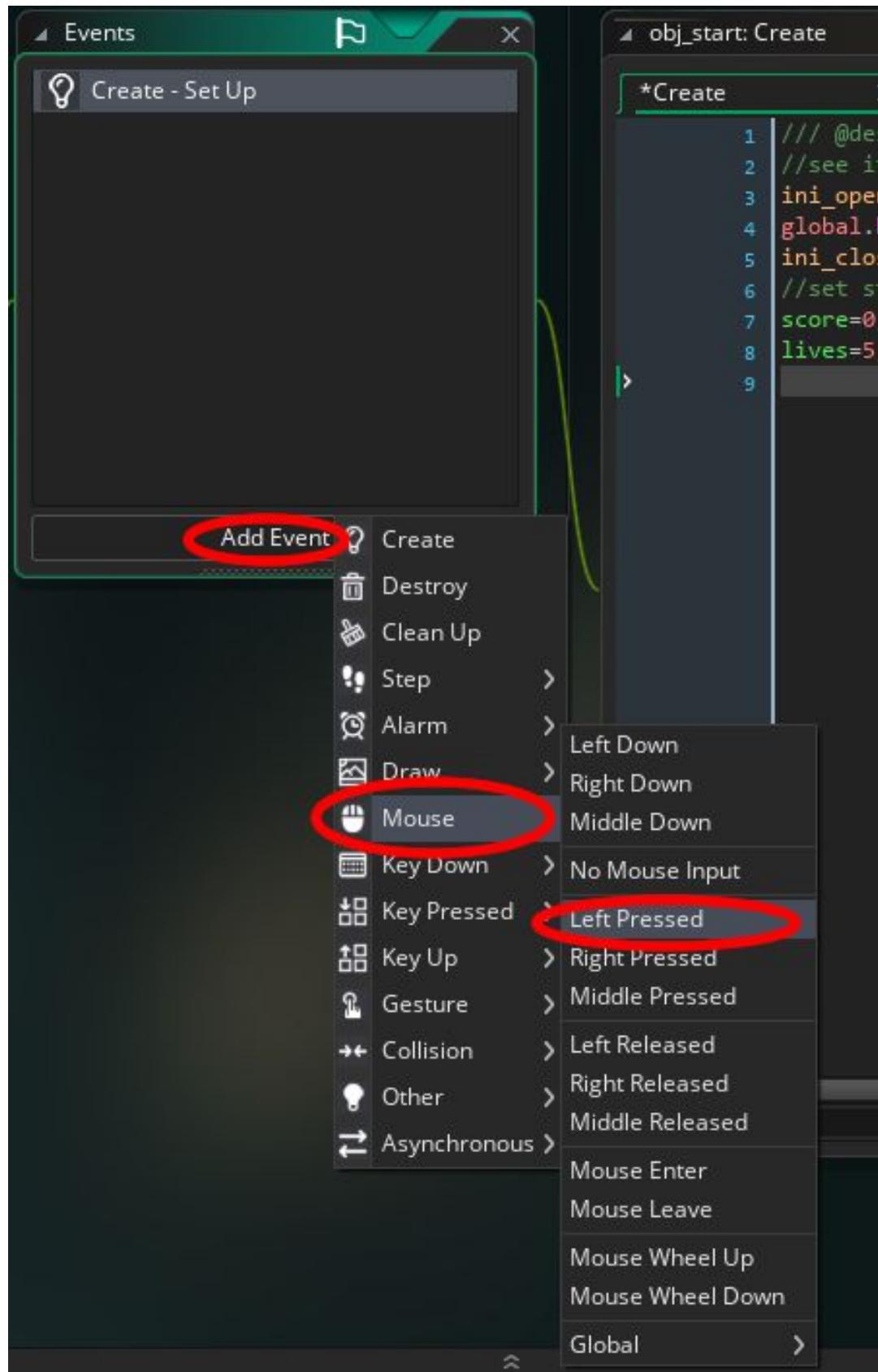


Figure 7_18: Creating a mouse left button released event

The code for this event is:

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

Next open up **obj_logo** and add a **Draw Event** by clicking **Add Event** followed by **Draw Event**, as shown in *Figure 7_19* :

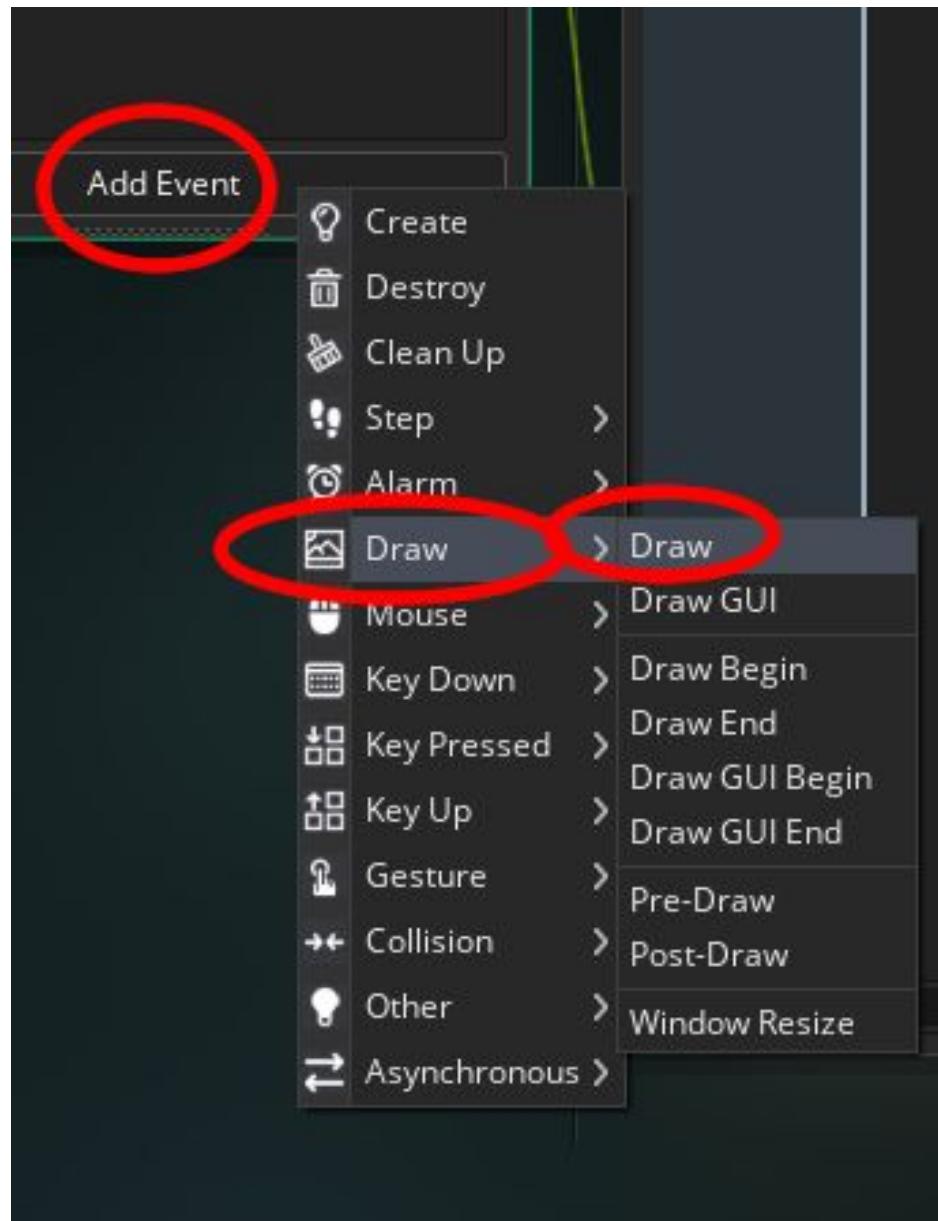


Figure 7_19: Selecting a draw event

The code for this event is:

```
/// @description Drawing Stuff  
draw_self(); //draws sprite assigned to this object  
draw_set_font(font_hud); //set font  
draw_set_halign(ha_center); //set horizontal alignment  
for drawn text  
draw_set_colour(c_white); //sets drawing colour as  
black  
draw_text(room_width/2, 280, "Highscore: "+  
string(global.highscore)); //draw Highscore:
```

A **Draw Event** is where you place your code, or D&D, to place text and images on the screen. Drawing functions, such as `draw_text` and `draw_self`, **must** be placed in **Draw Event**. If you have code in Draw Event, then you will need to force it draw the sprite assigned to the object, usually using the code:

draw_self();

If a **Draw Event** is not present, the object will automatically draw it's assigned sprite and subimage.

Click OK to save all changes.

Next create a new object **obj_exit** and assign the sprite **spr_exit**. Create a **Left Mouse Button Released Event** and add this code:

```
/// @description End the game  
game_end(); //closes game and returns to windows
```

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

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

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

already an instance of the object is already placed in it.

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

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

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

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

Next create an **Alarm Event0** as shown in *Figure 7_20*. This will activate if the player hasn't clicked the object in time. The GML will play a sound first, reduce the player's *lives* by 1, and create a new object, then destroy itself. There are other ways of making an instance move position and reset values, but I did it this way so I include some important code that you should know.

```
/// @description Play Sound, Create new target and
destroy current
//this code is executed when alarm finishes
audio_play_sound(snd_miss,1,false); //plays the sound
snd_you_are_dead
lives-=1; //reduces lives by 1
instance_create_layer(50,50,"Instances",obj_target);
//creates a new target
instance_destroy(); // removes current self from room
```

This will look like *Figure 7_20* when done:

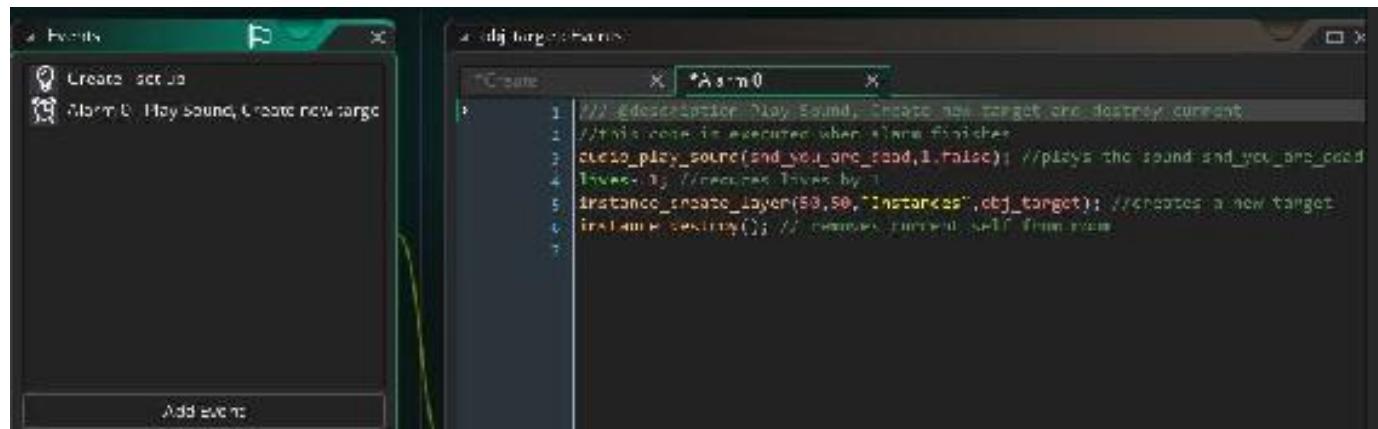


Figure 6-20: Showing code in an alarm 0 event

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

```
/// @description Draw self and timer bar  
draw_self();  
draw_set_colour(c_red); //sets drawing colour  
draw_rectangle(x-(alarm[0]/2), y-30, x+(alarm[0]/2), y-  
25,0); //draws a rectangle that reduces size based on  
alarm[0] value
```

The above code will draw the sprite for the object, set the drawing colour to red, and then draw a rectangle based on the current value of the **Alarm 0**—this will serve as a visual so the player knows how long they have to click the object. Save this object by clicking OK. You'll learn more about drawing geometric shapes in section 3.

Next is a **Left Button Pressed Event** with:

```
/// @description Player clicked on target  
score+=1; //add 1 to score  
audio_play_sound(snd_click,1,0) ; //play sound yeah  
instance_create_layer(50,50,"Instances",obj_target);  
//create new target  
instance_destroy(); //removes self from screen
```

Next create an object **obj_hud**. There is no sprite for this object. This object will be used as a control object that will be used to draw a HUD of the player's *lives* and *score*. It will also monitor how many lives the player has, and if the player has lost all of their lives it will update the high score if the player has a new high score and then restart the game. You do not need to create this file; it will be created automatically upon saving if it doesn't already exist (except if making an HTML5 export, when it must be included). Click **Add Event** and then **Step Event**. Add the following code to the **Step Event**:

```
if (lives<0) //checks if no more lives left, if 0 lives execute  
folling code  
/// @description Check Lives
```

```
{  
    if (score>global.highscore) //checks if score better  
    than saved score, if it is execute code  
    {  
        ini_open("savedata.ini"); //opens ini file  
        ini_write_real( "score", "highscore", score);  
        //writes and replaces current value  
        ini_close(); //closes ini file  
    }  
    game_restart();//restarts game  
}
```

This code will update the saved value in the INI file if the current *score* is bigger than global.highscore.

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

```
/// @description Drawing Stuff
draw_set_font(font_hud); //sets the font
draw_set_halign(ha_left); //sets alignment
draw_set_colour(c_white); //sets drawing colour
draw_text(25, 25, "Score: "+string(score)); //draws
score: + score
if score>global.highscore //executes following if score is
bigger
{
    draw_text(300, 25, "Highscore: "+string(score));
}
else //other wise just draw previous highscore
{
    draw_text(300, 25, "Highscore: "+ string
(global.highscore));
}
draw_sprite(spr_lives,0,600,25);
draw_text(590, 15,lives);
```

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

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

Use the objects resources tab to select objects and drag them into room, as shown in *Figure 7_21* :

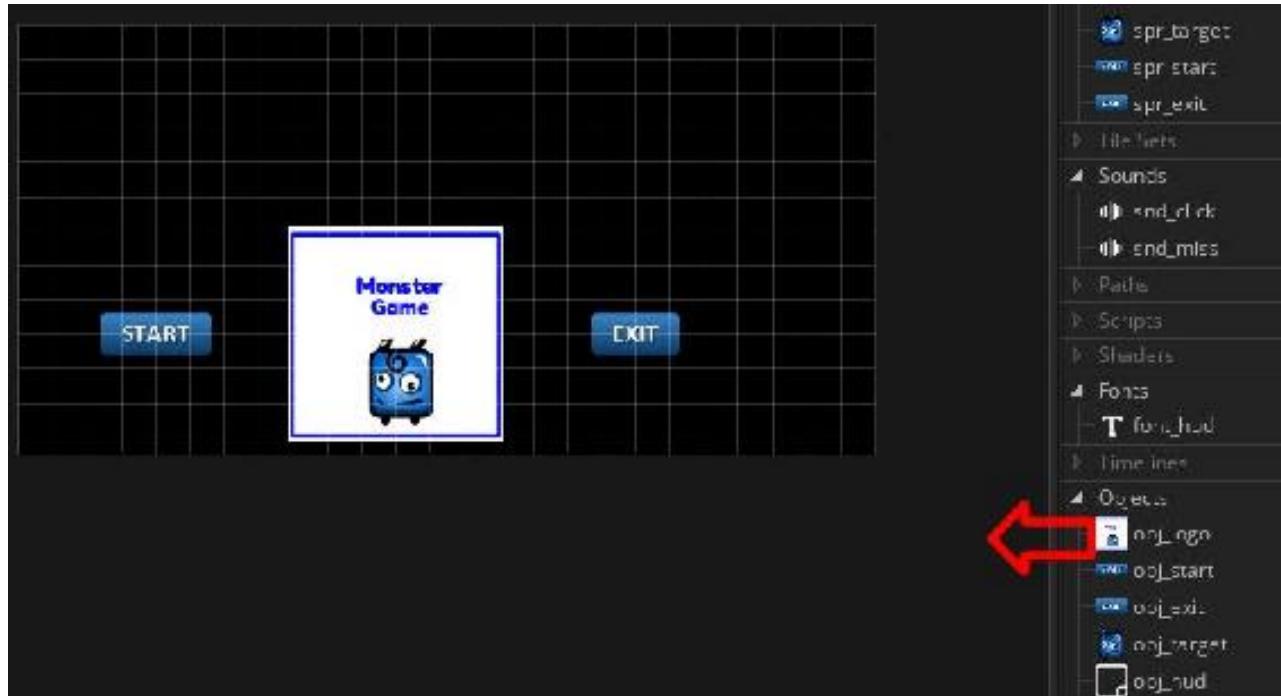


Figure 7_21: Placing instances in room

Next open **room_game**.

Place one instance of **obj_target** and one of **obj_hud** in the room. It doesn't matter where you place them.

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

Click the green triangle, shown in *Figure 7_22*, at the top left to play your game.

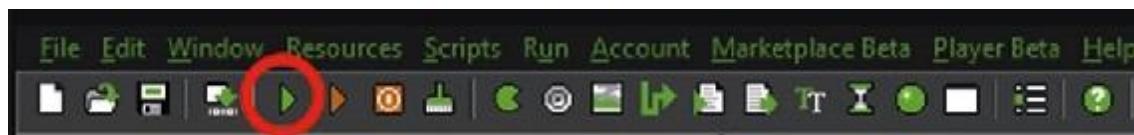


Figure 7_22. Testing the game

Comments

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

In GameMaker Studio 2 there are a number of types of comments you can use. The main types are a code block description:

```
/// @description What This Does..
```

Which shows on the preview plane.

Another example in code would look like this:

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

Which can placed on a blank line, or after code.

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

You start this section with /* and end with */ . For example:

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

```
ini-close();
```

```
*/
```

There are few more types of comments that can be used when using scripts, providing auto-complete text and info on the script arguments.

The appendix at the book of the book has an introduction to commonly used GML functions and coding. This is taken from my sister book **Beginning GameMaker Studio**. The next page shows where you can get the full book with a 50% discount.

Game Programming

The remainder of this chapter covers and explains the coding used for the sample game. This shows all the updates made after the beta testing a debugging. The code for each object is shown, though if you were making this game from scratch, you would approach it differently. When I make game, whether small or large, I try to do it in such way that I can add elements in an order that allows me test after each part. The order used for this game was:

- Load in player sprites
- Basic player movement
- Load in backgrounds and foreground
- Edit player code to set up parallax system
- Create player projectiles
- Update player to shoot projectiles
- Add first enemy and set up sprites and movement
- Add second enemy and set up sprites and movement
- Add boss enemy and set up sprites and movement
- Create enemy bullets
- Set up enemy collisions with player projectiles
- Set up player collision with enemy and enemy bullets
- Create HUD, score system and health
- Set up enemy health system
- Set up scripts
- Add music and audio
- Set up game menu
- Set up credits

- Set up how to play
- Tweak and test, making
- Send out for beta testing
- Edit game to take on board beta testing
- Final testing

I will show the code for the objects and events, and then break down and explain what the code does. As this book is intended for users of GameMaker Studio who are quite new, the code is written in such a way that it should be relatively easy to understand the code, due to the code and the comments. I make no claim that my coding is the best or most concise, mainly that it is easy to understand the code and how it works. The code shown is the edited code after beta testing changes have been made. Please note that this game is intended to be a game that will win any prizes for its quality and game play, rather than a little game that shows the process of programming a game.

I'll be using the **Rubber Duck Debugging** method.

If you haven't heard of this technique, here is part of the entry from Wikipedia (CC-SA 3.0), the free encyclopedia:

*In software engineering, rubber duck debugging is a method of debugging code. The name is a reference to a story in the book *The Pragmatic Programmer* in which a programmer would carry around a rubber duck and debug their code by forcing themselves to explain it, line-by-line, to the duck. Many other terms exist for this technique, often involving different inanimate objects.*

As such I may explain code you already understand, or state the obvious, feel free to skip any content that you already know.

Although the purpose of the objects was discussed in chapter 2, I will again provide a brief explanation of the object and its use.

Scripts

I use a few scripts in this game, they are:

scr_angle_rotate

```
/// @function scr_angle_rotate(angle1,angle2,speed)
//slowly rotates from current to target angle
return argument0 + clamp(angle_difference(argument1,
argument0), -argument2, argument2)
```

The above script will take in two angles, and return a new angle between the two, based on the speed given. This script is used by the player's special weapon that locks-on to a target. Using the script makes the weapon slowly rotate towards the target instance. Although only one object uses this code, a script is used in this case because I have used this code in many games and projects, using a script allows me to quickly copy and paste between projects.

scr_health_bar

```
xx=argument0;//xposition
yy=argument1;//yposition
wd=argument2;//width
draw_healthbar(x-wd+xx,y-10+yy,x+wd+xx,y+10+yy,
(100/start_hp)*hp,c_black,c_red,c_green,0,true,true);
```

This script takes in 3 arguments, and uses these values to draw a health bar at the given location and the given width. Lots of objects in the game make use of this script. Using a script allows me to quickly make a change that effects all uses of it, instead of having to search through lots of code to make changes.

Objects

This Section Is For The Splash Screen Room

obj_splash

This object sets up variables for the and loads in any highscore.

There is no sprite for this obect.

Create Event

```
/// @description Set Up
global.level=1;
global.shoot_speed=1;
global.pow=1;
health=100;
score=0;
ini_open("scores.ini");//open file
global.highscore=ini_read_real("scores","high",0);//load scores high if present, otherwise set as 0
ini_close();//close ini
room_goto(room_menu);
```

This object is placed in the room **room_splash**, which is the first room to load when the game is started. It sets up the initial values and loads a highscoe if present.

The first 5 lines of code set some initial global values. Global values are used so that these values stay in memory and are accessible by other objects within the game.

ini_open("scores.ini");//open file

This line opens an ini file **scores.ini** (if it exists).

global.highscore=ini_read_real("scores","high",0);//load scores high if present, otherwise set as 0

This will set the value of `global.highscore` to the saved value, or set as 0 if no value exists.

ini_close();//close ini

This will close the open ini file. It is very important to close the ini file as soon as you down with it.

room_goto(room_menu);

This then exit this room and goto the room **room_menu**.

This is usual practice for most games, to set initial values and load any needed data.

This Section Is For The Game's Menu Room

obj_player_menu

This object will be used in the menu room, to make it a bit more interesting. It will move up and down.

This object uses the sprite **spr_player**. The origin of this sprite is 60x65, and a size of 96x91 as shown in *Figure 7_23* :

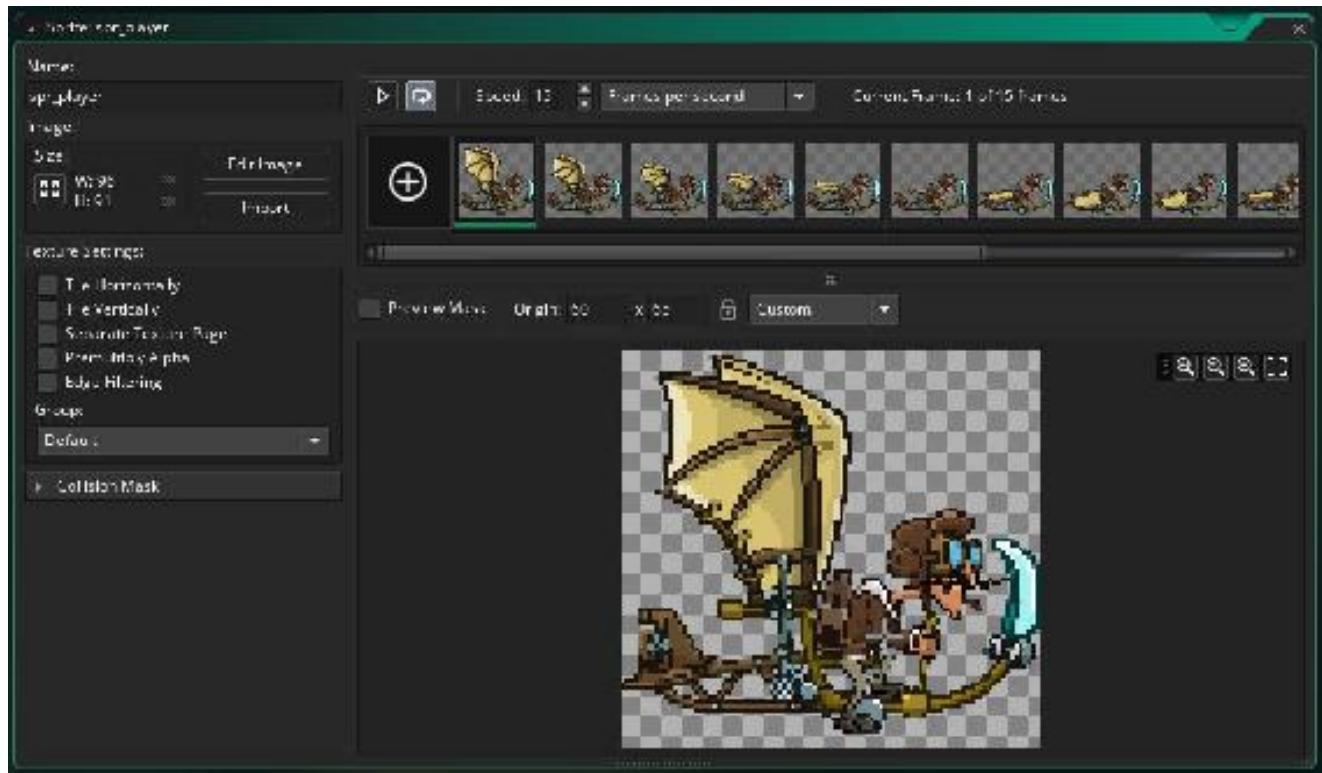


Figure 7_23: Showing sprite set up for spr_player

Create Event

```
/// @description Set up
flying_level=room_height/2;
y=flying_level;
xx=x;
img=10;
vspeed=3;
```

The event will set the initial values needed for this object. This object will move up and down, making the menu screen look a little better.

vspeed=3;

This code will set the vertical speed of 3. A positive value will make the instance move downwards.

Step Event

```
/// @description Movement & Parallax Control  
diff=(flying_level-y)/3;  
layer_y("bg_1",550+diff);  
layer_y("bg_2", (450+diff)/2);  
layer_y("bg_3", (-100+diff)/3);  
ang=img-(diff/4);  
ang=clamp(ang,1,20);  
image_angle=ang;
```

```
diff=(flying_level-y)/3;  
layer_y("bg_1",550+diff);  
layer_y("bg_2", (450+diff)/2);  
layer_y("bg_3", (-100+diff)/3);
```

These lines of code make the vertical positions of the background move up and down in relation of the player's y position. This helps create a cool looking parallax effect.

```
ang=img-(diff/4);  
ang=clamp(ang,1,20);  
image_angle=ang;
```

These 3 lines of code change player sprites angle based on it's y position. This makes the sprite point up and down as it moves.

Intersect Boundary Event

```
/// @description Change Direction  
vspeed*=-1;
```

```
vspeed*=-1;
```

When the instance collides with the room boundary (the border) it changes the vertical speed from positive to negative, or, negative to positive. This makes the instance change from moving up to down, and vice versa.



obj_menu

This will display various options (the sprites subimages), which the player can scroll through.

This object has **spr_menu** assigned, as shown in *Figure 7_24* :

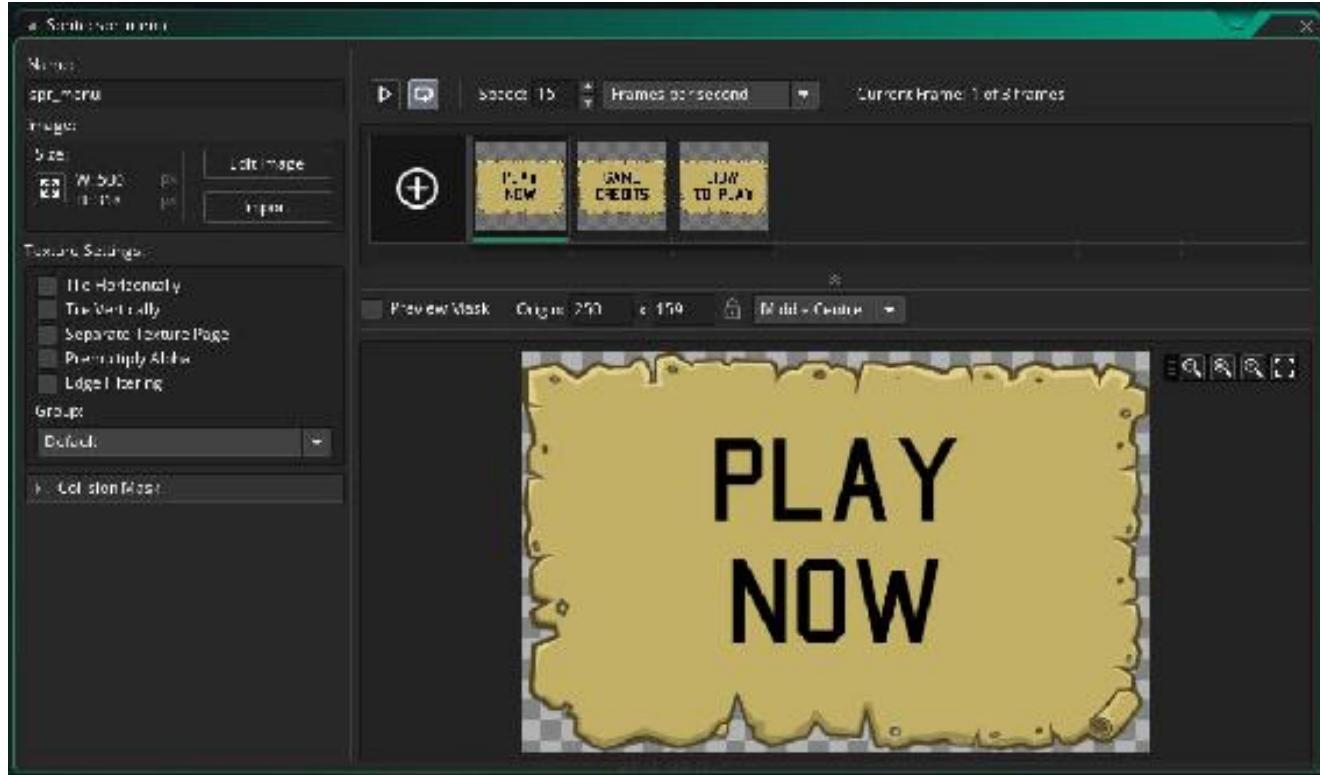


Figure 7_24: Showing sprite set up for spr_menu

Create Event

```
/// @description Set Up
global.selected=0;
rot=0;
global.item=0;
menu_x=room_width/2;
menu_y=room_height/2;
menu_width=500;
menu_height=50;
audio_stop_all();
```

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

```
global.selected=0;
```

```
rot=0;
```

```
global.item=0;
```

These 3 lines set initial variables needed.

```
menu_x=room_width/2;  
menu_y=room_height/2;  
menu_width=500;  
menu_height=50;
```

This set values needed for positioning the menu and how big it should be.

```
audio_stop_all();  
audio_play_sound(snd_music_menu,1,true);
```

These 2 lines stop and audio (music) and start playing the music for the menu room on loop.

Step Event

```
/// @description Step Stuff  
//Key Press  
global.selected+=keyboard_check_pressed(vk_left)-  
keyboard_check_pressed(vk_right);  
if keyboard_check_pressed(vk_left) or  
keyboard_check_pressed(vk_right)  
{  
    audio_play_sound(snd_menu_select,1,false);  
}  
//Set Angle  
rot=angle_difference(rot,global.selected*  
(360/image_number))/ (0.2*room_speed);  
  
//Set Selection  
global.item=-global.selected mod image_number;  
if (global.item)<0 global.item+=image_number;
```

```
global.selected+=keyboard_check_pressed(vk_left)-
keyboard_check_pressed(vk_right);
```

These 2 lines detect a left or right arrow key press and change the value of global.selected accordingly.

```
if keyboard_check_pressed(vk_left) or
keyboard_check_pressed(vk_right)
{
    audio_play_sound(snd_menu_select,1,false);
}
```

This play a sound effect when left or right is pressed.

```
//Set Angle
rot=angle_difference(rot,global.selected*
(360/image_number))/ (0.2*room_speed);
```

This the value of rot, which is used when in the draw event.

```
//Set Selection  
global.item=-global.selected mod image_number;  
if (global.item)<0 global.item+=image_number;
```

Sets which is the currently chosen option. Use when player makes a selection.

Draw Event

```
/// @description Draw Panels & Info  
//Set Variables  
var pr, i;  
i=0;  
//Create a List  
pr=ds_priority_create();  
//Add  
repeat (image_number) {  
    ds_priority_add(pr,i,lengthdir_y(1,(rot-90)+i*(  
        360/image_number)));  
    i+=1;}  
//Draw  
repeat (image_number) {  
    i=ds_priority_delete_min(pr);  
    draw_sprite_ext(sprite_index,i,menu_x+lengthdir_x(me  
        nu_width/2,(rot-90)+i*(  
            360/image_number)),menu_y+lengthdir_y(menu_heigh  
                t/2,(rot-90)+i*(  
                    360/image_number)),1+lengthdir_y(menu_height/2,  
                        (rot-90)+i*)
```

```
(360/image_number))/(menu_height*2),1+lengthdir_y(menu_height/2,(rot-90)+i*(360/image_number))/(menu_height*2),0,c_white,1);  
}  
  
draw_set_color(c_white);  
//Free Memory  
ds_priority_destroy(pr);  
  
//draw high  
draw_set_font(font_menu);  
draw_set_colour(c_black);  
draw_set_halign(fa_center);  
draw_text(300,20,"Highscore  
"+string(global.highscore));
```

```
var pr, i;  
i=0;  
//Create a List  
pr=ds_priority_create();  
//Add  
repeat (image_number) {  
    ds_priority_add(pr,i,lengthdir_y(1,(rot-90)+i*(360/image_number)));  
    i+=1;}  
//Draw  
repeat (image_number) {  
    i=ds_priority_delete_min(pr);
```

```
draw_sprite_ext(sprite_index,i,menu_x+lengthdir_x(menu_width/2,(rot-90)+i*(360/image_number)),menu_y+lengthdir_y(menu_height/2,(rot-90)+i*(360/image_number)),1+lengthdir_y(menu_height/2,(rot-90)+i*(360/image_number))/(menu_height*2),1+lengthdir_y(menu_height/2,(rot-90)+i*(360/image_number))/(menu_height*2),0,c_white,1);}
```

This code will make a list and add values which are then used to determine where and what size to draw each option.

ds_priority_destroy(pr);

This destroys the list as it is no longer. Destroying lists when done with is very important. Leaving them will eat memory and potentially crash your game.

```
draw_set_font(font_menu);
draw_set_colour(c_black);
draw_set_halign(fa_center);
draw_text(300,20,"Highscore
"+string(global.highscore));
```

These lines set the drawing style and draws the currently held value of the highscore on screen.

Mouse Wheel Up Event

```
/// @description Raise Selection
global.selected+=1;
audio_play_sound(snd_menu_select,1,false);
```

```
global.selected+=1;
```

audio_play_sound(snd_menu_select,1,false);

Changes the selected option and plays a sound.

Mouse Wheel Down Event

/// @description Lower Selection

global.selected=1;

audio_play_sound(snd_menu_select,1,false);

global.selected=1;

audio_play_sound(snd_menu_select,1,false);

Changes the selected option and plays a sound.

obj_button

This is a button the player can click, which will then take the player to selected room.

This has **spr_button** assigned, as shown in *Figure 7_25* :



Figure 7_25: Showing spr_button set up

Draw Event

```
/// @description Draw Button & Text
draw_self();
x=room_width/2;
draw_set_font(font_menu);
draw_set_colour(c_black);
draw_set_halign(fa_center);
draw_text(x,100,"Scroll With Arrow Keys or Mouse
Wheel");
draw_text(x,150,"Click Select To Play");
```

This sets the drawing style and position and draws the given text.

Left Pressed Event

```
/// @description Act On Selection
audio_stop_all();
audio_play_sound(snd_menu_button,1,false);
switch (global.item) {
    case 0:
        room_goto(room_game);
        break
    case 1:
        room_goto(room_intro);
        break
    case 2:
        room_goto(room_how_play);
        break
}
```

This code will take the current value of global.item and use a switch to act accordingly. If you know ahead what a value is before testing it, in a lot of cases a switch is better than using if.

This Section Is For The Game's Credits Room

obj_sparkle

This object will be used to star field effect in credits and how to play rooms. It is made to move out from a central point in the room.

This object uses **spr_sparkle**. As shown in *Figure 7_26* below:

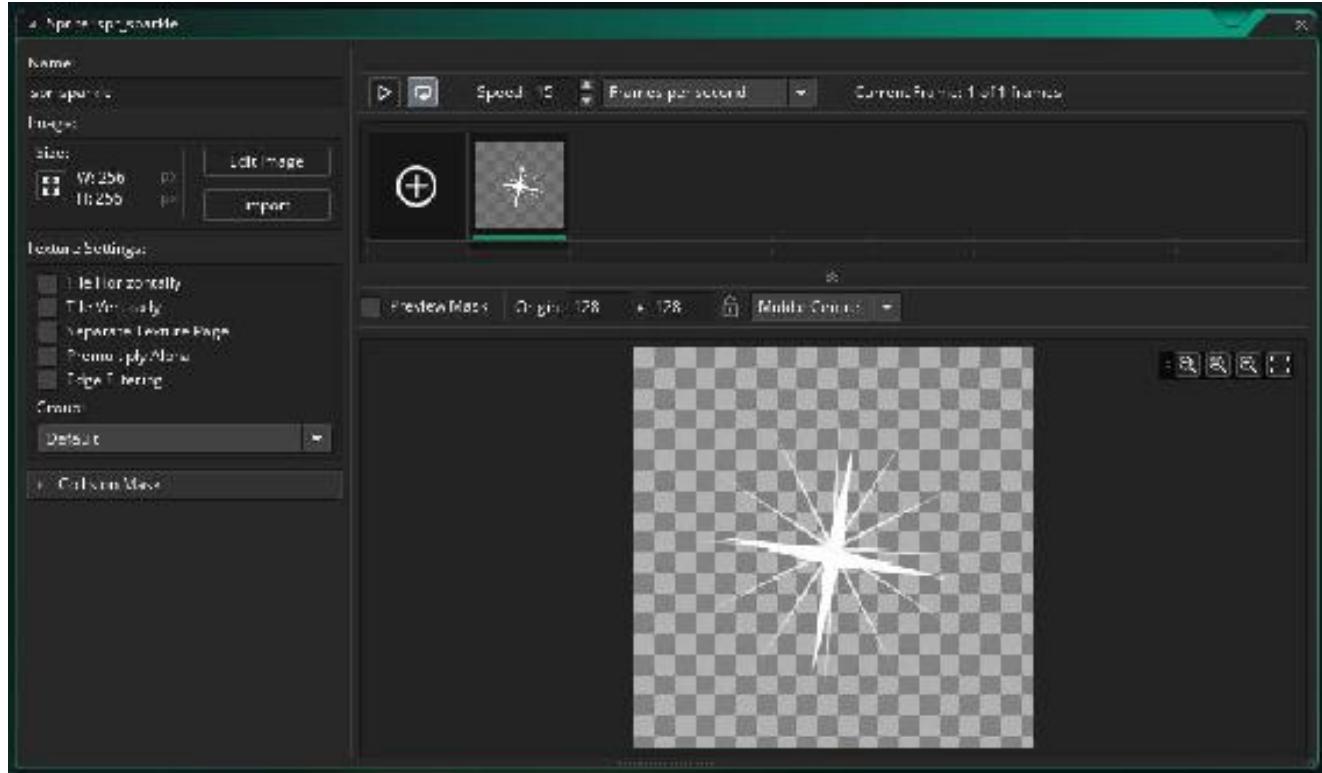


Figure 7_26: Showing set up for spr_sparkle

Create Event

```
/// @description Set Up  
image_xscale=0.02;  
image_yscale=0.02;
```

```
image_xscale=0.02;  
image_yscale=0.02;
```

This sets the initial scale size of the sprite.

Step Event

```
/// @description Rotate and speed up
image_xscale+=0.002;
image_yscale+=0.002;
image_angle+=0.5;
speed*=1.002;
```

This gradually increases the size, slowly rotates it and increases the speed (in the direction it is already travelling in).

Outside Room Event

```
/// @description Destroy
instance_destroy();
```

This event will check if the instance is outside the room borders, if it is it will be destroyed. Deleting instances when no longer needed, for example when outside the room – it is important to destroy them, this helps prevent memory leaks.

obj_intro_control

This sets up the text needed to be shown. It adds this a line at a time to a list, then pulls off each line and processes it.

There is no sprite for this object.

Create Event

```
/// @description Set Up Text
credits=ds_list_create();
ds_list_add(credits,
"Left Mouse Button To Exit",
"",
"",
"Graphics By",
"",
"",
"GameDeveoperStudio.com",
"",
"",
"",
"Music & Audio Effects",
"",
"",
"SoundImage.org",
"",
"",
"",
"Programmed By Ben",
"",
"",
"Many Thanks To The Following",
"Who Funded This Project",
"",
"",
"Michał Kamiński",
"Corey Cuhay",
"Honey",
```

```
"Pedro Santos",
"Mark Porter",
"Dean Radcliffe",
"Mickey Everett",
"Vasco",
"Mike Cowell",
"Gaven Renwick",
"");
show_debug_message(string(ds_list_size(credits))+" <<
size");
alarm[0]=room_speed;

audio_stop_sound(snd_music_menu);
audio_play_sound(snd_music_credits,1,true);
```

The first part of the code creates a list and adds all the text lines that will be displayed.

```
alarm[0]=room_speed;
```

This sets an alarm at the room_speed (this is equal to 1 second).

```
audio_stop_sound(snd_music_menu);
audio_play_sound(snd_music_credits,1,true);
```

This stops any music playing and plays the sound for the credits on loop.

Alarm 0 Event

```
/// @description Get Any Text
if ds_list_size(credits)>0
{
    text=credits[0];
    ds_list_delete(credits,0);
```

```
    inst=instance_create_layer(x,room_height+100,"Instances", obj_intro_text);
    inst.text=text;
    alarm[0]=room_speed*1.5;
}
else
{
    alarm[1]=room_speed*8;
}
```

```
if ds_list_size(credits)>0
{
    text=credits[0];
    ds_list_delete(credits,0);
```

This checks if the list still has content, if it has it sets the value of text to the next value and then deletes from the list.

```
inst=instance_create_layer(x,room_height+100,"Instances", obj_intro_text);
inst.text=text;
alarm[0]=room_speed*1.5;
```

This makes an instance of obj_intro_text and sends it the value of text. It then sets the alarm to 1 and a half seconds.

```
else
{
    alarm[1]=room_speed*8;
}
```

This sets the alarm 1 to 8 seconds if the list is empty. This 8 seconds gives time for any text on the screen to roll up.

Alarm 1 Event

```
/// @description Do Something When All Text Done  
room_goto(room_menu);
```

Goes back to the menu room when triggered.

Global Left Pressed Mouse Event

```
/// @description Go To Menu Room  
audio_stop_all();  
audio_play_sound(snd_menu_button,1,false);  
room_goto(room_menu);
```

If the player clicks the left mouse button, it will go back to the menu room.

obj_intro_text

This object has text sent to it when it is created. The text then moves up and reduces in size.

There is no sprite for this object.

Create Event

```
/// @description Set Up  
size=5;//Start Size 5x font size  
x=room_width/2;//Set In Center  
start=false;//prevent changing size intil in room
```

Sets initial values.

Step Event

```
/// @description Move & Change Siz  
if y<room_height //if on screen  
{  
    size==0.024; //how quickly to scale down  
}  
y=4;//how quickly to move up  
  
if size<0.01 //when invisible  
{  
    instance_destroy();  
    show_debug_message("Destroyed");  
}  
if y<room_height //if on screen  
{  
    size==0.024; //how quickly to scale down  
}
```

Checks if the instance is on screen, if it is reduces size each step.

y=4;//how quickly to move up

Changes the value of y by -4 each step, making the instance move upwards.

```
if size<0.01 //when invisible
{
    instance_destroy();
    show_debug_message("Destroyed");
}
```

Destroys when size is below 0.01.

Draw Event

```
/// @description Draw The Text
draw_set_font(font_intro_text);
draw_set_colour(c_white);
draw_set_halign(fa_center);
draw_set_valign(fa_middle);
draw_text_transformed(x,y,text,size,size,0);
```

Sets the drawing style, and then draws the text with the given size.

obj_emitter

This object creates the sparkle objects for the star field effect.

There is no sprite for this object.

Create Event

```
/// @description Set Up  
alarm[0]=5;
```

Sets alarm 0 to 5 (1/6 of a second)

Alarm 0 Event

```
/// @description Create Sparkle  
alarm[0]=room_speed/2;  
sparkle=instance_create_layer(room_width/2,room_height/2, "Instances",obj_sparkle);  
sparkle.direction=irandom(360);  
sparkle.speed=4+random(4);
```

alarm[0]=room_speed/2;

Resets alarm 0 to $\frac{1}{2}$ a second.

```
sparkle=instance_create_layer(room_width/2,room_height/2, "Instances",obj_sparkle);  
sparkle.direction=irandom(360);  
sparkle.speed=4+random(4);
```

Creates an instance of **obj_sparkle** in the middle of the room, sets it to move outwards at a speed between 4 and 8, in a random direction.

This Section Is For The Game's How To Play Room

The objects and code used here is a duplicate of that for the game credits, with text in the Create Event for **obj_intro_control** changes accordingly.

This Section Is For The Game Room

obj_player

This is the main player object that the user will control. It is set up to move on keypesses, within a given region, and move back to it's starting position when no key is pressed. This object can also shoot projectiles (when they are active) with the mouse buttons. It also checks for collisions with enemies or enemies' projectiles. It also draws a health bar at the bottom of the window.

This object uses the sprite **spr_player**, which is the same as that used for the menu room. It is shown again for consistency, in *Figure 7_27* :

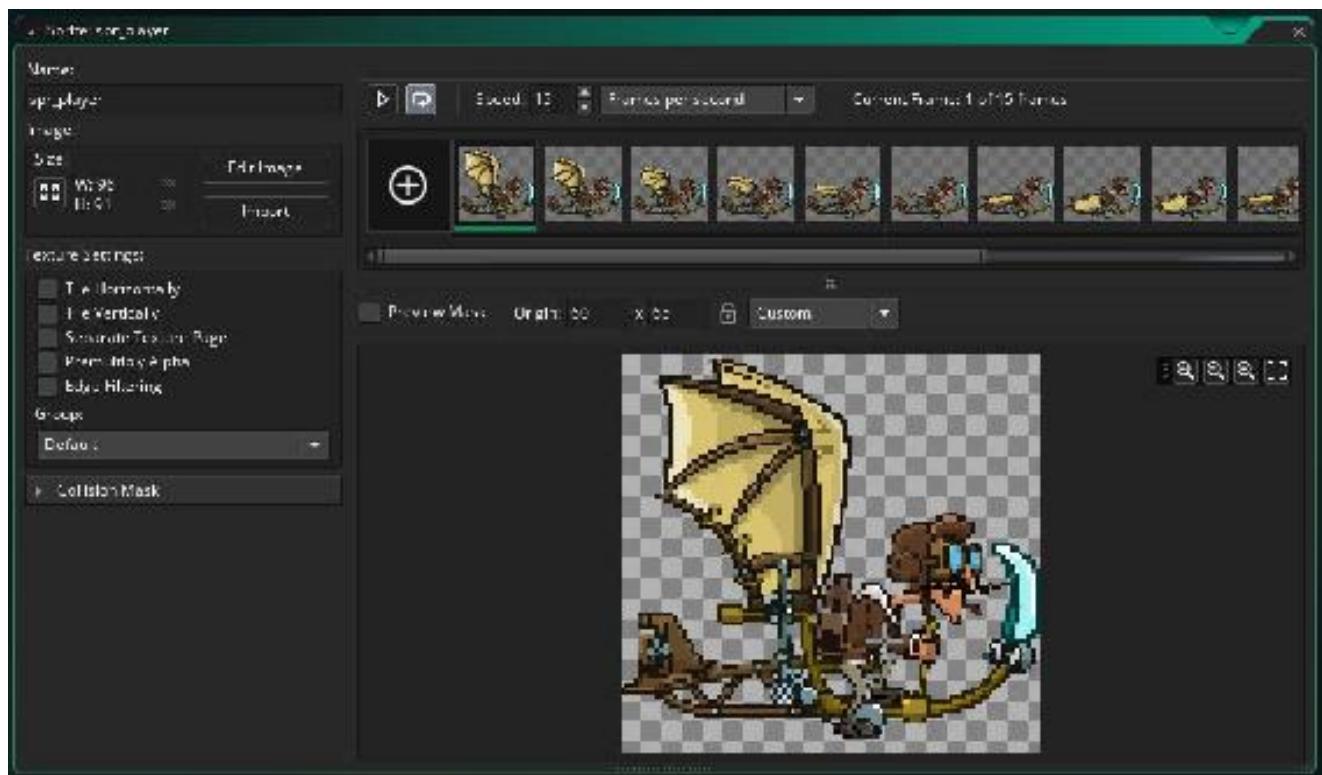


Figure 7_27: Showing sprite used for the player.

Create Event

```
/// @description Set up
flying_level=room_height/2;
xx=x;
power_active=false;
shield=0;
alarm[10]=room_speed*6;
can_shoot_1=true;
can_shoot_2=true;

//show info
info=instance_create_layer(x,y,"Foreground",obj_info)
;
info.image_index=1;

//for damage
flash=false;
```

```
flying_level=room_height/2;
xx=x;
ang=0;
power_active=false;
shield=0;
```

This sets the initial values needed. flying_level and xx will be used later for setting the parallax effect and moving the player back to the starting point. power_active sets whether the power weapon upgrade is active or not. The value shield is used to determine if the player has a shield, which be set to 1 when active and then count down.

```
alarm[10]=room_speed*6;
```

alarm 10 is used to trigger the power weapon (see [Alarm 10 Event](#)).

can_shoot_1=true;
can_shoot_2=true;

These two flags are used for players 1 and 2 weapons, which the player can fire using mouse buttons when it is active.

info=instance_create_layer(x,y,"Foreground",obj_info)
;
info.image_index=1;

This creates an instance of **obj_info** and tells it to display subimage 1.

flash=false;

This flag is used to show the player flash or not, when damaged. This used in unison with [Alarm 0](#), see below.

Step Event

```
/// @description Object management
global.difference=(flying_level-y)/10;//set as a global
value as it will be used for parallax background
image_angle=global.difference;
//keep in screen
y=clamp(y,80,room_height-80);
//move
//up down
nokey=keyboard_check(ord("W"))+keyboard_check(or
d("S"));
if nokey==0
{
    if y<flying_level y+=3;
    if y>flying_level y-=3;
}
if keyboard_check(ord("W"))
{
    y=6;
}
if keyboard_check(ord("S"))
{
    y+=6;
}

//forward back
if keyboard_check(ord("D"))
{
```

```

    x+=6;
}
if keyboard_check(ord("A"))
{
    x-=6;
}
x=clamp(x,xx-40,xx+180);
nokey=keyboard_check(ord("A"))+keyboard_check(ord("D"));
if nokey==0
{
    if x<xx x+=3;
    if x>xx x-=3;
}

//backgrounds

diff=(flying_level-y)/3;
layer_y("bg_1",550+diff);
layer_y("bg_2", (450+diff)/2);
layer_y("bg_3", (-100+diff)/3);

if global.level>8 power_active=true;

```

This code:

global.difference=(flying_level-y)/10;//

And this:

```
diff=(flying_level-y)/3;  
layer_y("bg_1",550+diff);  
layer_y("bg_2", (450+diff)/2);  
layer_y("bg_3", (-100+diff)/3);
```

Adjusts the y (vertical) position of the background, based on the player's y position. This helps create a nice and smooth parallax effect.

```
y=clamp(y,80,room_height-80);
```

This checks the y value of the player and checks it is range of between 80 and room_height less 80, if it outside this range the value will be changed. This is a great function, which would need several lines of code if coded in directly.

```
nokey=keyboard_check(ord("W"))+keyboard_check(ord("S"));  
if nokey==0  
{  
    if y<flying_level y+=3;  
    if y>flying_level y-=3;  
}
```

If the player is not pressing a key to move up or down, this code will move the player back to its starting value, at 3 pixels per step.

```
if keyboard_check(ord("W"))  
{  
    y=6;  
}  
if keyboard_check(ord("S"))  
{  
    y+=6;  
}
```

This checks for a keypress and then changes the player's y position accordingly at 6 pixels per step that the key is held down

```

//forward back
if keyboard_check(ord("D"))
{
    x+=6;
}
if keyboard_check(ord("A"))
{
    x-=6;
}
x=clamp(x,xx-40,xx+180);
nokey=keyboard_check(ord("A"))+keyboard_check(ord("D"));
if nokey==0
{
    if x<xx x+=3;
    if x>xx x-=3;
}

```

This code works in same way as moving up and down, instead it moves the player along the horizontal x position.

if global.level>8 power_active=true;

This checks if the player has upgraded to level 8, if so then the power weapon becomes active.

Alarm 0 Event

```

/// @description Flash Control
flash=false;

```

This sets the flag for flash back to false, this tells the game to no longer show player damage.

Alarm 1 Event:

```
/// @description Weapon 1 Control  
can_shoot_1=true;
```

Allows the player to shoot weapon 1 once again. This alarm is set when the player fires bullet 1, this flag is then set to false. This prevents the player shooting again until set back to true, preventing the player from shooting too often.

Alarm 2 Event

```
/// @description Weapon 2 Control  
can_shoot_2=true;
```

Allows player to shoot weapon 2 (if active)

Alarm 10 Event

```
/// @description Power Weapon
alarm[10]=room_speed*6;

if power_active
{
    pow=instance_create_layer(x,y,"Missiles",obj_power_bullet);
    pow.direction=image_angle;
    pow.image_angle=image_angle;
    pow.speed=8;
}
```

Restarts the alarm for 6 seconds, and if active creates an instance of the power weapon.

Alarm 11 Event

```
/// @description Sheild Control
shield=0.05;
if shield>0 alarm[11]=room_speed;
else
    shield=0;
```

Reduces the value of shield and resets alarm 11 if more than 0, if less the 0 the value of shield is set to 0 and the alarm is not reset. If the value of shield is more than 0 it means it is active and the player has protection against enemy weapons and asteroids.

Draw Event

```
/// @description Drawing Stuff

if flash==true && shield==0
{
```

```

    draw_sprite_ext(sprite_index,image_index,x,y,1,1,0,c_red,1);
}
else
{
    draw_self();
}
draw_sprite_ext(spr_shield,0,x,y,1,1,0,c_white,shield);
draw_set_alpha(0.3);
draw_healthbar(5,room_height-100,room_width-5,room_height-5,
(100/health)*health,c_black,c_red,c_green,0,true,true);
draw_set_alpha(1);

```

```

if flash==true && shield==0
{
    draw_sprite_ext(sprite_index,image_index,x,y,1,1,0,c_red,1);
}
else
{
    draw_self();
}

```

This code will draw the player with a red hue if it has damage, or just draw the set sprite otherwise.

draw_sprite_ext(spr_shield,0,x,y,1,1,0,c_white,shield);

This draws the shield over the player's sprite, with an alpha value of shield. This means that if the shield has a value 0 it's alpha will 0 and this not visible.

draw_set_alpha(0.3);

```
draw_healthbar(5,room_height-100,room_width-5,room_height-5,  
(100/health)*health,c_black,c_red,c_green,0,true,true);  
draw_set_alpha(1);
```

This sets the alpha value to 30% and then draws a health bar with the player's health at the bottom of the window.

Note: If you set the alpha value in code, like above, it will apply to all game objects until it is changed again, this we set it back to 1 (100%) when done drawing the health bar.

Global Left Pressed Event

```
/// @description Weapon 1  
if can_shoot_1==false exit;  
var xx = x + lengthdir_x(64, image_angle);  
var yy = y + lengthdir_y(64, image_angle);  
missile=instance_create_layer(xx,yy,"Missiles",obj_player_arrow);  
missile.image_angle=image_angle;  
missile.direction=image_angle;  
missile.speed=6;  
can_shoot_1=false;  
alarm[1]=room_speed*1.5;
```

This checks if the player can shoot (can_shoot_1 is true), and exits if false. If true it will create an instance of obj_player_arrow, with the same angle and direction as the player is. It then sets the flag can_shoot_1 to false (preventing the player from shooting until true again), an alarm 1 is set for 1.5 seconds.

```
var xx = x + lengthdir_x(64, image_angle);  
var yy = y + lengthdir_y(64, image_angle);
```

These two line simplify some complex maths, basically it sets two variables that stay at a point on the player's sprite, taking into account the player's image angle.

Global Right Pressed Event

```
/// @description Weapon 2
if global.level<13 exit;
if can_shoot_2==false exit;
alarm[1]=room_speed;
show_debug_message("Missile 2");
var xx = x + lengthdir_x(64, image_angle);
var yy = y + lengthdir_y(64, image_angle);
missile=instance_create_layer(xx,yy,"Missiles",obj_player_missile);
missile.image_angle=image_angle;
missile.direction=image_angle;
missile.speed=10;
can_shoot_2=false;
alarm[2]=room_speed*1.5;
```

Some logic here as previous event.

Key Down H Event

```
/// @description For Testing
health=100;
```

Put this in for testing, just resets health to 100.

Collision With obj_ball Event

```
/// @description Damage
if shield>0
{
    //do nothing
}
else
```

```
{  
    audio_play_sound(snd_lose_health,1,false);  
    health=5;  
    flash=true;  
    alarm[0]=room_speed/2;  
}  
with (other) instance_destroy();
```

Checks if shield is active, if it is skips the next block and destroys the ball. If the player does not have a shield it plays a sound, reduces health (which is automatically a global value that can be accessed by all game instances – so no global. Is needed), sets flash to true so the player draws with a red hue and sets the alarm 0 to 0.5 seconds.

Collision With obj_en_parent_projectile Event

```
/// @description Collision Control
if shield>0
{
    //do nothing
}
else
{
    audio_play_sound(snd_lose_health,1,false);
    health=other.strength;
    flash=true;
    alarm[0]=room_speed/2;
}
with (other) instance_destroy();
show_debug_message("Collision With Enemy
Detected");
show_debug_message("Health is now "+string(health));
```

Works in a similar way to the previous collision event. This collision event occurs when any instance that has this set as a parent collides with the player – this allows you to perform one collision check on various object instances – great if you have a game with many ten's or hundred's of instance.

health=other.strength;

This line will reduce the health by what ever value strength in the other holds, again great if you have many instances that have different strength values.

```
show_debug_message("Collision With Enemy
Detected");
show_debug_message("Health is now "+string(health));
```

This was covered in the debug chapter, it sends output to the console window.

Collision With obj_enemy_parent Event

```
/// @description Collision
if shield>0
{
    other.hp=global.pow;
}
else
{
    audio_play_sound(snd_lose_health,1,false);
    health=1;
    flash=true;
    alarm[0]=room_speed/2;
}
```

Again checks for a collision and whether shield is active or not.

other.hp=global.pow;

This code will reduce the colliding enemy's health by the current global.pow value.

Collision With obj_power_up Event

```
/// @description Increase Level  
with other instance_destroy();  
audio_play_sound(snd_power_up,1,false);  
global.level++;  
  
if global.level==2  
{  
    //show_message("Top Shooter");  
    instance_create_layer(x,y,"Player",obj_extra_1_top);  
    //show info  
    info=instance_create_layer(x,y,"Foreground",obj_info);  
    info.image_index=2;  
}  
  
if global.level==3  
{  
    ////show_message("Bottom Shooter");  
    instance_create_layer(x,y,"Player",obj_extra_1_bottom);  
    //show info  
    info=instance_create_layer(x,y,"Foreground",obj_info);  
    info.image_index=3;  
}  
  
if global.level==4
```

```
{  
    //show_message("Top Laser");  
    instance_create_layer(x,y,"Player",obj_extra_2_top);  
    //show info  
    info=instance_create_layer(x,y,"Foreground",obj_info);  
    info.image_index=4;  
}  
  
if global.level==5  
{  
    //show_message("Bottom Laser");  
    instance_create_layer(x,y,"Player",obj_extra_2_bottom);  
    //show info  
    info=instance_create_layer(x,y,"Foreground",obj_info);  
    info.image_index=5;  
}  
if global.level==6  
{  
    //show_message("Triple Shots");  
    //show info  
    info=instance_create_layer(x,y,"Foreground",obj_info);  
    info.image_index=6;  
}  
if global.level==7
```

```
{  
    //show_message("Mine 1");  
    instance_create_layer(x,y,"Player",obj_mine_1);  
    //show info  
    info=instance_create_layer(x,y,"Foreground",obj_info);  
    info.image_index=7;  
}  
if global.level==8  
{  
    //show_message("Mine 2");  
    instance_create_layer(x,y,"Player",obj_mine_2);  
    //show info  
    info=instance_create_layer(x,y,"Foreground",obj_info);  
    info.image_index=8;  
}  
if global.level==9  
{  
    //show_message("Plasma");  
    //show info  
    info=instance_create_layer(x,y,"Foreground",obj_info);  
    info.image_index=9;  
}  
if global.level==10  
{  
    //show_message("Mine 3 & 4");
```

```
instance_create_layer(x,y,"Player",obj_mine_3);
instance_create_layer(x,y,"Player",obj_mine_4);
//show info
info=instance_create_layer(x,y,"Foreground",obj_info);
info.image_index=10;
}
if global.level==11
{
//show_message("Faster Shots");
global.shoot_speed++;
//show info
info=instance_create_layer(x,y,"Foreground",obj_info);
info.image_index=11;
}
if global.level==12
{
//show_message("Faster Shots");
global.shoot_speed++;
//show info
info=instance_create_layer(x,y,"Foreground",obj_info);
info.image_index=12;
}
if global.level==13
{
//show_message("Homing Missile");
```

```

///show info
info=instance_create_layer(x,y,"Foreground",obj_info);
info.image_index=13;
}
if global.level>13
{
    global.pow++;
//show info
info=instance_create_layer(x,y,"Foreground",obj_info);
info.image_index=14;
}

```

This will destroy the bonus item and then check the current level, it will then create an instance of **obj_info** and tell it to use the given subimage. This is used to tell the player what bonus they got for collecting the item.

Collision With obj_shield_collect Event

```

/// @description Activate Shield
shield=1;
alarm[11]=room_speed;
with other instance_destroy();
audio_play_sound(snd_shield,1,false);

```

This will set the shield to 1 (active) and set an alarm. It will also destroy the shield collect item and play a sound.

obj_player_weapon_parent

This object is used so that we can reduce the number of collision event calls in the game's enemies. A parent object has children (with the parent set in the child object), allowing you check against a parent, which check for all children. This is used so we can reduce the number of collision checks needed when checking enemy and player projectile collisions. 3 is not a huge number, but when you make a game with lots of weapons, this can reduce the coding time and complexity needed.

Is the parent of the following player's projectiles.

obj_bullet_1, obj_bullet_2, and obj_player_arrow

There is no code for this object.*obj_mine_parent*

Is the parent of ***obj_mine_1, obj_mine_2, obj_mine_3*** and ***obj_mine_4***

Again, a parent is used to reduce collision event checks.

obj_mine_1

This is an object that rotates around the player, which can damage enemies or destroy enemy projectiles.

This has the sprite **spr_mine**, as shown in *Figure 7_28* :

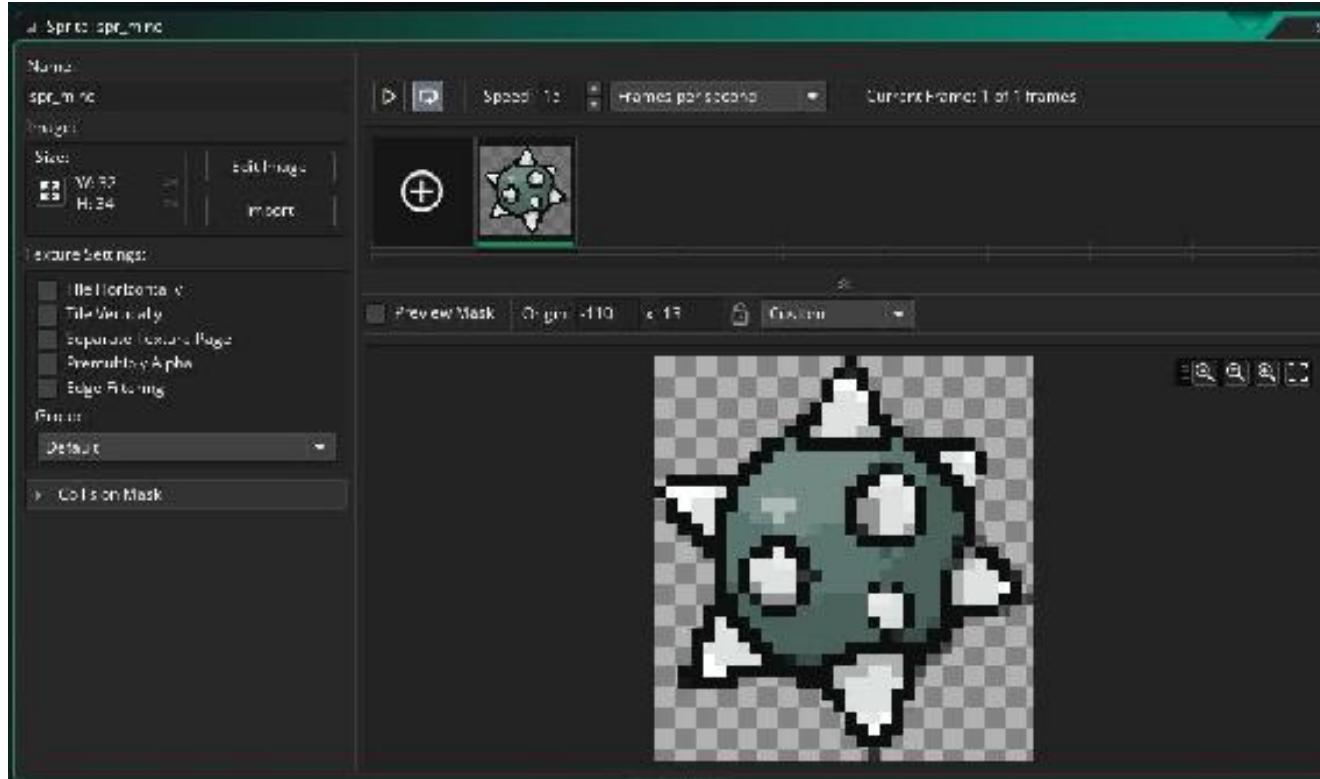


Figure 7_28: Showing sprite set up for spr_mine

You should note the sprite origin as -110 x 13. This negative value is used to make rotating it around the player, without complex code.

Create Event

```
/// @description Set Up  
my_power=50;
```

Sets the strength of the mine.

Step Event

```
/// @description Rotate & Clamp To Player & Power  
image_angle++;  
x = obj_player.x;  
y = obj_player.y;
```

```
my_power=50*global.pow;
```

```
image_angle++;
```

This makes the sprite rotate around it's origin by 1' each step.

x = obj_player.x;

y = obj_player.y;

Keeps the mine clamped with the player, so if the player moves the mine will stay relative to it.

The final line sets the strength of the mine, based on the current value of global.pow.

obj_mine_2

Does the same as **obj_mine_1** and uses the same sprite.

Create Event

```
/// @description Set Up  
my_power=50;
```

Step Event

```
/// @description Rotate & Clamp To Player & Power  
image_angle=obj_mine_1.image_angle+180  
x = obj_player.x;  
y = obj_player.y;  
my_power=50*global.pow;
```

image_angle=obj_mine_1.image_angle+180

This keeps the angle 180 off of mine 1, so that they are stay on either side of the player instance.

obj_mine_3

Does the same as **obj_mine_1** and uses the same sprite.

Create Event

```
/// @description Set Up  
my_power=50;
```

Step Event

```
/// @description Rotate & Clamp To Player &  
Powerimage_angle=obj_mine_1.image_angle+180  
image_angle=obj_mine_1.image_angle+90  
x = obj_player.x;  
y = obj_player.y;  
my_power=50*global.pow;
```

As previous object, though at 90' off **obj_mine_1**'s angle.

obj_mine_4

Does the same as **obj_mine_1** and uses the same sprite.

Create Event

```
/// @description Set Up  
my_power=50;
```

Step Event

```
/// @description Rotate & Clamp To Player & Power  
image_angle=obj_mine_1.image_angle-90  
x = obj_player.x;  
y = obj_player.y;  
my_power=50*global.pow;
```

As previous object, though at -90° off **obj_mine_1**'s angle.

This set up ensures that once all 4 mines are active, they smoothly rotate around the player at equal distances.

obj_player_arrow

This is the weapon that the player starts with, it is fired by the player pressing the left mouse button (when the weapon is active and can be fired).

The sprite for this instance is sprite **spr_rocket** as shown in *Figure 7_29* :

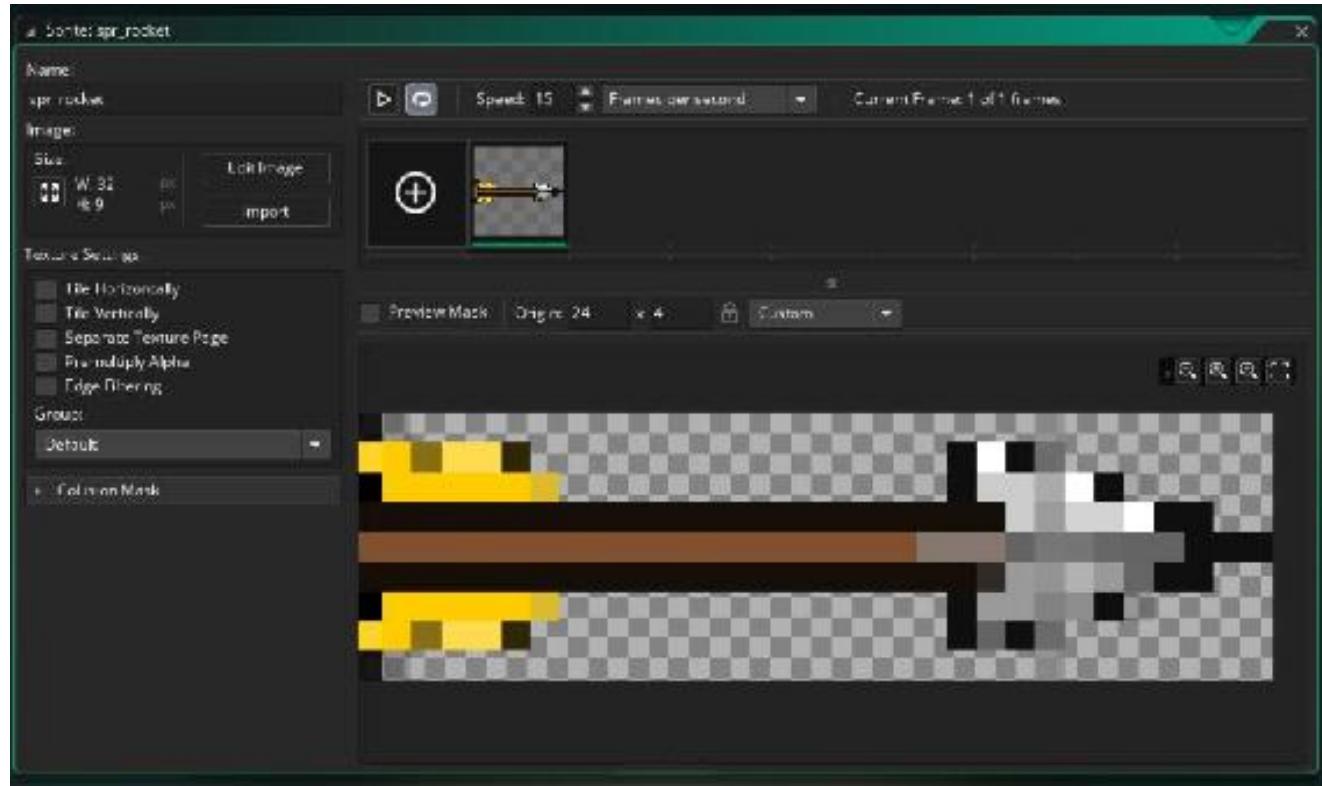


Figure 7_29: Showing player's primary weapon

Create Event

```
/// @description Set Up  
my_power=1*global.pow;  
audio_play_sound(snd_rocket,1,false);
```

Sets the power and plays a sound.

Outside Room Event

```
/// @description Destroy when not needed  
instance_destroy();
```

Destroys once outside the room, good practice and helps prevent memory issues.

obj_player_missile

This is one the weapons the player can upgrade to. This weapon will find a target enemy and attempt to hit it.

The sprite for this is **spr_player_missile_2**, as shown in *Figure 7_30* below:

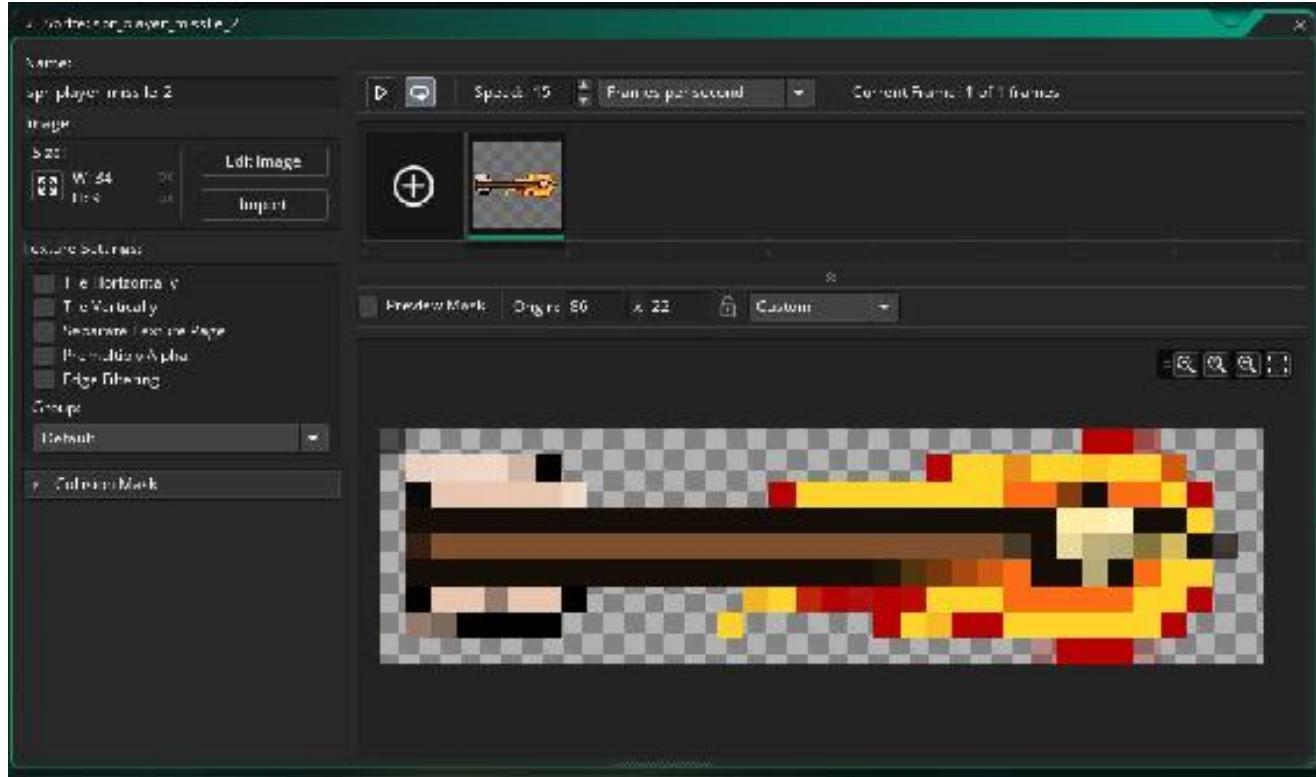


Figure 7_30: Sprite set up for spr_player_missile

Create Event

```
/// @description Set Up
live=true;
active=true;
alarm[0]=room_speed*5;

//look for initial target
if instance_exists(obj_enemy_parent)
{
    target=instance_nearest(mouse_x,mouse_y,obj_enemy_parent);
```

```
}
```

else

```
{
```

```
    target=noone;
```

```
}
```

start_hp=200;

hp=200;

my_power=5*global.pow;

audio_play_sound(snd_missile,1,false);

live=true;

active=true;

alarm[0]=room_speed*5;

Sets a flag as true for live and active. Sets an alarm 0 for 5 seconds.

if instance_exists(obj_enemy_parent)

```
{
```

```
    target=instance_nearest(mouse_x,mouse_y,obj_enemy_parent);
```

```
}
```

else

```
{
```

```
    target=noone;
```

```
}
```

Looks for a target (that has the parent obj_enemy_parent), sets the value of target to the enemy's id (each instance has a different id), otherwise sets it to noone.

start_hp=200;

hp=200;

This object has an hp, which counts down. When run out the missile destroys.

```
my_power=5*global.pow;  
audio_play_sound(snd_missile,1,false);
```

Sets the power of the missile and plays a sound.

Step Event

```
/// @description Look For Target  
if instance_exists(obj_enemy_parent)  
{  
    if target=noone  
    target=instance_nearest(x,y,obj_enemy_parent);  
}  
else  
{  
    target=noone;  
}  
//track target  
if target!=noone  
if instance_exists(target)  
{  
    tx = target.x;  
    ty = target.y;  
    direction = scr_angle_rotate(direction,  
point_direction(x, y, tx, ty),3);  
    image_angle = direction;  
}
```

```

else
{
    if instance_exists(obj_enemy_parent)
    {
        target=instance_nearest(x,y,obj_enemy_parent);
    }
    else
    {
        target=noone;
    }
}

if instance_exists(target) && target.x<0 target=noone;

//health bar
hp--;
if hp<1
{
    instance_destroy();
}

if instance_exists(obj_enemy_parent)
{
    if target=noone
        target=instance_nearest(x,y,obj_enemy_parent);
}

```

Checks for a new target if **obj_enemy_parent** exists and the weapon does not currently have a target set.

```
if target!=noone  
if instance_exists(target)  
{  
    tx = target.x;  
    ty = target.y;  
    direction = scr_angle_rotate(direction,  
point_direction(x, y, tx, ty),3);  
    image_angle = direction;  
}
```

If the weapon has a target and the target still exists, then changes direction to move towards the target.

```

else
{
    if instance_exists(obj_enemy_parent)
    {
        target=instance_nearest(x,y,obj_enemy_parent);
    }
    else
    {
        target=noone;
    }
}

```

Looks for a new target again if current does not exist.

```

hp--;
if hp<1
{
    instance_destroy();
}

```

Reduces hp by 1 each step (-- is basically the same as hp-=1 or hp=hp-1), if less than 1 then destroys itself.

Draw Event

```

/// @description Drawing
draw_self();
if instance_exists(target)
{
    draw_sprite(spr_locked,0,target.x,target.y);
    draw_set_colour(c_lime);
    draw_line_width(x,y,target.x,target.y,3);
}

```

```
}
```

scr_healthbar(0,0,40);

draw_self();

Draws self (if you put any code in a Draw Event that doesn't force drawing of a sprite, it will not draw – adding this forces it to draw).

```
if instance_exists(target)
{
    draw_sprite(spr_locked,0,target.x,target.y);
    draw_set_colour(c_lime);
    draw_line_width(x,y,target.x,target.y,3);
}
```

This will check if the target exists, and if it does it will draw a lime green line from the weapon to the target.

scr_healthbar(0,0,40);

This will draw a small healthbar relative to the weapons current location.

obj_bullet_1

This object is the projectile that will fire the player's secondary weapon (when player has collected enough power ups and the weapon is active).

Figure 7_31 shows the sprite set up:

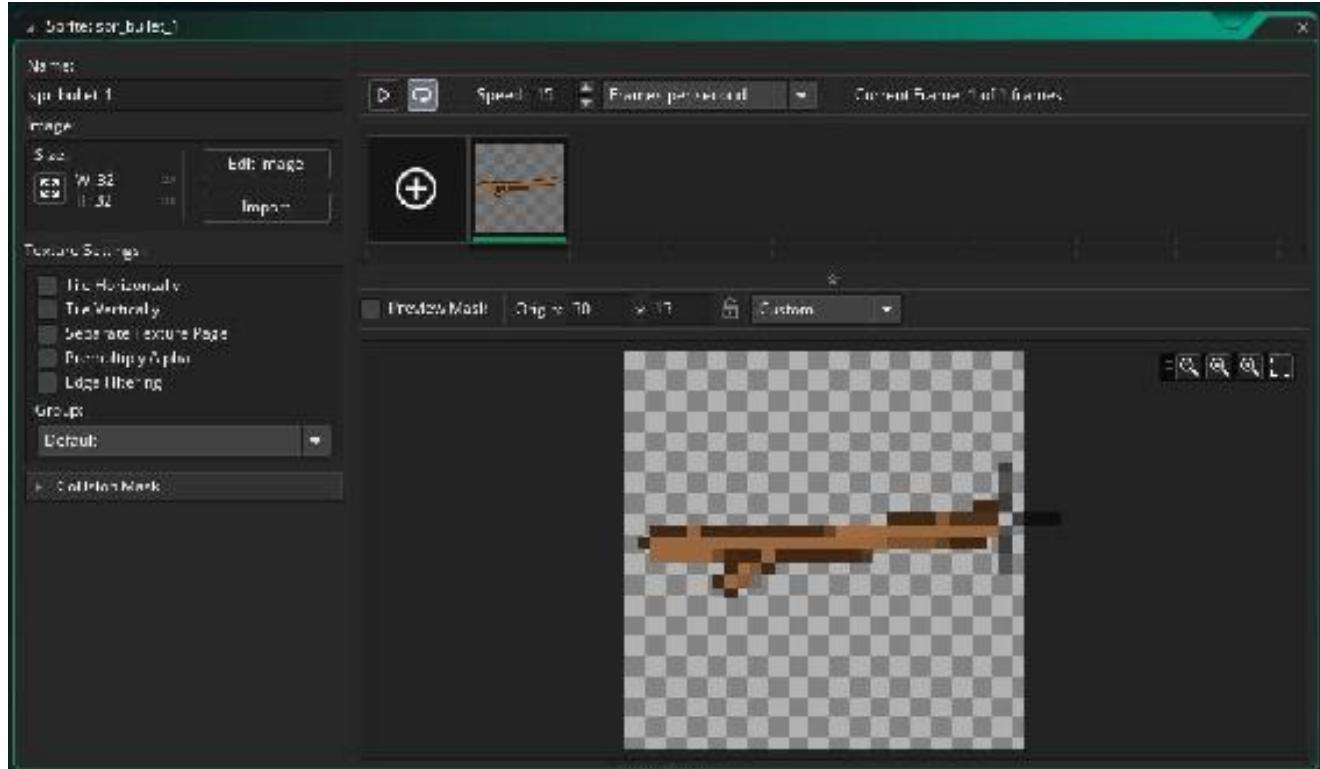


Figure 7_31: Showing sprite set up for spr_bullet_1

Create Event

```
/// @description Set Up  
my_power=1*global.pow;  
audio_play_sound(snd_bullet_1,1,false);
```

Sets the strength of the weapon and plays a sound.

Outside Room Event

```
/// @description Destroy when not needed  
instance_destroy();
```

Destroys the instance when outside the room.

obj_bullet_2

This object is the projectile that will fire the player's triary weapon (when player has collected enough power ups and the weapon is active).

This uses the same sprite as shown in *Figure 7_30* .

Create Event

```
/// @description Set Up  
my_power=2*global.pow;  
audio_play_sound(snd_bullet_2,1,false);
```

Sets the strength of the weapon and plays a sound.

Outside Room Event

```
/// @description Destroy When no longer needed  
instance_destroy();
```

Destroys the instance when outside the room.

obj_power_bullet

This is a power weapon that becomes active when the player has collected enough power ups.

Figure 7_32 shows spr_power_effect, which is assigned to this object:

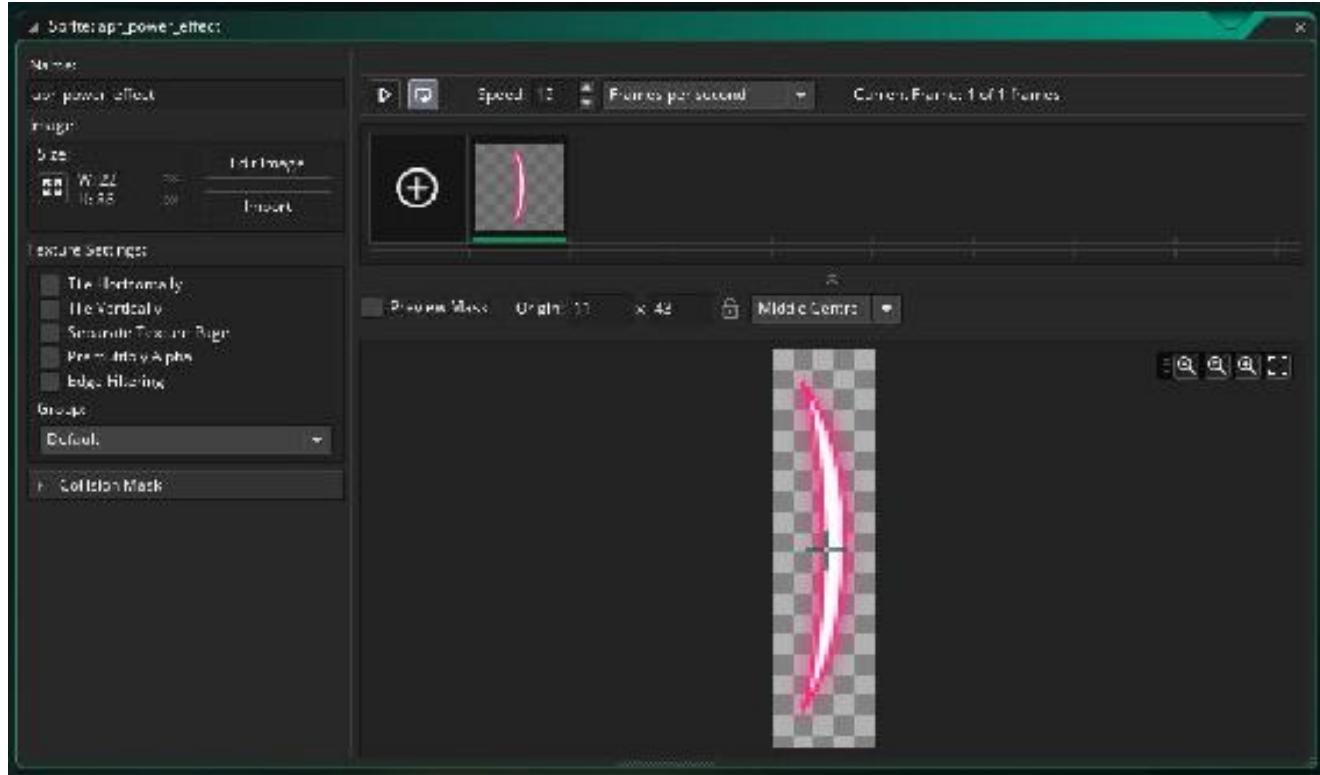


Figure 7_32: Showing the sprite spr_power_effect

Create Event

```
/// @description Set Up
image_xscale=0.1;
image_yscale=0.1;
my_power=2*global.pow;
audio_play_sound(snd_power,1,false);
```

Sets the initial scale (size), power and plays a sound.

Step Event

```
/// @description Increase Size
image_xscale+=0.06;
```

```
image_yscale+=0.06;
```

Gradually increases size.

Outside Room Event

```
/// @description Destroy When Done
instance_destroy();
```

Destroy when done with, to prevent memory errors.

obj_extra_1_top

This weapon will be created when the player has collected enough power ups. It shoots automatically.

Figure 7_33 below shows the sprite assigned for this object:

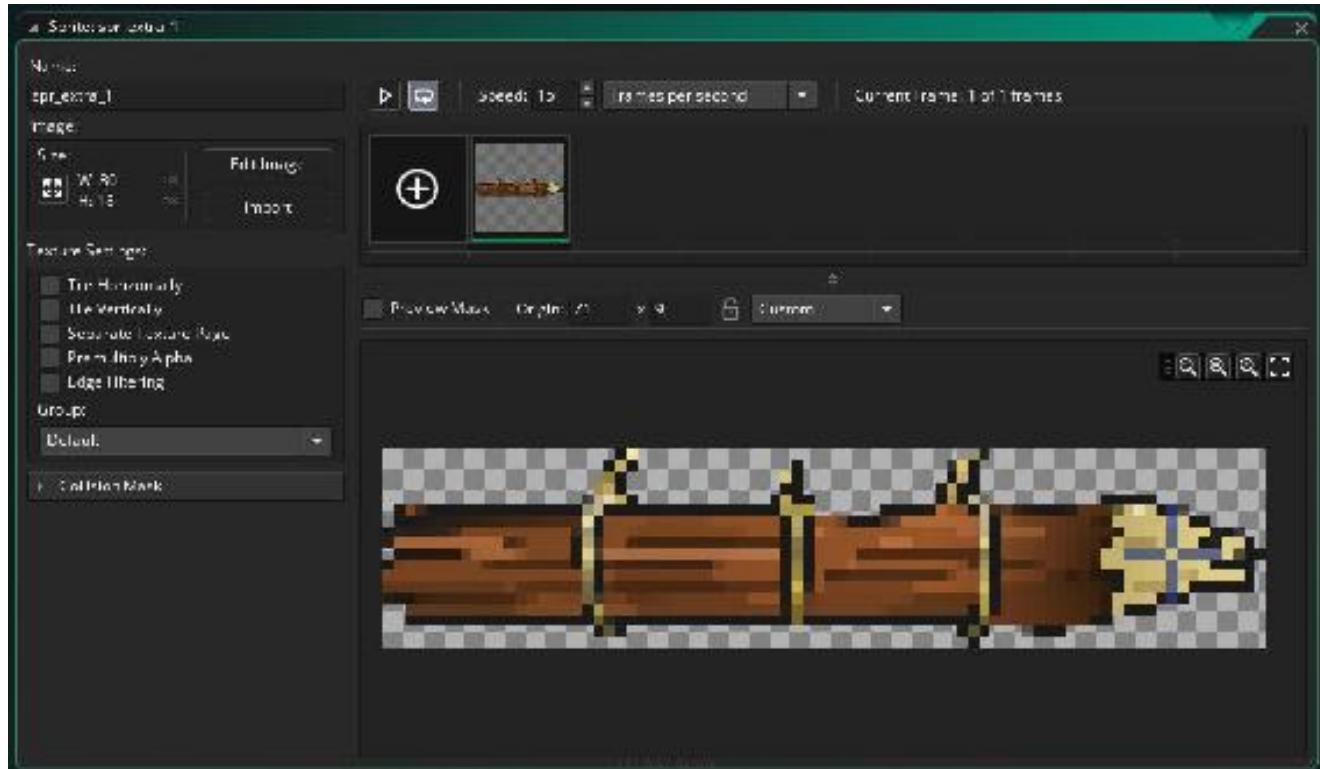


Figure 7_33: Showing sprite spr_extra_1

Create Event

```
/// @description Set initial alarm  
alarm[0]=room_speed*2;
```

Sets an alarm for firing control

Step Event

```
/// @description Keep In Relation to player location  
and angle  
xx = obj_player.x + lengthdir_x(64,  
obj_player.image_angle);  
yy = obj_player.y + 100 + lengthdir_y(128,  
obj_player.image_angle);  
image_angle=obj_player.image_angle;  
x=xx;  
y=yy-204;
```

Keeps the angle and position relative to the player.

Alarm 0 Event

```
/// @description Shoot Projectiles  
alarm[0]=  
(room_speed*1.5)+room_speed*5/global.shoot_speed;  
ang=image_angle;  
xx = x + lengthdir_x(32, ang);  
yy = y + lengthdir_y(32, ang);  
  
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);  
bullet.direction=ang;  
bullet.speed=7;  
bullet.image_angle=ang;  
if global.level<6 exit;
```

```
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
bullet.direction=ang-30;
bullet.speed=7;
bullet.image_angle=ang-30;

bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
bullet.direction=ang+30;
bullet.speed=7;
bullet.image_angle=ang+30;
```

alarm[0]=

(room_speed*1.5)+room_speed*5/global.shoot_speed;

Resets the alarm.

ang=image_angle;

xx = x + lengthdir_x(32, ang);

yy = y + lengthdir_y(32, ang);

Keeps it relative to the player.

```
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
```

bullet.direction=ang;

bullet.speed=7;

bullet.image_angle=ang;

Creates an instance of the bullet object, with a starting location relative the weapon source, and with given speed and angle.

if global.level<6 exit;

Exists if player is on less than level 6. If it is 6 or above then 2 more projectiles are created, with angles of +30' and -30' of the initial weapon.

obj_extra_1_bottom

This weapon will be created when the player has collected enough power ups. It shoots automatically.

This object uses the same sprite as shown in *Figure 7_33* previously.

Create Event

```
/// @description Set Alarm (synced with top)
alarm[0]=obj_extra_1_top.alarm[0];
level=1;
```

Step Event

```
/// @description Keep in relation to player
xx = obj_player.x + lengthdir_x(64,
obj_player.image_angle);
yy = obj_player.y + 100 + lengthdir_y(128,
obj_player.image_angle);
image_angle=obj_player.image_angle;
x=xx;
y=yy;
```

Alarm 0 Event

```
/// @description Shoot Projectiles
alarm[0]=obj_extra_1_top.alarm[0];
ang=image_angle;
xx = x + lengthdir_x(32, ang);
yy = y + lengthdir_y(32, ang);

bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);
bullet.direction=ang;
```

```
bullet.speed=7;  
bullet.image_angle=ang;  
if global.level<6 exit;  
  
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);  
bullet.direction=ang-30;  
bullet.speed=7;  
bullet.image_angle=ang-30;  
  
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_1);  
bullet.direction=ang+30;  
bullet.speed=7;  
bullet.image_angle=ang+30;
```

Works in the same way as the first weapon above.

obj_extra_2_top

This weapon will be created when the player has collected enough power ups. It shoots automatically.

The sprite for this object is shown below in *Figure 7_34* :

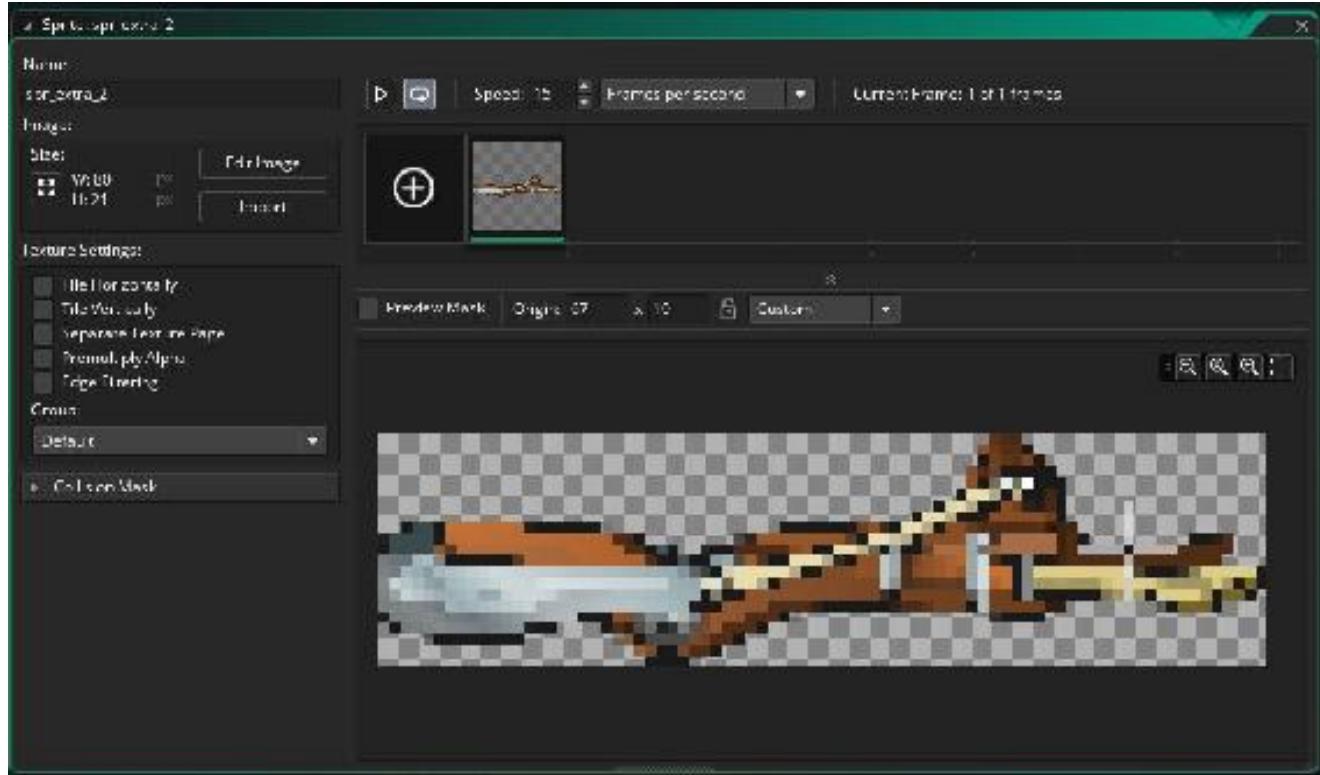


Figure 7_34: Showing sprite spr_extra_2

Create Event

```
/// @description Set initial alarm  
alarm[0]=room_speed*2;
```

Step Event

```
/// @description sync with player  
xx = obj_player.x + lengthdir_x(64,  
obj_player.image_angle);  
yy = obj_player.y + 100 + lengthdir_y(128,  
obj_player.image_angle);
```

```
image_angle=obj_player.image_angle;  
x=xx;  
y=yy-332;
```

Alarm 0 Event

```
/// @description Create Projectile  
alarm[0]=room_speed*4;  
ang=image_angle;  
xx = x + lengthdir_x(32, ang);  
yy = y + lengthdir_y(32, ang);  
  
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_2);  
bullet.direction=ang;  
bullet.speed=7;  
bullet.image_angle=ang;
```

No new code to comment on here.

obj_extra_2_bottom

This weapon will be created when the player has collected enough power ups. It shoots automatically.

This uses the same sprite as shown in *Figure 7_34* .

Create Event

```
/// @description Set Initial Alarm  
alarm[0]=obj_extra_2_top.alarm[0];
```

Step Event

```
/// @description SSync with player  
xx = obj_player.x + lengthdir_x(64,  
obj_player.image_angle);  
yy = obj_player.y + 100 + lengthdir_y(128,  
obj_player.image_angle);  
image_angle=obj_player.image_angle;  
x=xx;  
y=yy+128;
```

Alarm 0 Event

```
/// @description Cretae Projectile  
alarm[0]=obj_extra_2_top.alarm[0];  
ang=image_angle;  
xx = x + lengthdir_x(32, ang);  
yy = y + lengthdir_y(32, ang);
```

```
bullet=instance_create_layer(xx,yy,"Missiles",obj_bullet_2);  
bullet.direction=ang;  
bullet.speed=7;  
bullet.image_angle=ang;
```

No new code to comment on here.

obj_spawn_enemy

This object will act as a controller, and spawn enemies accordingly.

There is no sprite for this object.

Create Event

```
//@description Set Initial alarms  
alarm[0]=room_speed*8;  
alarm[1]=room_speed*16;  
alarm[2]=room_speed*60;
```

Sets some initial alarms.

Alarm 0 Event

```
/// @description Make Enemy Floor
alarm[0]=room_speed*8;
if instance_number(obj_boss)==0 &&
instance_number(obj_enemy_floor)==0
enemy=instance_create_layer(room_width+200,0,"Enemy",obj_enemy_floor);
```

Restarts the alarm, then checks if no instances of **obj_boss** or **obj_enemy_floor** are currently present, if not it creates an instance of **obj_enemy_floor**.

Alarm 1 Event

```
/// @description Make Enemy Air
alarm[1]=room_speed*16;
if instance_number(obj_boss)==0 &&
instance_number(obj_enemy_air)==0
enemy=instance_create_layer(room_width+200,0,"Enemy",obj_enemy_air);
```

Restarts the alarm, and then checks whether **obj_boss** or **obj_enemy_air** is present, if not an instance of **obj_enemy_air** is created.

Alarm 2 Event

```
/// @description Make Boss
alarm[0]=room_speed*60;
if instance_number(obj_boss)==0
enemy=instance_create_layer(room_width+200,0,"Enemy",obj_boss);
```

Checks whether an instance of **obj_boss** is currently present, if not it creates one.

obj_enemy_floor

This is the enemy that will move along the bottom of the window. It does not fire any projectiles.

Figure 7_35 shows the sprite set up for this object:

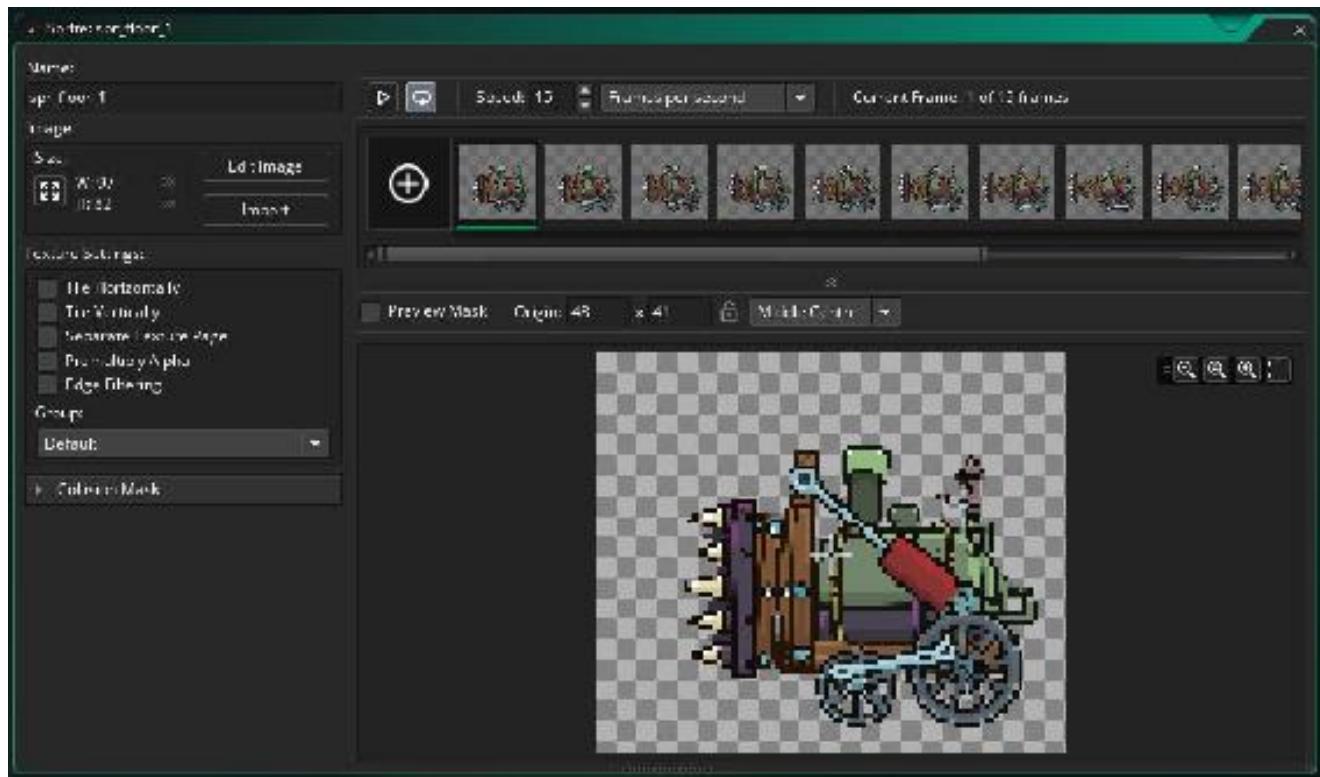


Figure 7_35: Showing the sprite spr_floor_1 for object obj_enemy_floor

Create Event

```
/// @description Set Up
start_hp=global.level*global.level;
hp=start_hp;
hspeed=-3.7;
```

This sets the instances starting hp values, used for drawing detecting when dead. hspeed is the horizontal speed, a negative value makes it move towards the left.

Step Event

```
/// @description Check pos, hp
if x<0 instance_destroy();
//if hp<1
if hp<1
{
    instance_destroy();
```

```

coin=instance_create_layer(x,y,"Player",obj_coin_bubble);
coin.value=start_hp;
for (var i = 0; i < 15; i += 1)
{
    part=instance_create_layer(x,y,"Score_Effect",obj_floor_parts);
    part.image_index=i;
}
}

//control y
y=layer_get_y("bg_1")+100;

```

if x<0 instance_destroy();

This line of code checks if it has gone off the left side of the window, if it has then it no longer needed and is destroyed.

```

if hp<1
{
    instance_destroy();
    coin=instance_create_layer(x,y,"Player",obj_coin_bubble);
    coin.value=start_hp;
}

```

Checks the value of hp, if it is less than 1, this code block is executed. It firstly creates a coin at the enemy's location and gives is a value equal to the starting hp value.

```

for (var i = 0; i < 15; i += 1)
{
    part=instance_create_layer(x,y,"Score_Effect",obj_floor_parts);
}

```

```
    part.image_index=i;  
}  
}
```

This creates instances of **obj_floor_parts** and assigns each a different `image_index`.

```
y=layer_get_y("bg_1")+100;
```

This makes the instance move up and down relative to background **bg_1**. This ensures that it matches the backgrounds movement.

Draw Event

```
/// @description Drawing  
draw_self();  
scr_healthbar(0,-40,60);
```

Draws the current subimage and a health bar.

Collision With **obj_mine_parent** Event

```
/// @description Reduce if in contact with mine  
hp=other.my_power;
```

This will reduces it's hp every step it is colliding with any mine.

Collision With **obj_player_weapon_parent** Event

```
/// @description Reduce hp  
hp=other.my_power;  
with other instance_destroy();  
audio_play_sound(snd_exp_1,1,false);
```

Reduces hp by the colliding instances `my_power` value, destroys the colliding instance, and plays a sound.

Collision with **obj_power_bullet** Event

```
/// @description Reduce hp  
hp=other.my_power/10;
```

Reduces hp when in collision with the **power_bullet** object, reduces by 1/10 of the **power_bullet**'s power.

obj_enemy_air

This enemy will move onto the window, then follow a set path on a loop. This will fire projectiles towards the player.

Figure 7_36 shows the sprite used for this object:

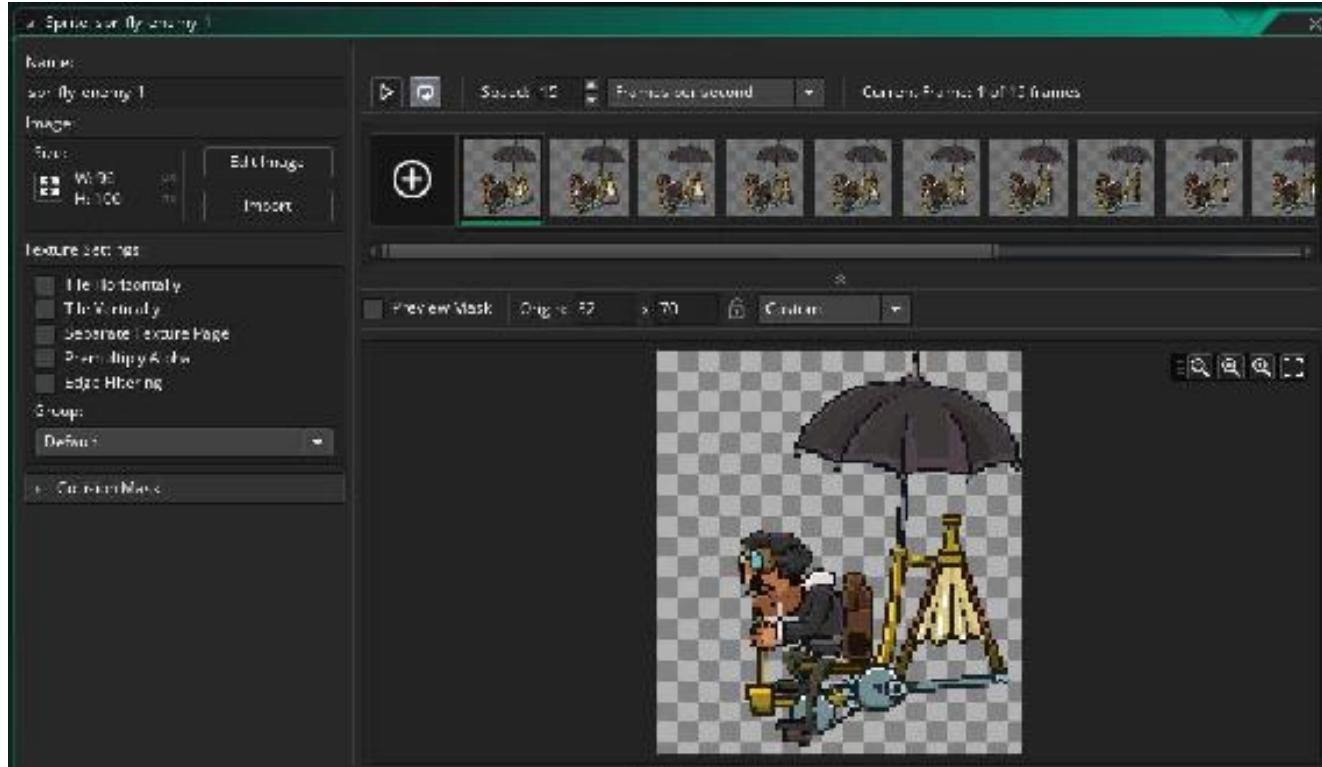


Figure 7_36: Showing sprite spr_fly_enemy_1

Create Event

```
/// @description Set Up
start_hp=global.level*global.level*2;
hp=start_hp;
hspeed=-4;
on_path=false;
y=irandom_range(200,room_height-200);
alarm[0]=room_speed*4;
```

Sets up initial hp values, a horizontal speed of -4 (moves to the left) and a flag for on_path.

```
y=irandom_range(200,room_height-200);
```

It's y position (up and down location) is set as a random value between 200 and the room's height less 200.

An alarm 0 is set for 4 seconds.

Step Event

```
/// @description Check Health and Start Path
if x<0 instance_destroy();
//if hp<1 && image_index==14
if hp<1
{
    instance_destroy();
    coin=instance_create_layer(x,y,"Player",obj_coin_bubble);
    coin.value=start_hp;
    for (var i = 0; i < 11; i += 1)
    {
        part=instance_create_layer(x,y,"Score_Effect",obj_air_parts);
        part.image_index=i;
    }
}

if on_path==false && x<room_width-400
{
    on_path=true;
    path_start(path_circle,4,path_action_restart,false);
}
```

The first part of this code works in a similar way to the code in **obj_enemy_floor_1**.

```
if on_path==false && x<room_width-400
{
    on_path=true;
    path_start(path_circle,4,path_action_restart,false);
}
```

This will check if it is currently moving on a path, if it is not and the y position is less than 400 from the right edge it will process the code block. The code block changes the flag for on_path so it won't trigger again and start moving on the defined path.

Alarm 0 Event

```
/// @description Create a Projectile
alarm[0]=room_speed*3;
arrow=instance_create_layer(x,y,"Enemy_Missile",obj_enemy_arrow_1);
ang=point_direction(x,y,obj_player.x,obj_player.y);
arrow.direction=ang;
arrow.image_angle=ang;
arrow.speed=12;
arrow.strength=global.level;
```

Restarts the alarm, creates an projectile that moves to the player's current location.

Draw Event

```
/// @description Draw Self & healthbar
draw_self();
scr_healthbar(30,-75,60);
```

Draws self and a health bar.

Collision With obj_mine_parent Event

```
/// @description Reduce hp
hp-=other.my_power;
```

Creates damage if in collision with one of the player's mines (when present).

Collision With obj_player_weapon_parent Event

```
/// @description REduce hp  
hp-=other.my_power;  
with other instance_destroy();  
audio_play_sound(snd_exp_1,1,false);
```

Reduces hp based on strength of player's projectile, destroys the projectile and plays a sound.

Collision With obj_power_bullet Event

```
/// @description reduce hp  
hp-=other.my_power/10;
```

Reduces hp by 1/10 the value of the projectile.

obj_enemy_arrow_1

This is a projectile that will cause damage to player if they collide it.

Figure 7_37 shows the sprite setup for this object.

Note: It is pointing to the right. In GMS2, right is equal to 0°, so when moving it will point in the correct direction.

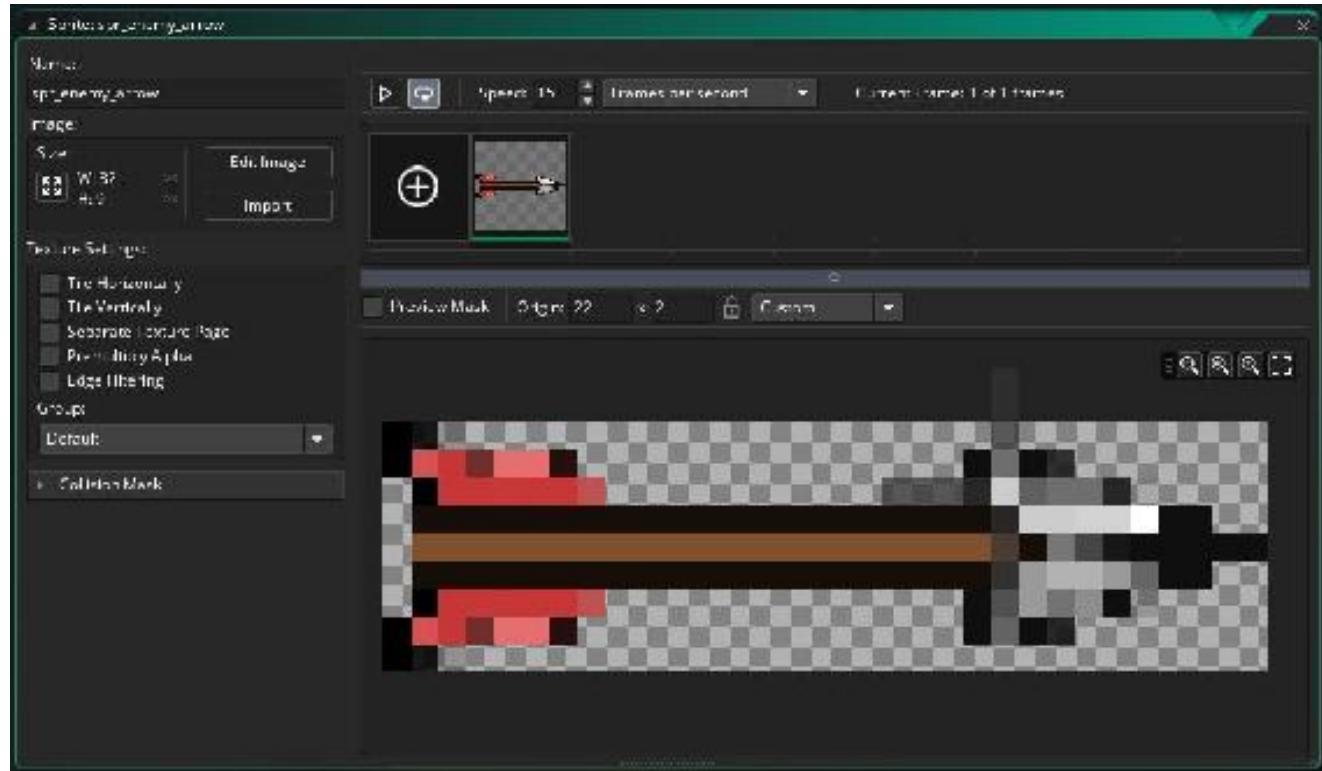


Figure 7_37: Showing sprite set up for spr_enemy_arrow

Collision With obj_mine_parent Event

```
/// @description destroy  
instance_destroy();
```

Destroys self if in collision with the player's mines.

Outside Room Event

```
/// @description destroy
```

instance_destroy();

Prevent a memory leak by destroying when no longer needed.

obj_boss

This boss object will follow a path, and then move up and down.

Figure 7_38 shows the sprite set up for this object.

Note: The sprite origin is the location that we will be fixing the boss's weapon to.

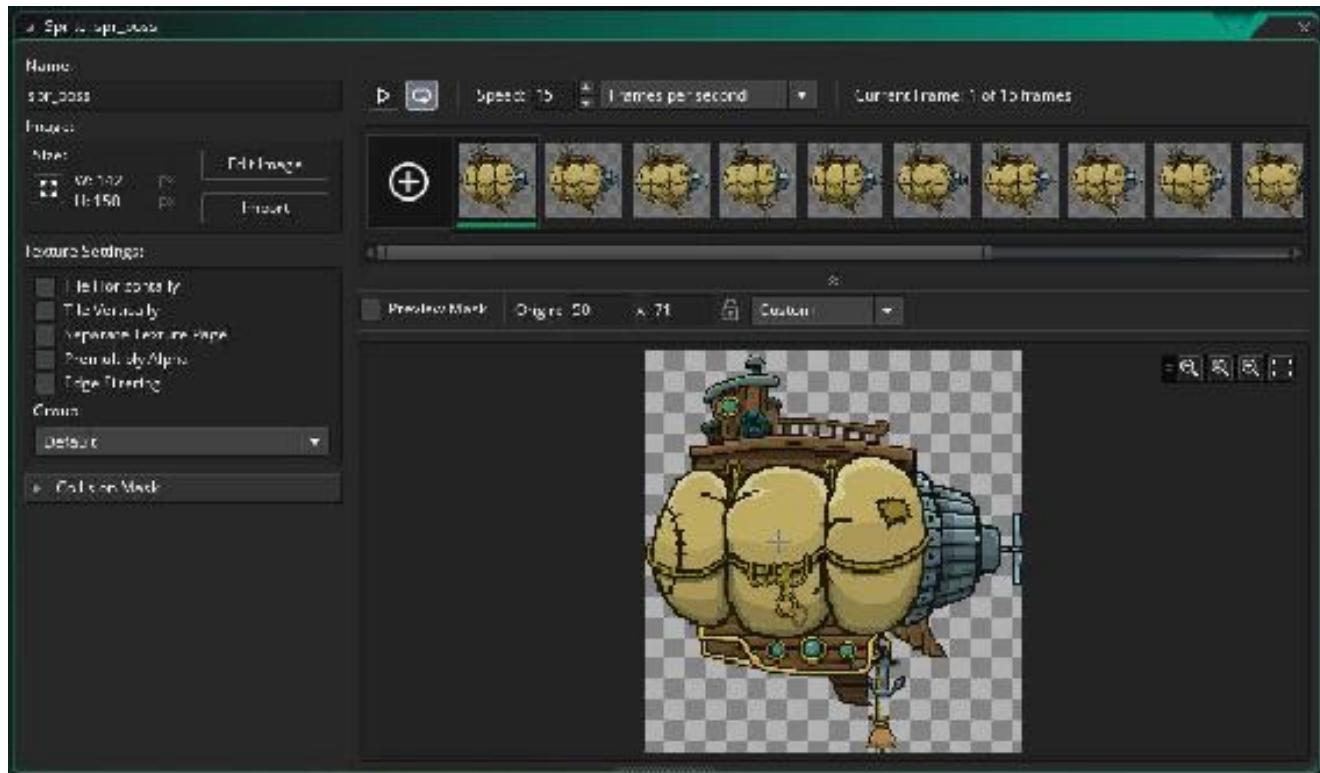


Figure 7_38: Showing the sprite for obj_boss

Create Event

```
/// @description set up
path_start(path_boss,4,path_action_stop,true);
instance_create_layer(x,y,"Enemy_Cannon",obj_cannon);
instance_create_layer(x,y,"Enemy_Cannon",obj_boss_hit);
start_hp=5*global.level*global.level;
hp=start_hp;
hit=false;
```

flash=false;

This sets the boss object to start moving on the defined path and create instances of obj_cannon and obj_boss_hit. It also sets up the initial hp based on the current games level. It will also set a flag for hit and flash, which will be used to graphically show damage.

Step Event

```
/// @description Check hp & for hit
if hp<0
{
    coin=instance_create_layer(x,y,"Player",obj_coin_bubble);
    coin.value=start_hp;
    with (obj_boss_hit) instance_destroy();
    with (obj_cannon) instance_destroy();
    instance_destroy();
    for (var i = 0; i < 10; i += 1)
    {
        part=instance_create_layer(x,y,"Score_Effect",obj_boss_parts);
        part.image_index=i;
    }
}
if hit==true
{
    hit=false;
    flash=true;
    alarm[0]=room_speed;
}
```

Checks the hp, creates a coin instance if it is dead, destroys itself and creates instances for the explosion effect. Sets flags needed to show damage.

Alarm 0 Event

```
/// @description Turn off flash  
flash=false;
```

Resets the flag for flash to false.

Draw Event

```
/// @description Drawing stuff  
scr_healthbar(-20,-70,80);  
  
if flash==true  
{  
    draw_sprite_ext(sprite_index,image_index,x,y,1,1,0,c  
    _red,1);  
}  
else  
{  
    draw_self();  
}
```

Draws a health bar, and then draws the assigned sprite with a red hue if to show damage, or draws normally.

Collision With obj_player_weapon_parent Event

```
/// @description destroy other  
with (other) instance_destroy();
```

Destroys the colliding instance – note no damage is recorded or acted upon.

Intersect Boundary Event

```
/// @description Change vert direction  
vspeed=vspeed*-1;
```

Changes the vertical speed to move up / down on collision with border.

Path Ended Event

```
/// @description Set moving  
vspeed=4;
```

When the boss initially reaches the end of the path, this code will make it start moving down at a speed of 4 pixels per step.

obj_boss_hit

This is the object instance that the player hit with their weapons to make damage to the boss enemy.

Note: In the object settings, the visible checkbox has been unchecked.

Figure 7_39 shows the sprite used for this instance.

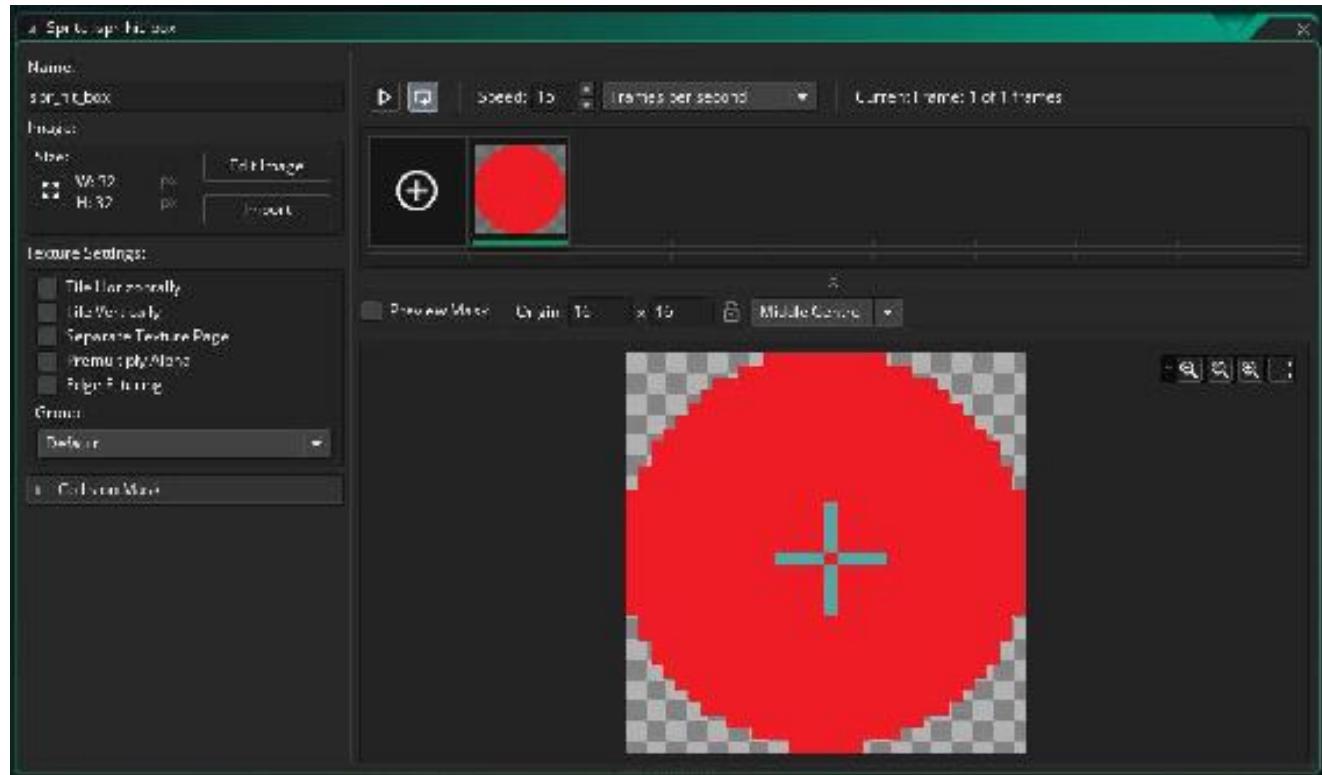


Figure 7_39: Showing sprite for obj_boss_hit

Step Event

```
/// @description Keep in sync with boss  
x=obj_boss.x+36;  
y=obj_boss.y+70;
```

This keeps it's location relative the boss, so when the boss moves, so does the target.

Collision With obj_player_weapon_parent Event

```
/// @description Make Damage  
obj_boss.hp-=5;  
obj_boss.hit=true;  
with (other) instance_destroy();
```

Reduces the boss's hp and set's flag to true so it shows damage, destroys colliding projectile.

Note: Addressing an object like this should only be done when you know that there is only one instance of the object you are referencing present in the game's room.

obj_canon

This object moves the boss enemy, and periodically fires a projectile towards the player.

Figure 7_40 shows the sprite for this object.

Note: It is upside down so when it rotates, for example by 180°, it points in correct direction and the has correct orientation.

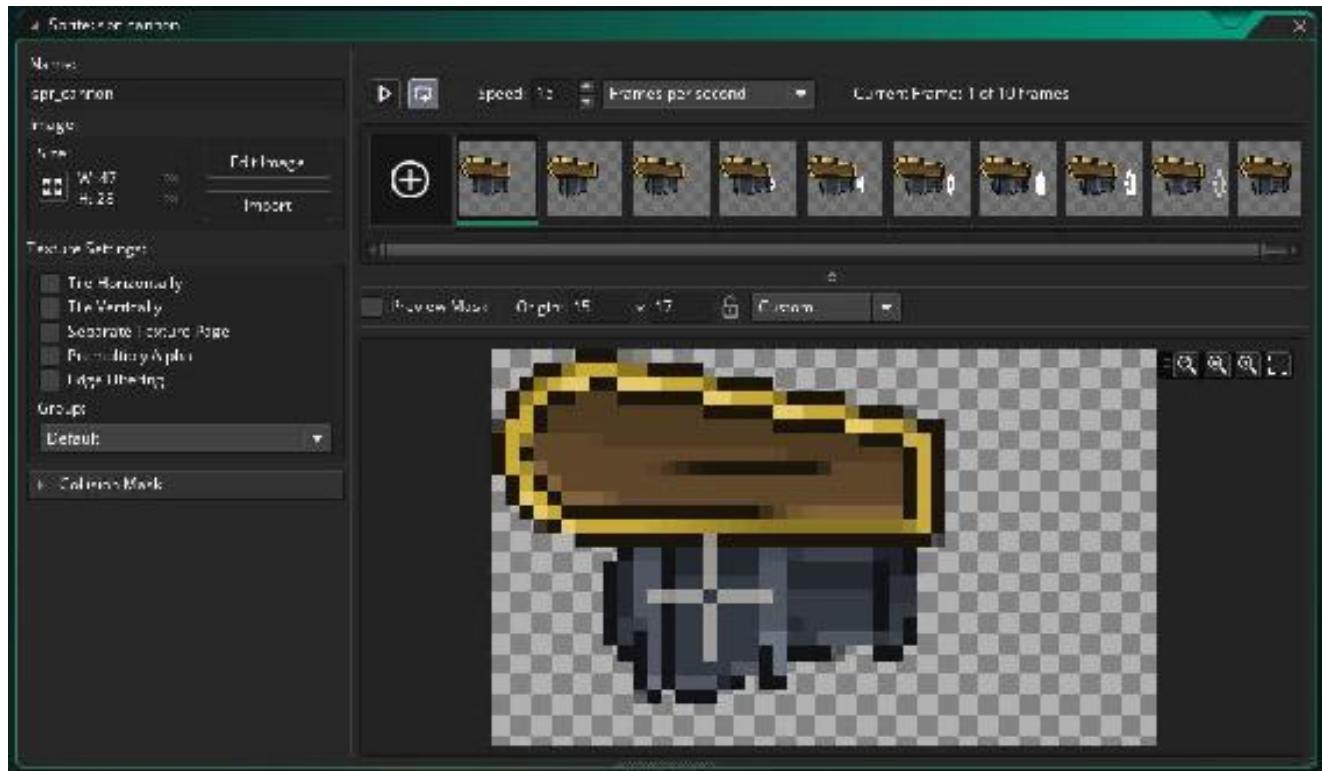


Figure 7_40: Cannon sprite assigned.

Create Event

```
/// @description Set Up  
image_speed=1;
```

Sets the initial speed that the frames will animate by.

Step Event

```
/// @description Point to player, fire projectile, sync  
with boss
```

```
image_angle=point_direction(x,y,obj_player.x,obj_player.y);
if image_index==5
{
    alarm[0]=room_speed*2;
    ball=instance_create_layer(x,y,"Enemy_Ball",obj_ball);
    ball.direction=image_angle;
    ball.speed=12;
}
x=obj_boss.x;
y=obj_boss.y;
```

image_angle=point_direction(x,y,obj_player.x,obj_player.y);

This line keeps it pointing at the player's location.

```
if image_index==5
{
    alarm[0]=room_speed*2;
    ball=instance_create_layer(x,y,"Enemy_Ball",obj_ball);
    ball.direction=image_angle;
    ball.speed=12;
}
```

This sets an alarm for two seconds, and fires a ball instance towards the player's current location.

```
x=obj_boss.x;
y=obj_boss.y;
```

This keeps the canon relative to the boss object.

Alarm 0 Event

```
/// @description Set Image Speed
image_speed=1;
```

Sets the image speed to 1, normal animation.

Animation End Event

```
/// @description Stop Anitmatting
image_speed=0;
```

Stops the sprite from animating when the final frame is reached, the alarm will make it animate again when triggered.

obj_ball

The previous object fires this projectile towards the player's current location.

Figure 7_41 shows the sprite for this object:

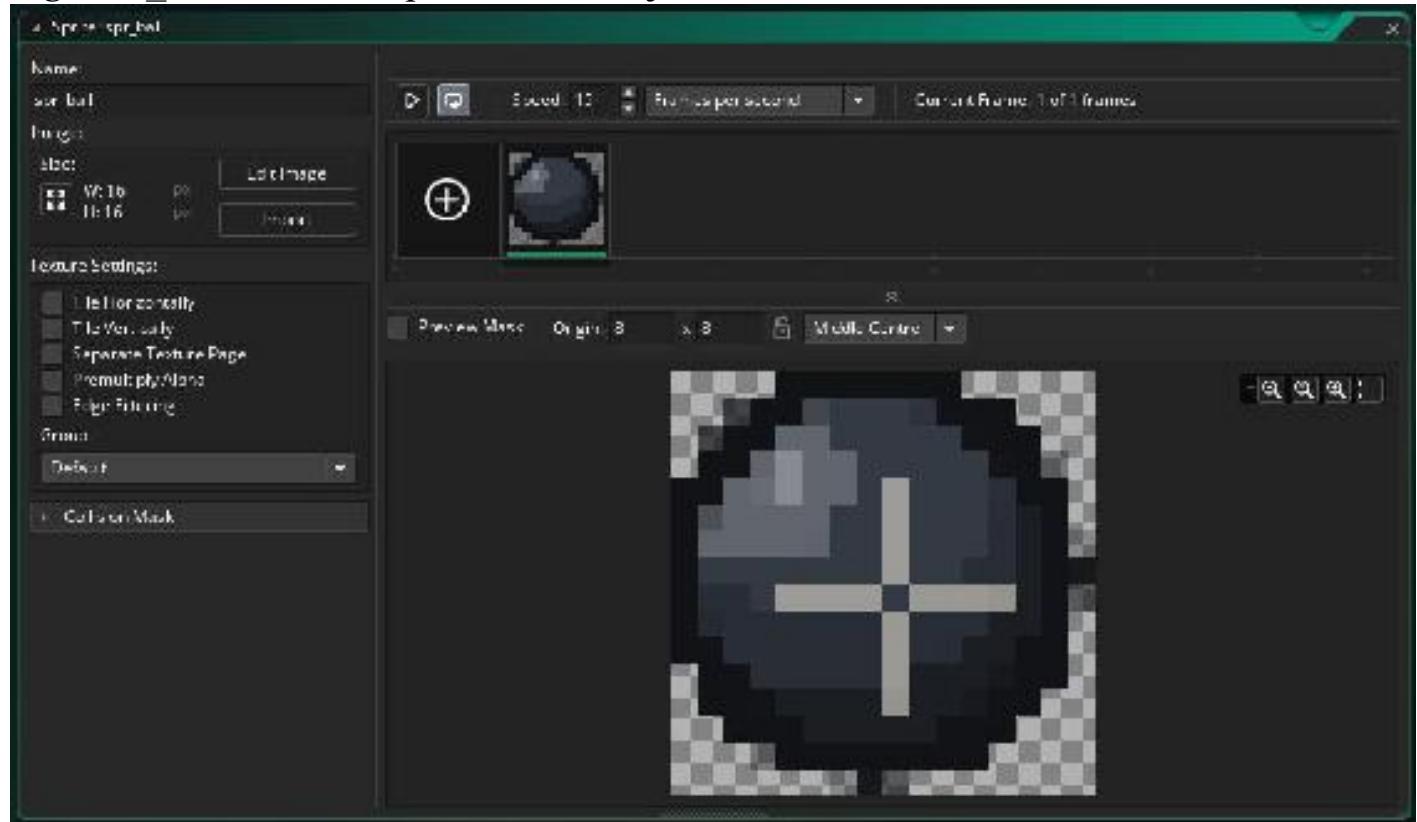


Figure 7_41: Showing sprite spr_ball

Collision With obj_mine_parent Event

```
/// @description Destroy  
instance_destroy();
```

Destroys self if collided a player's mine.

Outside Room Event

```
/// @description Destroy  
instance_destroy();
```

Prevents memory issues by destroying when done with.

obj_en_parent_projectile

There is no code or sprite for this object

obj_floor_parts

Figure 7_42 shows sprite for this object:

When the floor enemy has been destroyed, this object creates a cool explosion effect.

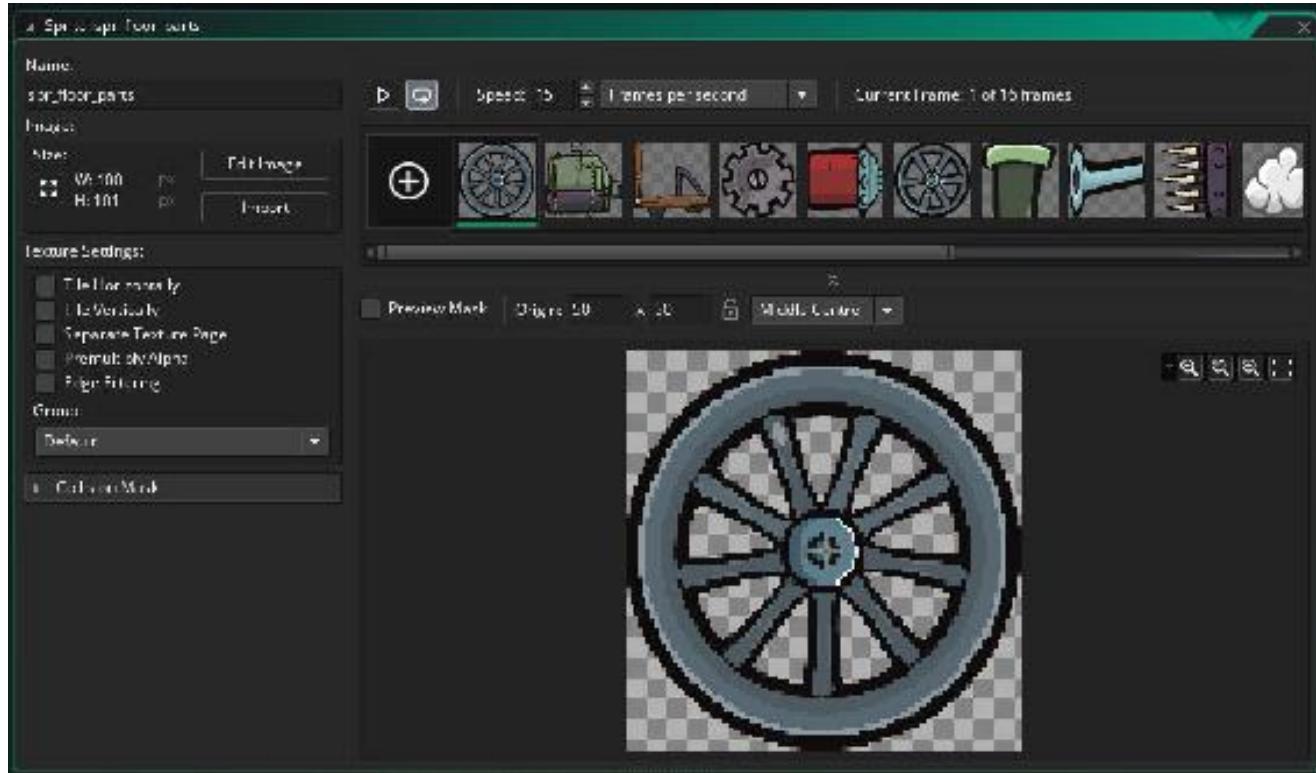


Figure 7_42: Sprite spr_floor_parts

Create Event

```
/// @description Set Moving  
image_speed=0;  
rot=choose(-4,4);  
direction=irandom(360);  
speed=4+irandom(4);  
size=0.2;  
alpha=1;
```

Sets the image speed to 0 to prevent animation. Sets to rotate clockwise or anti-clockwise, a random direction and a speed between 4 and 8. Sets the initial size as 20% and full visibility.

Step Event

```
/// @description Rotataate and change size and alpha
image_angle+=rot;
size+=0.01;
image_xscale=size;
image_yscale=size;
if size>1 alpha=0.02;
```

```
image_angle+=rot;
size+=0.01;
image_xscale=size;
image_yscale=size;
```

This code rotates the image and slowly increases the size.

```
if size>1 alpha=0.02;
```

Starts lowering the value of alpha when size is over 1.

Draw Event

```
/// @description Draw
image_alpha=alpha;
draw_self();
image_alpha=1;
```

Sets the alpha value, draws the image, then resets alpha so other object instances are not affected.

Outside Room Event

```
/// @description Destroy
instance_destroy();
```

Destroys when outside room.

obj_air_parts

When the air enemy has been destroyed, this object creates a cool explosion effect.

Figure 7_43 shows sprite set up for this object:

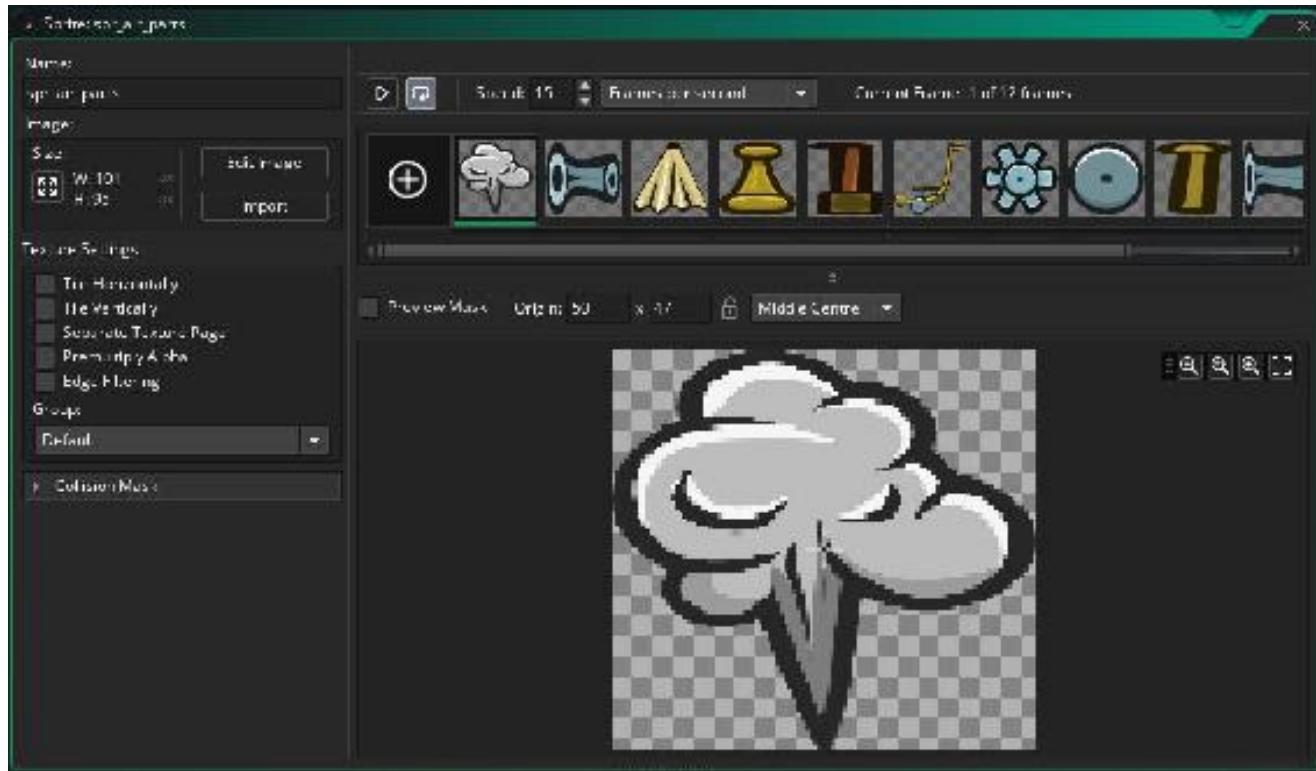


Figure 7_43: Showing the sprite set up for spr_air_parts

Code below is not explained, as it is the same as for **obj_floor_parts**.

Create Event

```
/// @description Set up
image_speed=0;
rot=choose(-4,4);
direction=irandom(360);
speed=4+irandom(4);
size=0.2;
alpha=1;
```

Step Event

```
/// @description Scale & set alpha
image_angle+=rot;
size+=0.01;
image_xscale=size;
image_yscale=size;
if size>1 alpha-=0.02;
```

Draw Event

```
/// @description Drawing Stuff
image_alpha=alpha;
draw_self();
image_alpha=1;
```

Outside Room Event

```
/// @description destroy
instance_destroy();
```

obj_boss_parts

When the boss enemy has been destroyed, this object creates a cool explosion effect.

Figure 7_44 shows sprite set up for this object:

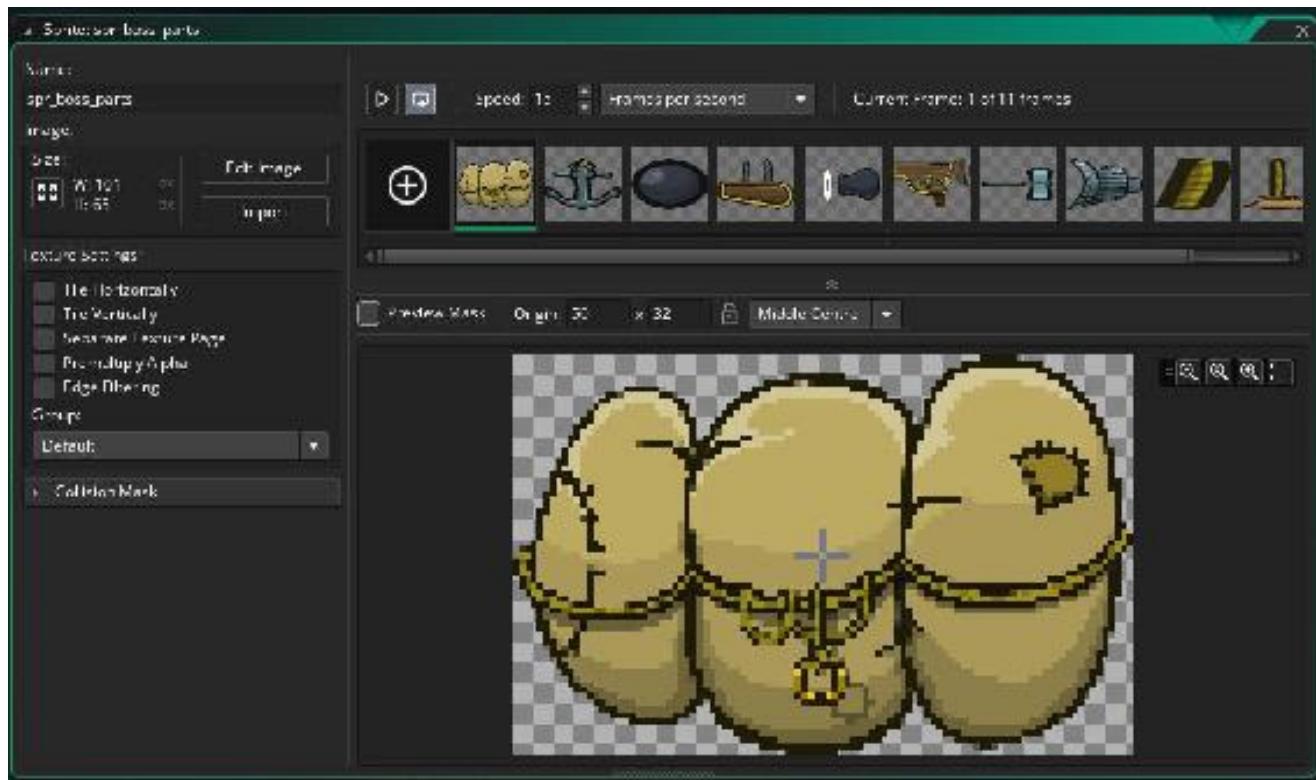


Figure 7_44: Showing the sprite set up for spr_boss_parts

Code below is not explained, as it is the same as for **obj_floor_parts**.

Create Event

```
/// @description Set up
image_speed=0;
rot=choose(-4,4);
direction=irandom(360);
speed=4+irandom(4);
size=0.2;
alpha=1;
```

Step Event

```
/// @description Update  
image_angle+=rot;  
size+=0.01;  
image_xscale=size;  
image_yscale=size;  
if size>1 alpha=0.02;
```

Draw Event

```
/// @description Drawing code  
image_alpha=alpha;  
draw_self();  
image_alpha=1;
```

Outside Room Event

```
/// @description Destroy  
instance_destroy();
```

obj_shield_collect

This is an object that moves across the window, and makes the player temporary invincible.

Figure 7_45 shows the sprite for this object:

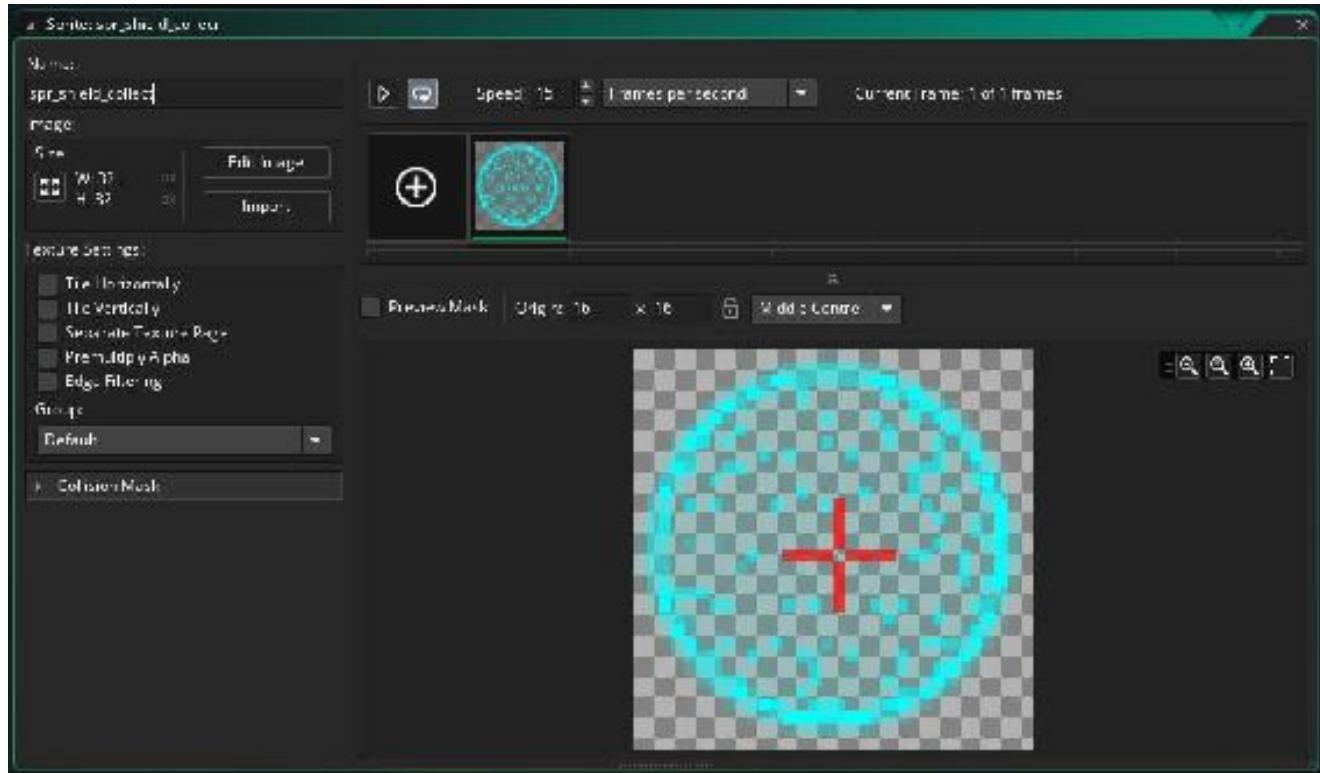


Figure 7_45: Sprite for obj_shield_collect

Create Event

```
/// @description Start Movin  
hspeed=-5;
```

Sets the instance moving to left at a speed of 5.

Step Event

```
/// @description destroy  
if x<0 instance_destroy();
```

Destroys self when off the left side of the window.

obj_power_up

Moves across the screen and increases the level and upgrades weapons, when collected by the player.

Figure 7_46 shows sprite **spr_power_up**:

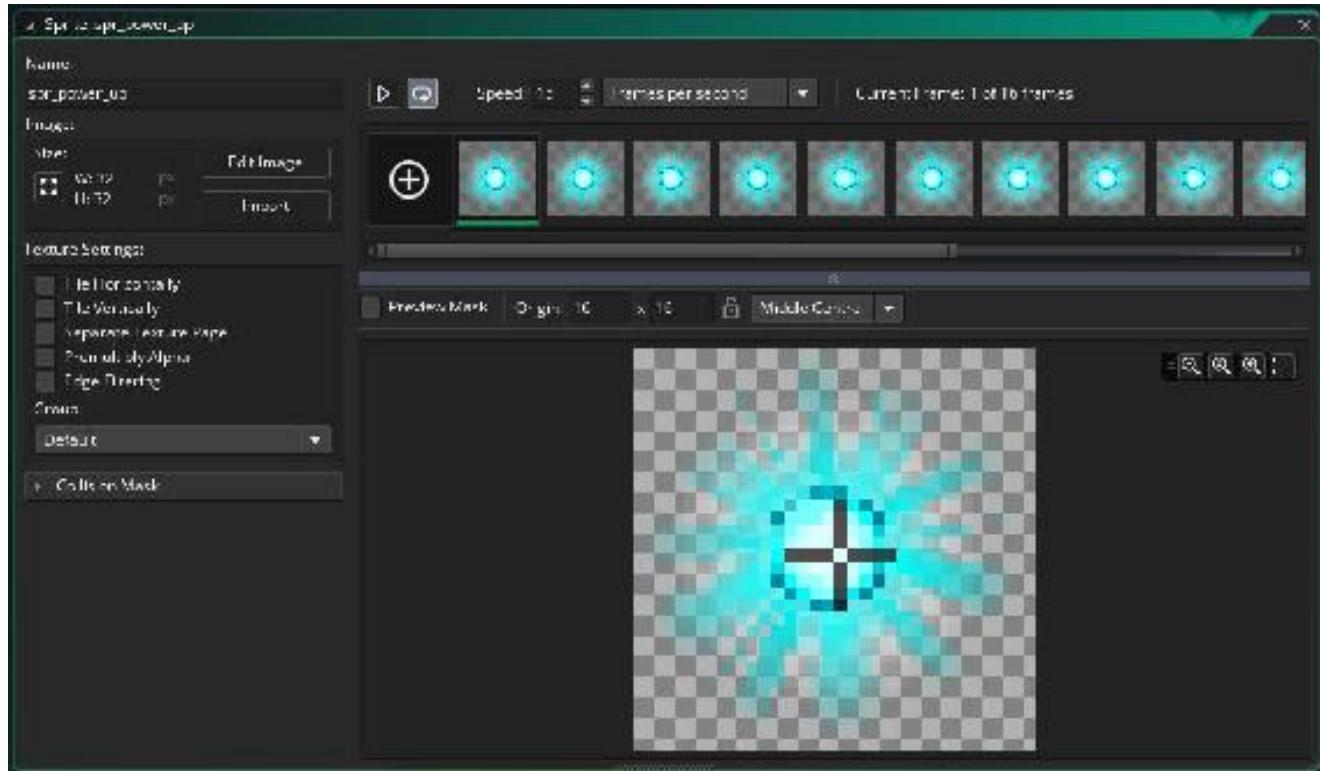


Figure 7_46: Showing *spr_power_up* used for *obj_power_up*
The code does the same as the previous object, so is not explained.

Create event

```
/// @description Start moving  
hspeed=-6;
```

Step Event

```
/// @description Destroy  
if x<0 instance_destroy();
```

obj_coin_bubble

Player can collect score points by collecting instances of this object.

Figure 7_47 shows the sprite set up for this object:

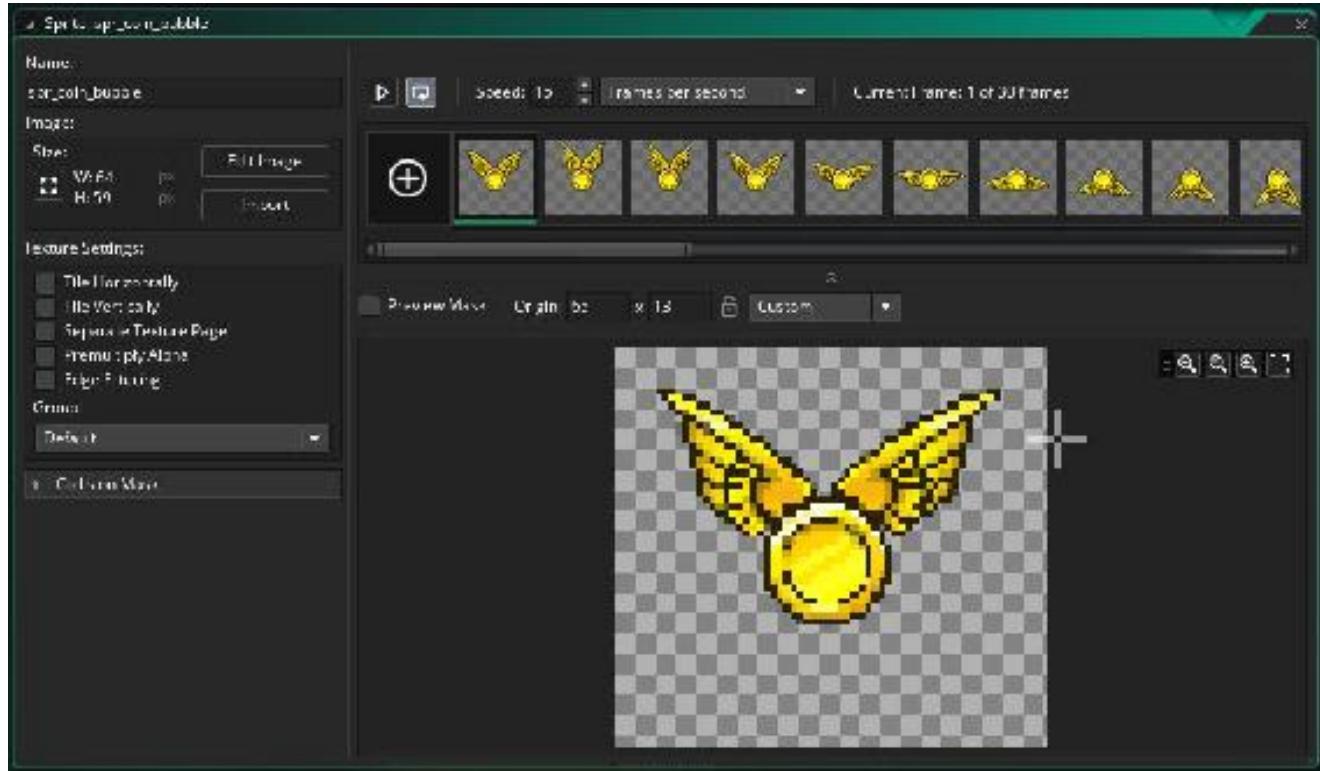


Figure 7_47: Showing the sprite spr_coin_bubble

Create Event

```
/// @description Move towards player location  
motion_set(point_direction(x,y,obj_player.x,obj_player.y),5);
```

This code sets the instance to move towards the player's current location at a speed of 5.

Collision With obj_player Event

```
/// @description Start Score Effect  
audio_play_sound(snd_coin,1,false);  
ins=instance_create_layer(x,y,"Score_Effect",obj_to_score);
```

```
ins.value=value;  
instance_destroy();
```

When it collides with the player, it plays a sound and makes an instance of obj_to_score and gives it a value value. It then destroys itself.

Outside Room Event

```
/// @description Destroy  
instance_destroy();
```

Destroys if not collected by the player, once off the left side of the window.

obj_to_score

Increases the score once near the drawn score on the HUD.

There is no sprite for this instance.

Create Event

```
/// @description Move towards score on HUD  
dir=point_direction(x,y,200,40);  
motion_set(dir,4);
```

Moves towards point 200,40 (which is the point on the window where the score is drawn).

Step Event

```
/// @description Check location  
if distance_to_point(200,40)<8  
{  
    instance_destroy();  
    score+=value;  
}
```

When close to the score location, destroys self and increases the score.

Draw Event

```
/// @description Draw Value  
draw_text(x,y,value);
```

Draws it's value.

obj_asteroid_large

A large asteroid that the player can shoot at. If shot it creates smaller asteroids.
Damages player if player hits it.

The sprite for this is shown in *Figure 7_48* :

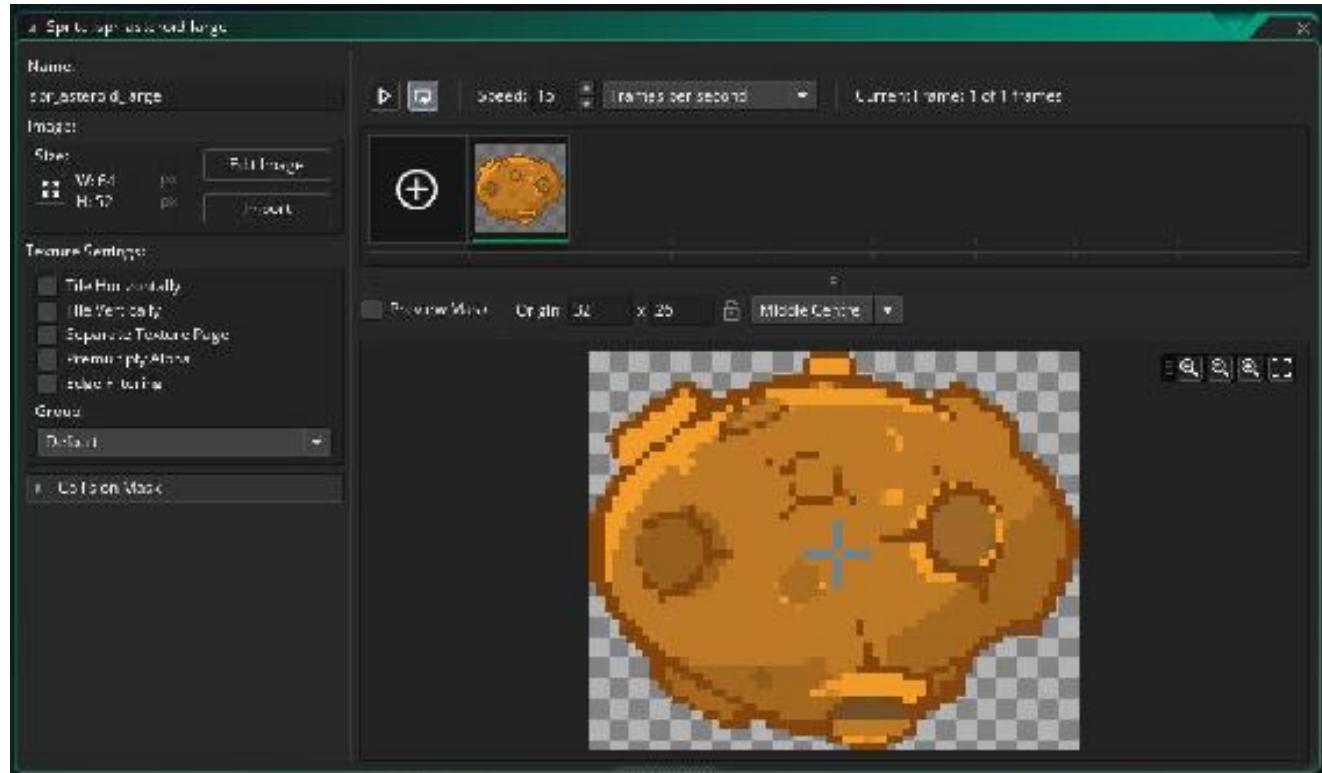


Figure 7_48: Showing sprite spr_asteroid_large

Create Event

```
/// @description Set Movement
ang=irandom_range(150,210);
x=room_width+200;
y=irandom_range(100,room_height-100);
motion_set(ang,4);
rot=choose(-1,1);
```

Sets a random angle between 150 and 210 (30° each way from 180°), sets the start location 200 pixels off of the right side of the window, and a random y position between 100 and room height less 100. Sets moving at a speed of 4, and rotation clockwise or anti-clockwise.

Step Event

```
/// @description Rotate  
image_angle+=rot;
```

Rotates each step.

Collision With obj_player Event

```
/// @description Damage player if no shield  
if other.shield>0  
{  
    coin=instance_create_layer(x,y,"Player",obj_coin_bubble);  
    coin.value=60;  
    instance_destroy();  
}  
else  
{  
    audio_play_sound(snd_lose_health,1,false);  
    health-=3;  
    other.flash=true;  
    other.alarm[0]=room_speed/2;  
    instance_destroy();  
}
```

When it collides with player, checks for shield. If shield is active it makes a coin bubble instance and destroys itself. If the shield is not active the player loses health, shows damage and the asteroid is destroyed.

Collison With obj_player_weapon_parent Event

```
/// @description Damage Control  
ast=instance_create_layer(x,y,"Asteroid",obj_asteroid_medium);  
ast.direction=direction-20;  
ast.speed=speed+1;  
ast.rot=rot*-2;
```

```
ast=instance_create_layer(x,y,"Asteroid",obj_asteroid_
medium);
ast.direction=direction+20;
ast.speed=speed+1;
ast.rot=rot*-2;
with other instance_destroy();
instance_destroy();
```

If hit by a player's primary weapons, two new smaller asteroids are created, the projectile is destroyed and the current asteroid is destroyed.

obj_asteroid_medium

A medium asteroid that the player can shoot at. If shot it creates smaller asteroids.
Damages player if player hits it.

Figure 7_49 shows the sprite set up for this object.

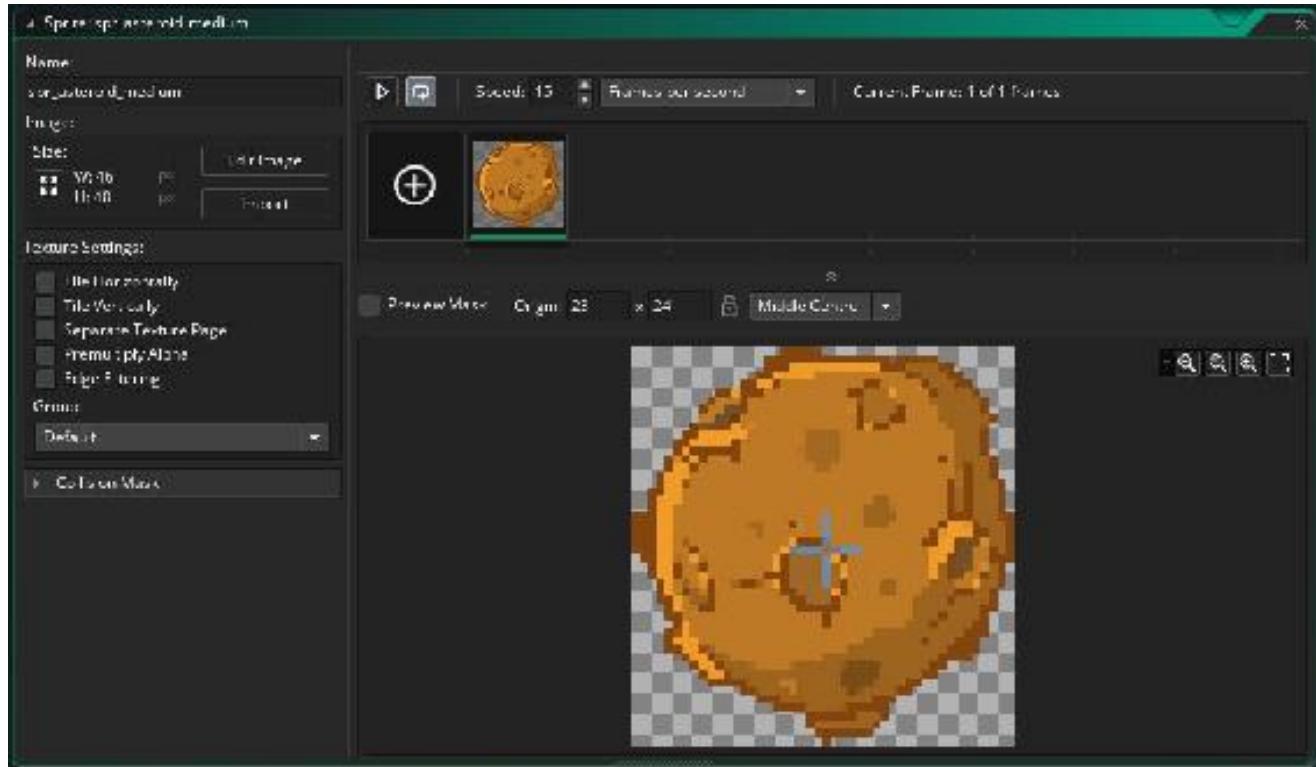


Figure 7_49: Showing spr_asteroid_medium

Step Event

```
/// @description rotate  
image_angle+=rot;
```

Collision With obj_player_weapon_parent Event

```
/// @description Damage Control  
ast=instance_create_layer(x,y,"Asteroid",obj_asteroid_  
small);  
ast.direction=direction-20;  
ast.speed=speed+1;  
ast.rot=rot*-2;
```

```
ast=instance_create_layer(x,y,"Asteroid",obj_asteroid_
small);
ast.direction=direction+20;
ast.speed=speed+1;
ast.rot=rot*-2;
with other instance_destroy();
instance_destroy();
```

Creates 2 new small asteroids, and destroys projectile and self.

Collision With obj_player Event

```
/// @description Damage Player If no Shield
if other.shield>0
{
    coin=instance_create_layer(x,y,"Player",obj_coin_bubble);
    coin.value=30;
    instance_destroy();
}
else
{
    audio_play_sound(snd_lose_health,1,false);
    health=3;
    other.flash=true;
    other.alarm[0]=room_speed/2;
    instance_destroy();
}
```

Checks for player's shield and act accordingly.

obj_asteroid_small

A small asteroid that the player can shoot at. If shot it creates a coin bonus instance for the player to collect. Damages player if player hits it

Figure 7_50 shows the sprite for this object:



Figure 7_50: Showing sprite set up for spr_asteroid_small

Create Event

```
/// @description Set Up  
image_speed=0;  
image_index=irandom(5);
```

Prevents animation and chooses a random sub image.

Step Event

```
/// @description Rotate  
image_angle+=rot;
```

Rotates in the given direction (when created by **obj_asteroid_medium**)

Collision With obj_player Event

```
/// @description Damage Player if no shield
```

```

if other.shield>0
{
    coin=instance_create_layer(x,y,"Player",obj_coin_bubble);
    coin.value=15;
    instance_destroy();
}
else
{
    health-=1;
    other.flash=true;
    other.alarm[0]=room_speed/2;
    audio_play_sound(snd_lose_health,1,false);
    instance_destroy();
}

```

Creates a coin or damages player, depending if the player's shield is active or not.

Collision With obj_player_weapon_parent Event

```

/// @description Make Coin
coin=instance_create_layer(x,y,"Player",obj_coin_bubble);
coin.value=20;
with other instance_destroy();
instance_destroy();

```

Creates a coin instance if hit by player's primary weapons.

obj_spawn_poles

Creates foreground instances to create a better parallax effect.

Create Event

```
/// @description Set an Alarm  
alarm[0]=(room_speed*10)+irandom(room_speed*10);
```

Alarm 0 Event

```
/// @description Spawn obj_pole  
alarm[0]=(room_speed*10)+irandom(room_speed*10);  
instance_create_layer(room_width+300,room_height/2,  
"Foreground",obj_pole);
```

obj_pole

A foreground object that moves across the window, and moves up and down in relation to the player's movement.

Figure 7_51 shows the sprite set up for this object:

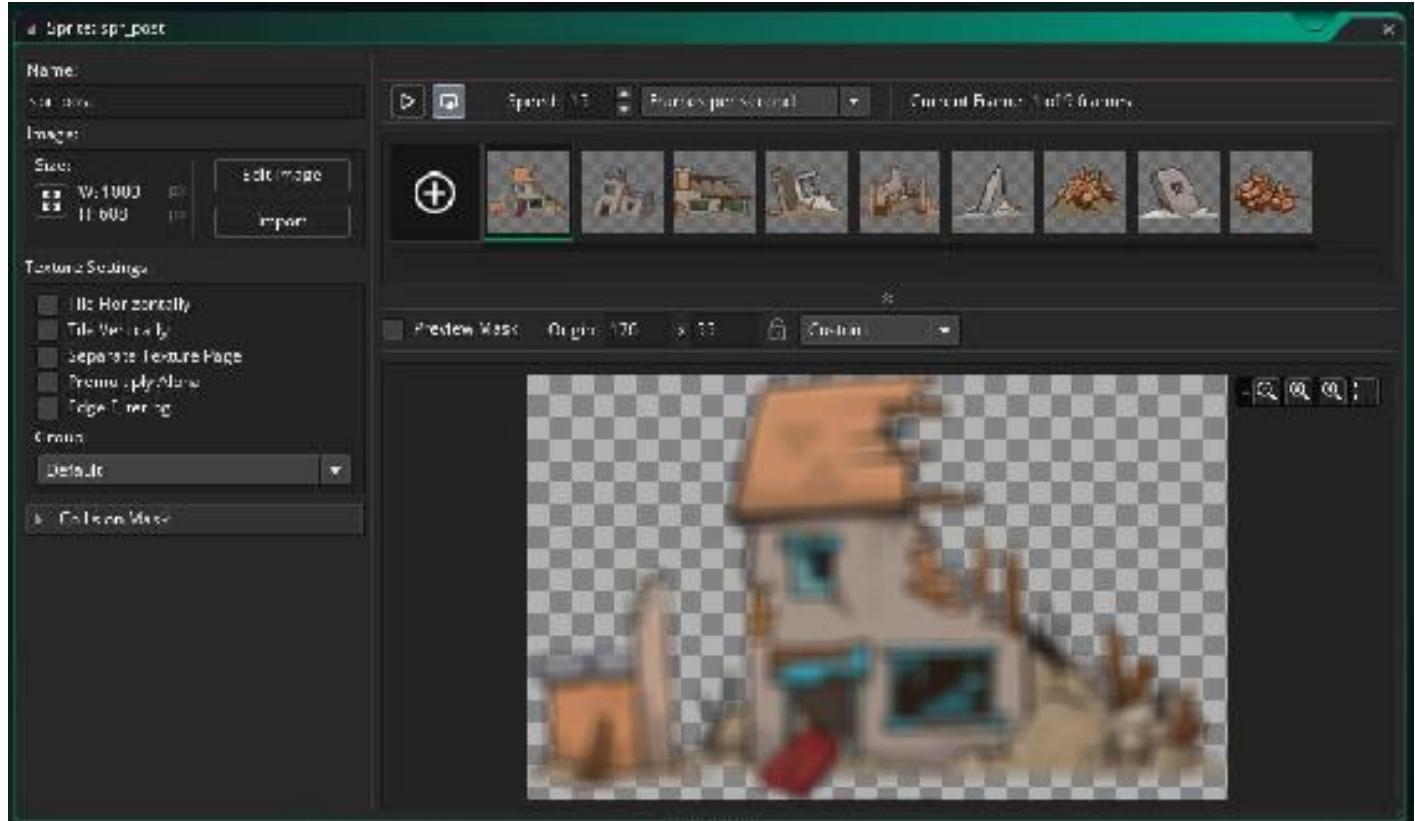


Figure 7_51: Showing sprite set up for spr_post

Create Event

```
/// @description Set Up
hspeed=-6;
image_speed=0;
image_index=irandom(8);
```

Moves left at a speed of 6. Prevents animation and chooses a random sub_image.

Step Event

```
/// @description Check & change position
if x<-1000 instance_destroy();
player=obj_player.y
y=(room_height/2)-player+550;
```

Moves up and down, taking into account the player's y position.

obj_info

Tells the player what the bonus they collected does.

Figure 7_52 shows the sprites and set up for this object:

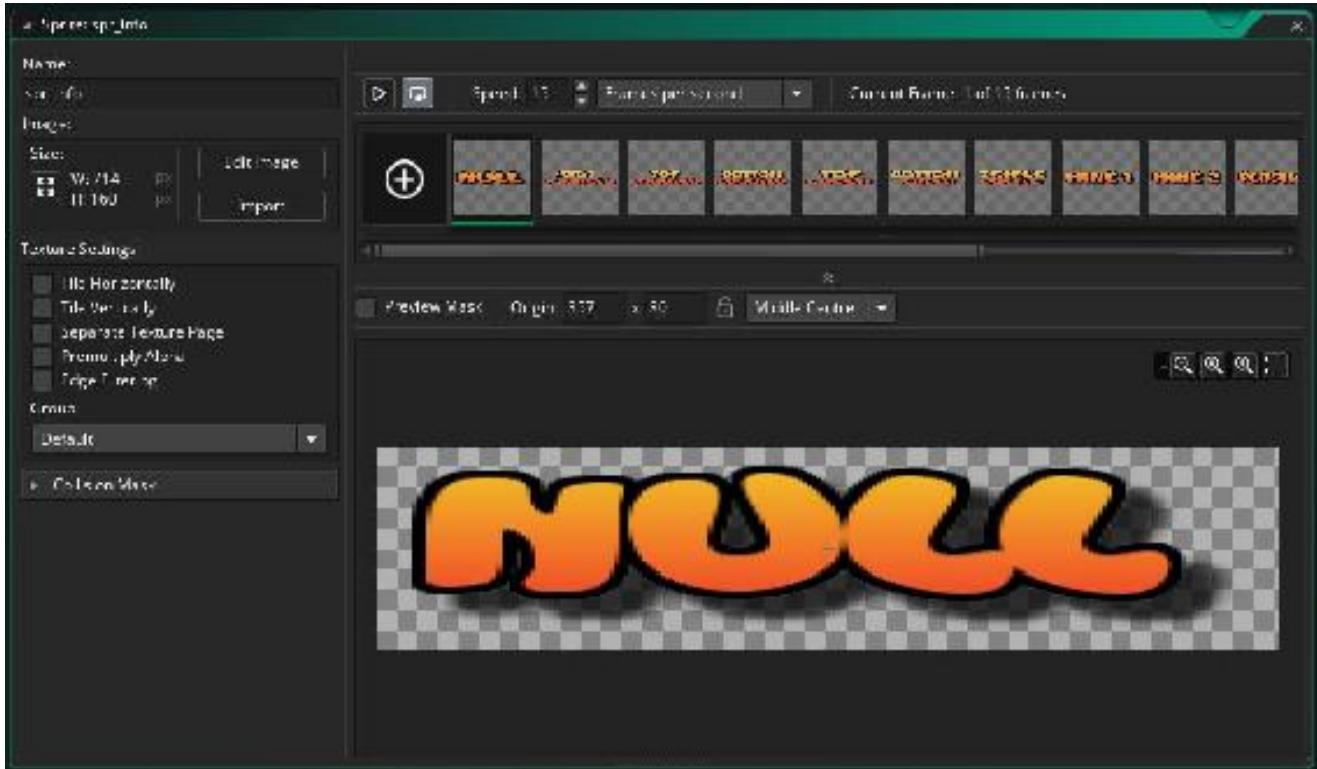


Figure 7_52: Showing spr_info

Create Event

```
/// @description Set Up  
image_speed=0;  
alarm[0]=room_speed*5;  
x=room_width/2;  
y=room_height/2;
```

Prevent automatic animation, set an alarm for 5 seconds, and position in middle of the window.

Alarm 0 Event

```
/// @description Destroy  
instance_destroy();
```

Destroys self when alarm triggers.

obj_cursor

Follows the player around the room, used for one of the player's weapons. Also draws the locked-on sprite on the nearest enemy (if upgraded and an enemy instance exists).

There is no sprite assigned to this object, though it does use a few. This is shown in *Figure 7_53* :

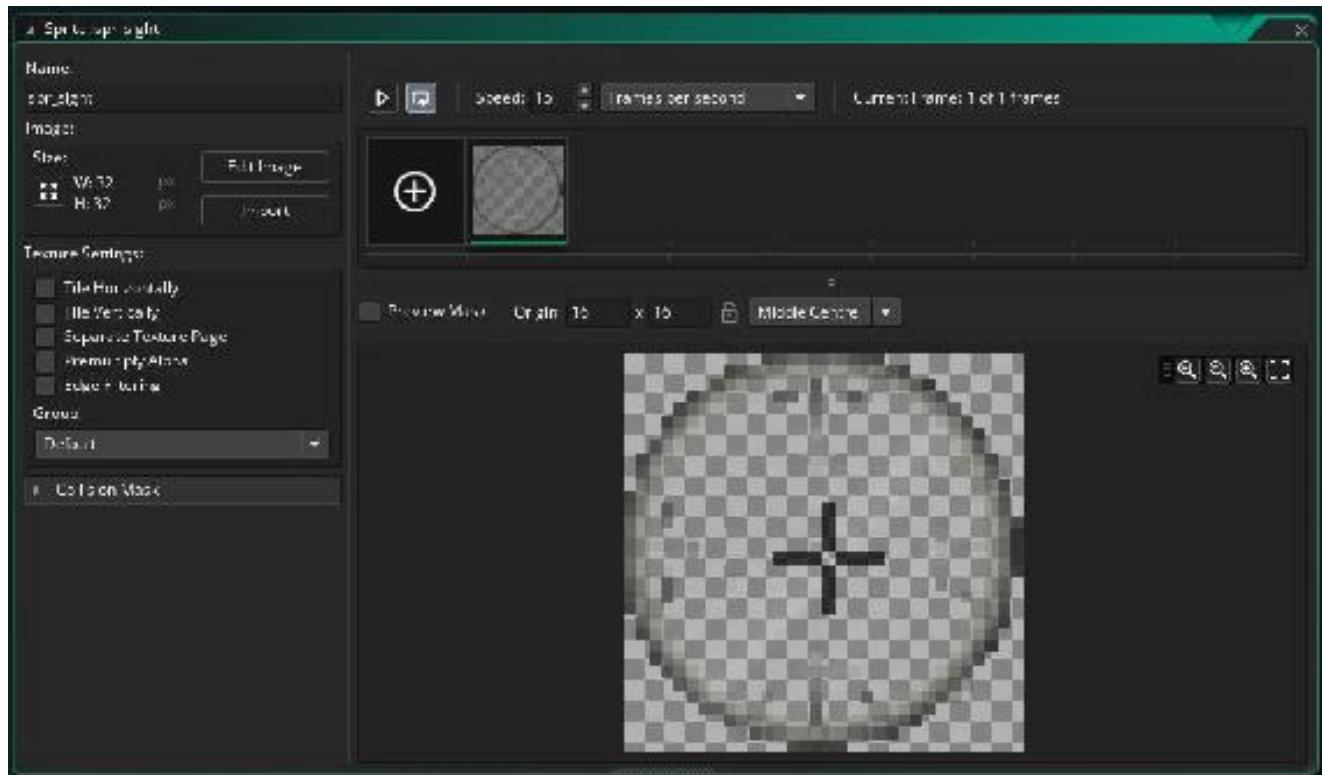


Figure 7_53: Showing sprite spr_sight

Figure 7_54 shows sprite spr_lock

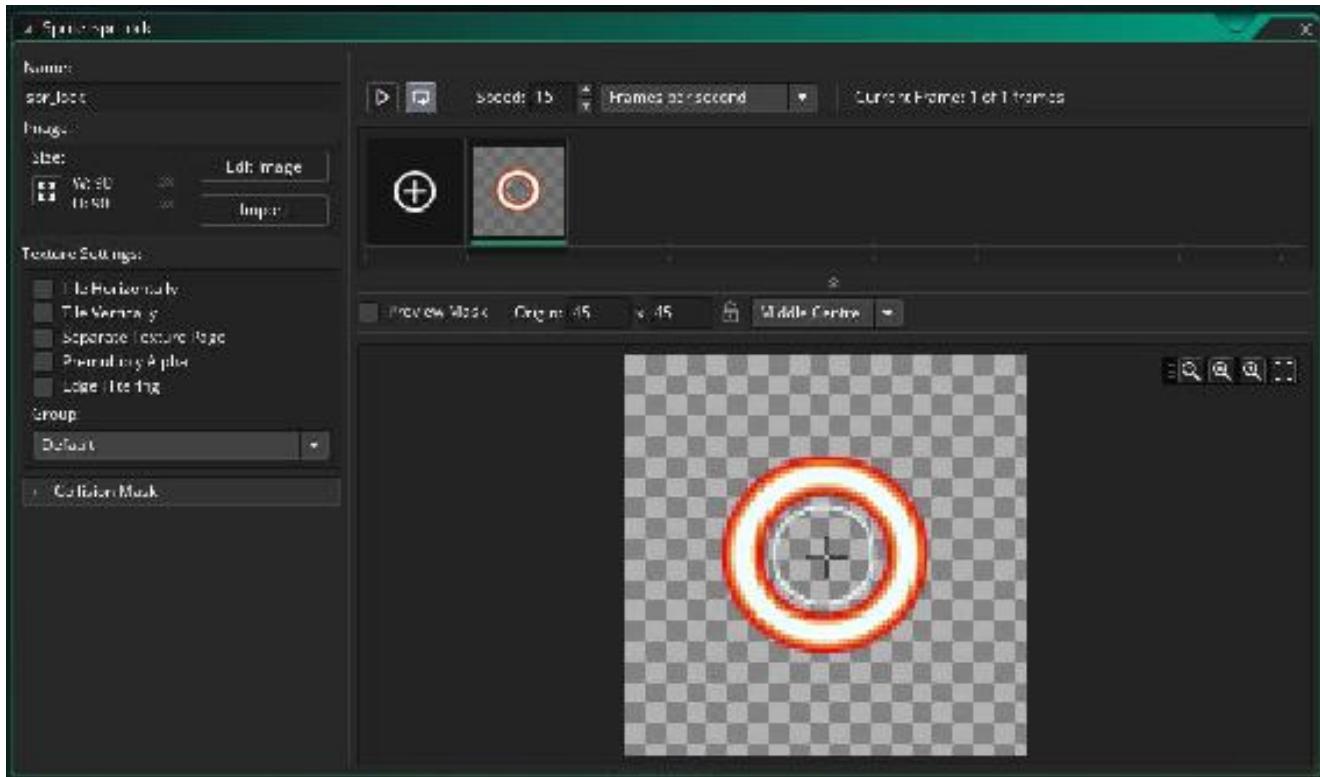


Figure 7_54: Showing sprite spr_lock

Create Event

```
/// @description Set Default Cursor  
cursor_sprite=spr_sight;
```

Tells the program to set the cursor to sprite spr_sight.

Step Event

```
/// @description Update Position  
x=mouse_x;  
y=mouse_y;
```

Updates the location to the current mouse's position.

Draw Event

```
/// @description Lock On  
if instance_exists(obj_enemy_parent) and  
global.level>12  
{
```

```
nearest=instance_nearest(x,y,obj_enemy_parent);
draw_sprite(spr_lock,0,nearest.x,nearest.y);
}
```

Will draw the lock on sprite if player's level is over 12, at the nearest enemy location.

obj_spawner

This will spawn shield and upgrade bonus instances.

There is no sprite for this object.

Create Event

```
/// @description Set Alarms  
alarm[0]=room_speed;//shield  
alarm[1]=room_speed;//power power up
```

Sets some initial alarms.

Alarm 0 Event

```
/// @description Create a Shield Bonus  
instance_create_layer(room_width+200,  
random_range(100,room_height-  
100),"Player",obj_shield_collect);  
alarm[0]=room_speed*40;
```

Creates a shield for the player to try and collect, restarts the alarm.

Alarm 1 Event

```
/// @description Create Power Up  
instance_create_layer(room_width+200,  
random_range(100,room_height-  
100),"Player",obj_power_up);  
alarm[1]=room_speed*35;
```

Creates a power up for the player to try and collect, restarts the alarm.

obj_asteroid_spawn

Periodically spawns asteroids.

There is no sprite for this object.

Create Event

```
/// @description Set an Alarm  
alarm[0]=room_speed*6;
```

Sets the initial alarm.

Alarm 0 Event

```
/// @description Make asteroid and restart alarm  
alarm[0]=room_speed*6;  
instance_create_layer(x,y,"Asteroid",obj_asteroid_larg  
e);
```

Restarts the alarm, creates an asteroid object.

obj_hud_control

Draws a background images, and info needed for the player (is drawn semi-transparent).

There is no sprite assigned for this object, though it does use one, which is shown below in *Figure 7_55* :

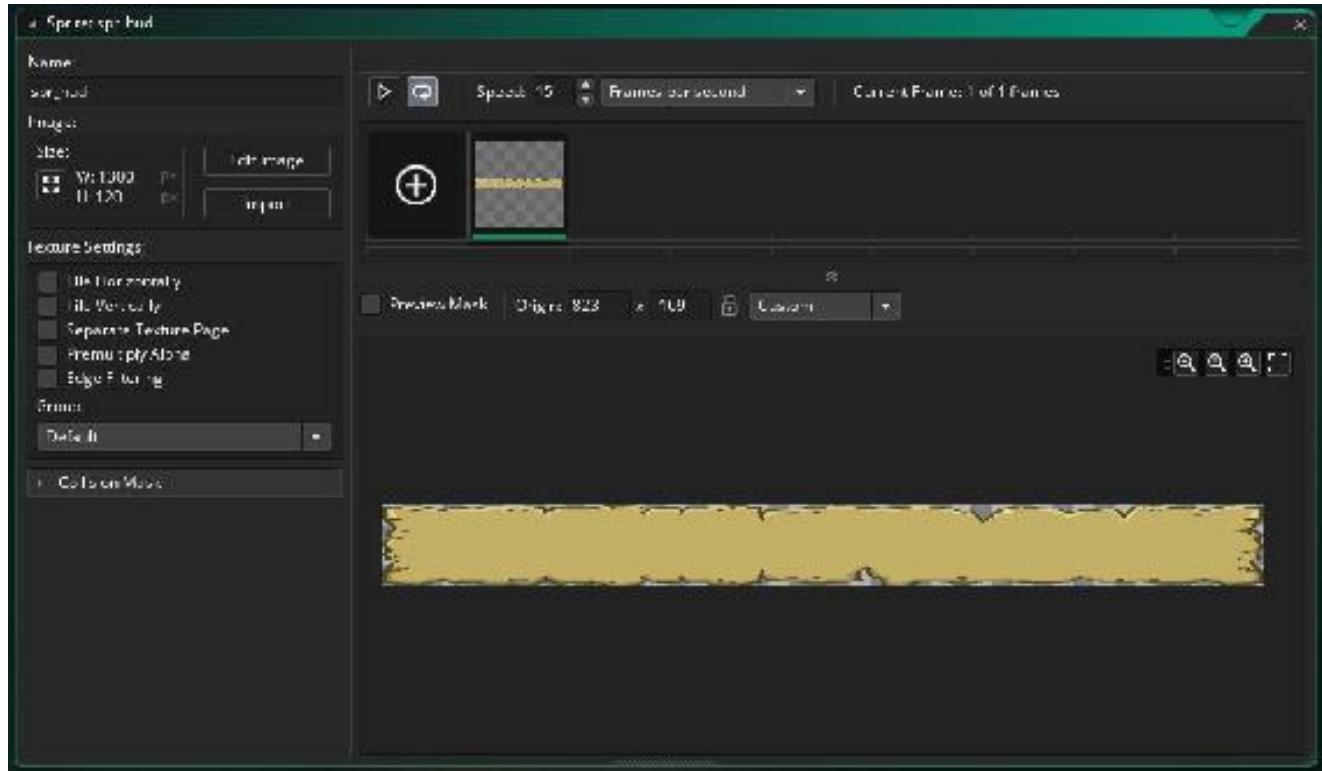


Figure 7_55: Showing sprite spr_hud

Create Event

```
/// @description Stop & Play Music  
audio_stop_sound(snd_music_menu);  
audio_play_sound(snd_music_game,1,true);
```

Stops any music and player the music for the game room.

Step Event

```
/// @description Format Score & Save on Gameover If  
New Highscore  
score_text=string(score);
```

```

while string_length(score_text)<6
score_text="0"+score_text;

hiscore_text=string(global.highscore);
while string_length(hiscore_text)<6
hiscore_text="0"+hiscore_text;

//check health;
if health<0
{
if score>global.highscore//do this if bigger than current
score
{
ini_open("scores.ini");//open ini
ini_write_real("scores","high",score);//write a value
ini_close();//close value
global.highscore=score;//update highscore
global.saved=true;
}
room_goto(room_gameover);
}

```

This formats the text for the score and highscore. It checks the players health (a global value) and if the player is dead it updates the saved highscore. It then goes to the gameover room.

Draw Event

```

/// @description Draw HUD
draw_set_font(font_menu);

```

```
draw_set_colour(c_black);
draw_set_halign(fa_center);
draw_set_valign(fa_middle);
draw_set_alpha(0.5);
draw_sprite(spr_hud,0,0,0);
draw_text(200,40,"Score "+string(score));
draw_text(200,80,"Highscore
"+string(global.highscore));

draw_text(600,40,"Level "+string(global.level));
draw_text(600,80,"Power "+string(global.pow));
```

```
if obj_player.shield>0
{
    draw_text(1000,40,"Shield Active");
}
else
{
    draw_text(1000,40,"No Shield");
}
```

```
draw_text(1000,80,"Health "+string(health));
```

```
draw_set_alpha(1);
```

Sets style for text drawing and sets the alpha to 50%. Draws HUD image all HUD text and then sets alpha back to normal value.

This Section Is For The Game Over Room

obj_game_over

Will display player's score and highscore, then goto the starting room.

There is no sprite for this object.

Create Event

```
/// @description Set alarm and change music
alarm[0]=room_speed*6;
audio_stop_all();
audio_play_sound(snd_music_gameover,1,true);
```

Sets an alarm to 6 seconds, stops music and plays music for gameover room on loop.

Alarm 0 Event

```
/// @description Goto Game Start room
room_goto(room_splash);
```

Goes back to menu room when alarm is triggered.

Draw Event

```
/// @description Draw Score Info
draw_set_colour(c_white);
draw_text(room_width/2,400,"Score "+string(score));
draw_text(room_width/2,500,"Highscore
"+string(global.highscore));
```

Sets drawing style and draws player's score and highscore.

Rooms

This game has a few rooms, the setups are shown below

room_splash

This room has a size of 1300 wide and a height of 768.

It has a single instance of **obj_splash**.

room_menu

This room has a size of 1300x768.

It has a number of layers as shown in *Figure 7_56* :

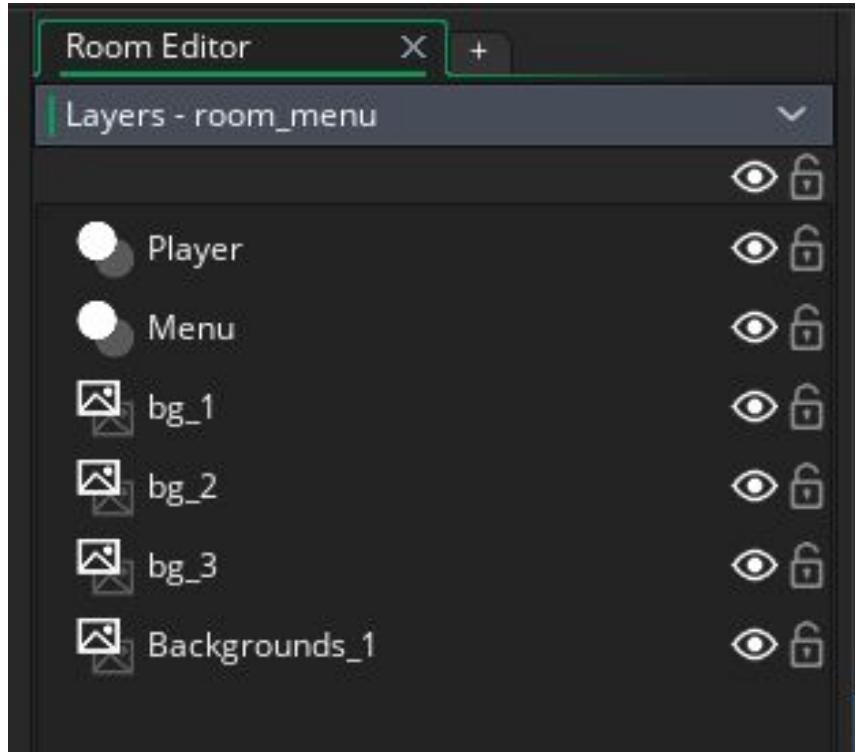


Figure 7_56: Showing the layers for room_menu

The **Player** layer has one instance of **obj_player_menu** placed on it.

The **Menu** layer has one of **obj_menu** and one of **obj_button**.

Figure 7_57 shows these instances placed in the room:



Figure 7_57: Showing instances placed in the room

This room also has 3 backgrounds, used for a parallax effect.

Figure 7_58 shows the setting for **bg_1**:

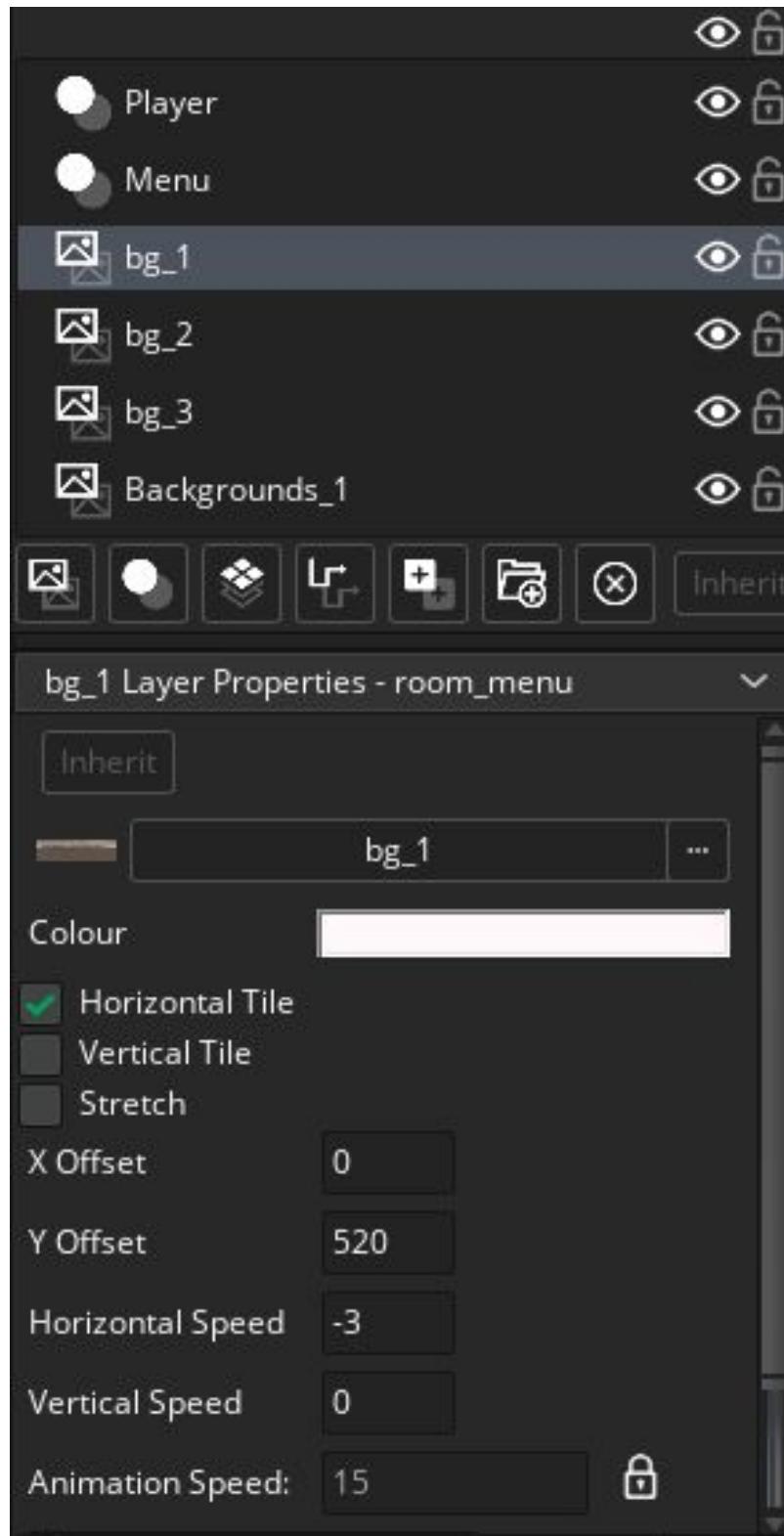


Figure 7_58: Showing background setting for bg_1

Background **bg_2** is set with a y offset of -100 and horizontal speed of -2.

Background **bg_3** is set with a y offset of 200 and horizontal speed of -1.

Figure 7_59 shows the room with backgrounds and instances in place:



Figure 7_59: Showing room set up

room_intro

This room has a size of 1300 x 768.

It has one instance of **obj_intro_control** and one of **obj_emitter**.

room.HowPlay

This room has a size of 1300 x 768.

It has one instance of **obj_how_to_play** and one of **obj_emitter**.

room_game

This room has a dimension of 1300 by 768. It has many layers, as shown in *Figure 7_60*:

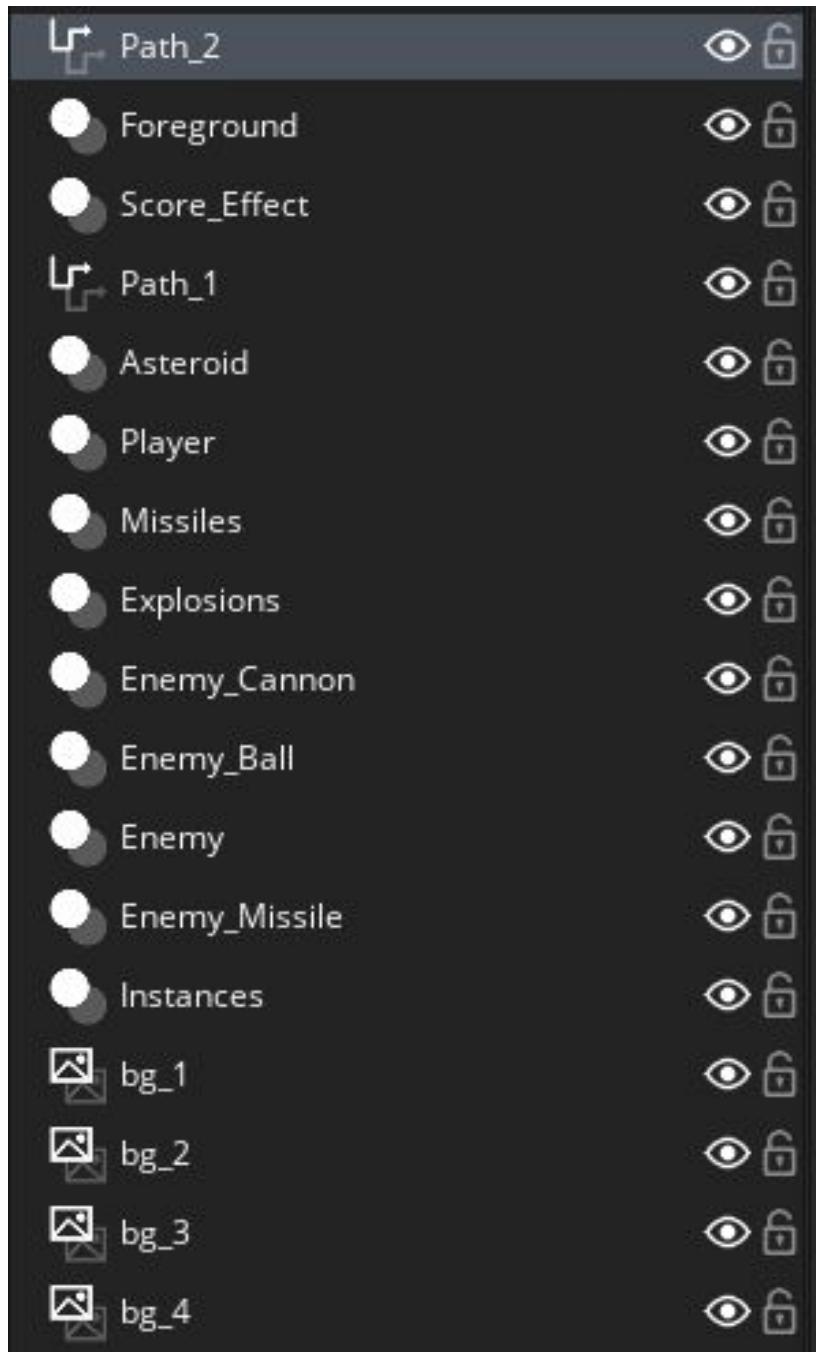


Figure 7_60: Showing layers for room_game

The **Foreground** layer has:

- **obj_spawn_poles**
- **obj_cursor**

The **Score_Effect** layer has no instances present at room start.

The **Asteroid** layer has:

- **obj_asteroid_spawn**

The **Player** layer has:

- **obj_player**
- **obj_hud_control**
- **obj_spawner**

The **Missile** layer has instances when the room starts.

The **Enemy_Cannon** layer has no instances.

The **Enemy_Ball** layer has no instances.

The **Enemy** layer has no instances.

The **Enemy_Missile** layer has no instances.

The **Instances** layer has no instances.

Backgrounds are set up in the same way as the menu room.

The **Foreground** layer has:

- **obj_spawn_poles**
- **obj_cursor**

The **Foreground** layer has:

- **obj_spawn_poles**
- **obj_cursor**

room_gameover

This room has a size of 1300 x 768.

It has one instance of **obj_game_over**.

Paths

This game uses two paths, as shown in *Figure 7_61* and *Figure 7_62*:

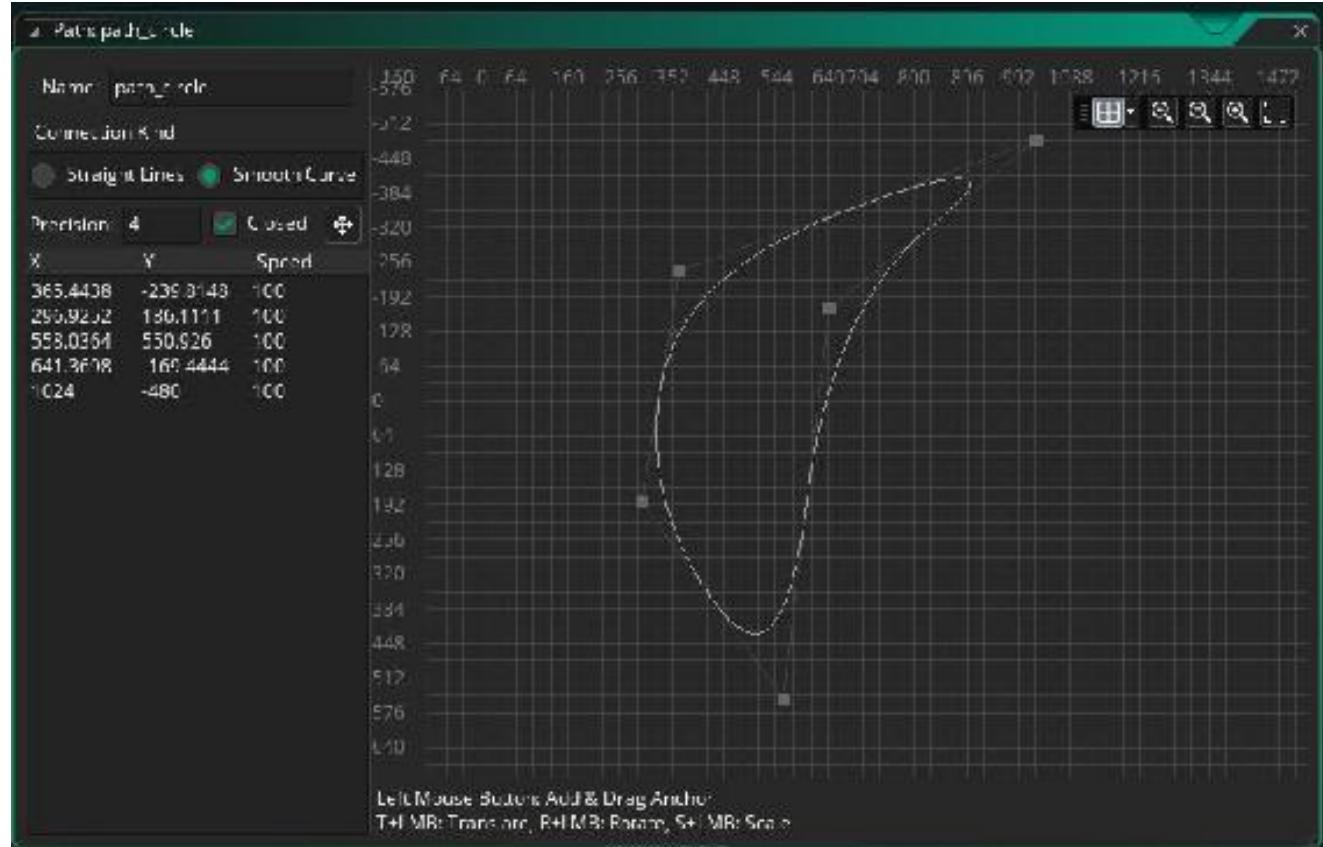


Figure 7_61: Showing path set up for path_circle

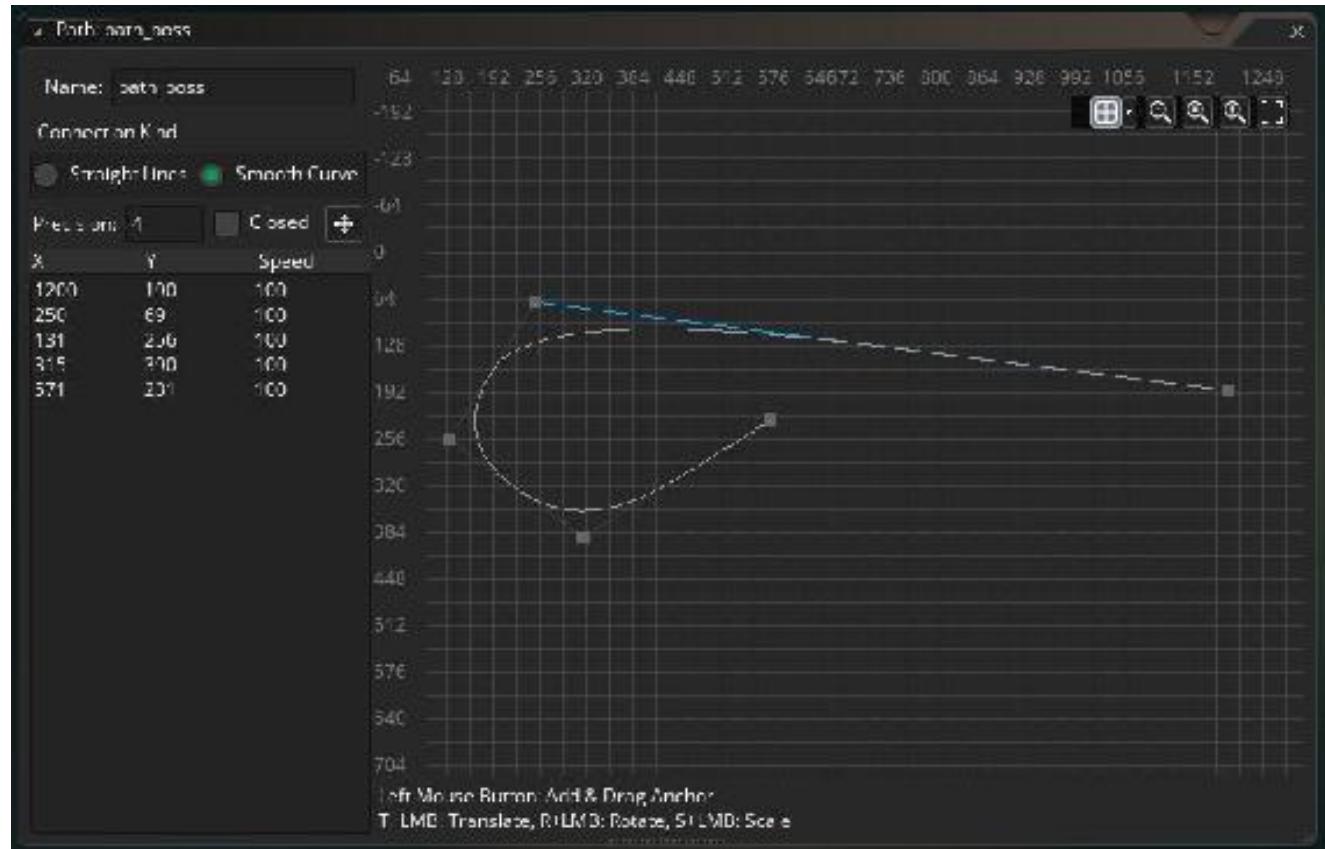


Figure 7_62: Showing path set up for path_boss

Audio

This game uses sound audio, all from [SoundImage.org](#). *Figure 7_63* shows the sounds used in this game.



Figure 7_63: Showing sounds

There is a playable exe version of this game in the download.

There is also a skeleton project with all the code and instances set up, but with no sprites.

All sprites for the game are from:

[GameDeveloperStudio.com](#)

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.

Chapter 10 Summary

Target Audience

When designing your game and acquiring assets you should keep in mind your target audience. You should consider such things as:

- Age range
- Will they play on PC, tablet or phone?
- What genre are they expecting?
- If aimed at a younger audience, who will you make revenue?

Pricing

OK, so you've made a game (maybe your first). Earning a bit of cash would be nice. There are a few ways to make revenue from your production. You can list your game in an online Appstore, such as Google Play. You could charge for in-game purchases – such as coins or weapon upgrades. Alternatively you can tie it in with your Patreon account and provide exclusive levels or upgrades every month.

Working as a Team

One of the best things about indie game development is the friendliness and willingness to help. Although most indie's work alone, some prefer to work in small teams. This is great if you each have certain skills – for example: programming, artwork, music, game promotion. A team project can be great if everyone plays their role and communicates well, allowing the projects to progress quickly.

That said, it doesn't always go as planned. You may find that some team members have no experience working as part of a team, which can cause issues. All teams really require someone to take on the management role and keep the wheels in motion – a skill that does take some time and effort to learn.

Some indie's do it for the fun, some for the (hopefully) money, and some as stepping-stone that may lead onto greater things. Working as part of a team (or managing one) can be very rewarding and look great on a resume.

Before accepting a role in a team project, I recommend spending some time researching the project leader or other team members. I have seen a number of requests for team members from people who have no prior knowledge to game development,

programming or other expertise in managing the project. I say hats off to them for trying – but I see such projects doomed to failure.

Useful Links

The internet is a big place, but here are some sites I frequently frequent:

forum.YoYoGames.com

A great community for asking and answering GameMaker related questions and issues.

GameDeveloperStudio.com

My favourite graphical assets site. Premade sprites, in high quality and fairly priced. Several assets are free, so a great place to visit if you have a low budget.

SoundImage.org

A huge collection of free music and many sound effects (credit asked for). If you are looking for something usable for your game, this is a great place to start.

GraphicRiver.net

A massive collection of premade sprites. Cheap for personal use, commercial use is a bit more expensive – but if you have the funds it's a great place to buy from.

CreativeMarket.com

Another great site for purchase premade graphics.

OpenGameArt.org

Huge selection of artwork, mostly on CC or CC-BY. Great if you are working with a low or zero budget.

Google

Have a quick coding question? Need some example GML code? A well-formed search query will bring up some useful results.

KickStarter.com

A crowd funding site for raising funds for your project.

IndieGoGo.com

Another crowd funding site for raising funds for your project.

Crediting Creators

If you have ever made your own assets, you will appreciate how much time and effort goes into making them. As such you should also credit authors in the way they dictate, whether that would be just a name or website link. Some creators provide multiple licence options, maybe access for non-commercial use for \$20, or \$150 for single commercial use. Please buy the correct licence as a way of thanking the creator. Some assets (such as fonts) are usually free for private use, and perhaps a donation for commercial use.

Consider providing a free copy of your game (or key) so creators can see their assets in use. Always check for usage terms for any assets you use, if in doubt contact the creator and ask.

The above is not legal advice, I'm just pointing out that asset use generally has some terms attached for what you can use it for.

Educational Use

When I first got the bug for development (around the age of 8), there were few options available with regards to software and hardware for making games. Times have changed since then. GameMaker Studio 2 allows for quickly making mini game projects – without a full understanding of programming but has the scope and power that allow commercial quality games to be made. How I wish something like GameMaker Studio existed when I started.

GameMaker Studio 2 is ideally suited for educational use. The software itself has an educational licence option, a huge number of online tutorials and videos, along with some great books, such as the one that you are reading now ??.

It's great for younger students, who require instant satisfaction. With GameMaker Studio 2, within 5 minutes you can have an object moving around a screen, making noises and responding player input. As such over the course of a term students can start off with something very basic, and progressively be improved over a number of weeks. There are a few books available for teaching GML in the classroom, such as another of my books, **Beginning GameMaker Studio 2**.

For educators looking to teach GameMaker Studio 2 and its GML language, the learning curve is quite small. If you have previous knowledge of C# or similar languages, the transition is not so tough.

Like most programming languages, GML requires some understanding of maths to make the most out of it – therefore not making GML suitable for younger learners. Fortunately, GameMaker Studio 2 has a Drag & Drop system that allows creation of games with little or no code. This allows younger users to make basic games, and learn the logic needed for when they progress onto GML coding.

Where Next

OK, so you now know the basics. Where next?

I have some other books out that teach further GML and game creation. The ones that may interest you are:

Beginning GameMaker Studio

Covers the basic of GML, suitable for educational use.

100 Programming Challenges

100 challenges to push you GML skills to their limit. Shows the basic code you will need and leaves it to you to make a working example, solutions to the challenges in GMZ format.

Practical GameMaker Projects

Covers the coding of a number of games.

Programming is just one part of making a game. Have a go at trying to learn to use software for the creation of graphics and audio. If you intend make a career out of game making, this are great skills to learn when you decide to full time indie or seek employment in a studio.

Maths is integral to making games. Perhaps consider taking an evening class, or home study course in basic or more advanced – you'll be amazed how much of what you learn can be applied to game design and programming.

The website, **forum.yoyogames.com** holds regular game jams. These jams challenge you to make a game on a theme within the time period (usually 3 days). This is a great way to force you to actually finish a game (*I have lots of unfinished projects*). It's a great way to see what other devs are making and to share feedback. There usually a number of community prizes, either for winning or just taking part. Certainly worth checking out.

Conclusion

Hopefully you will now have a better idea of how to approach your own game project and considerations to consider along your journey.

I had a great time writing this book, and I hope that you had enjoyment from reading it.

If you ever have any questions or issues about GML programming or GameMaker Studio in general, I suggest you try out the forums at forum.yoyogames.com. It a great and friendly place with lots of skilled GMS2 users willing to help you out – providing you can demonstrate that you have put some effort in and not expecting someone else to do all the work for you.

If you find any issues of problems with this book (such as omissions or mistakes) please drop me an email to: **Ben@LearnGameMakerStudio.com**

The appendix following covers some basic GML which you are more than likely to use within your game endeavours.

Appendix

This appendix gives a brief introduction to programming with GameMaker Studio 2's GML language.

It is suggested to use this for reference, or study after completing Chapter 7, or at least the programming introduction of Chapter 7. The contents in this appendix comes from my sister book, **Beginning GameMaker Studio 2**, which includes additional content, and material for educational use, and is available on Amazon.

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
- Determining 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`. In front 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*:

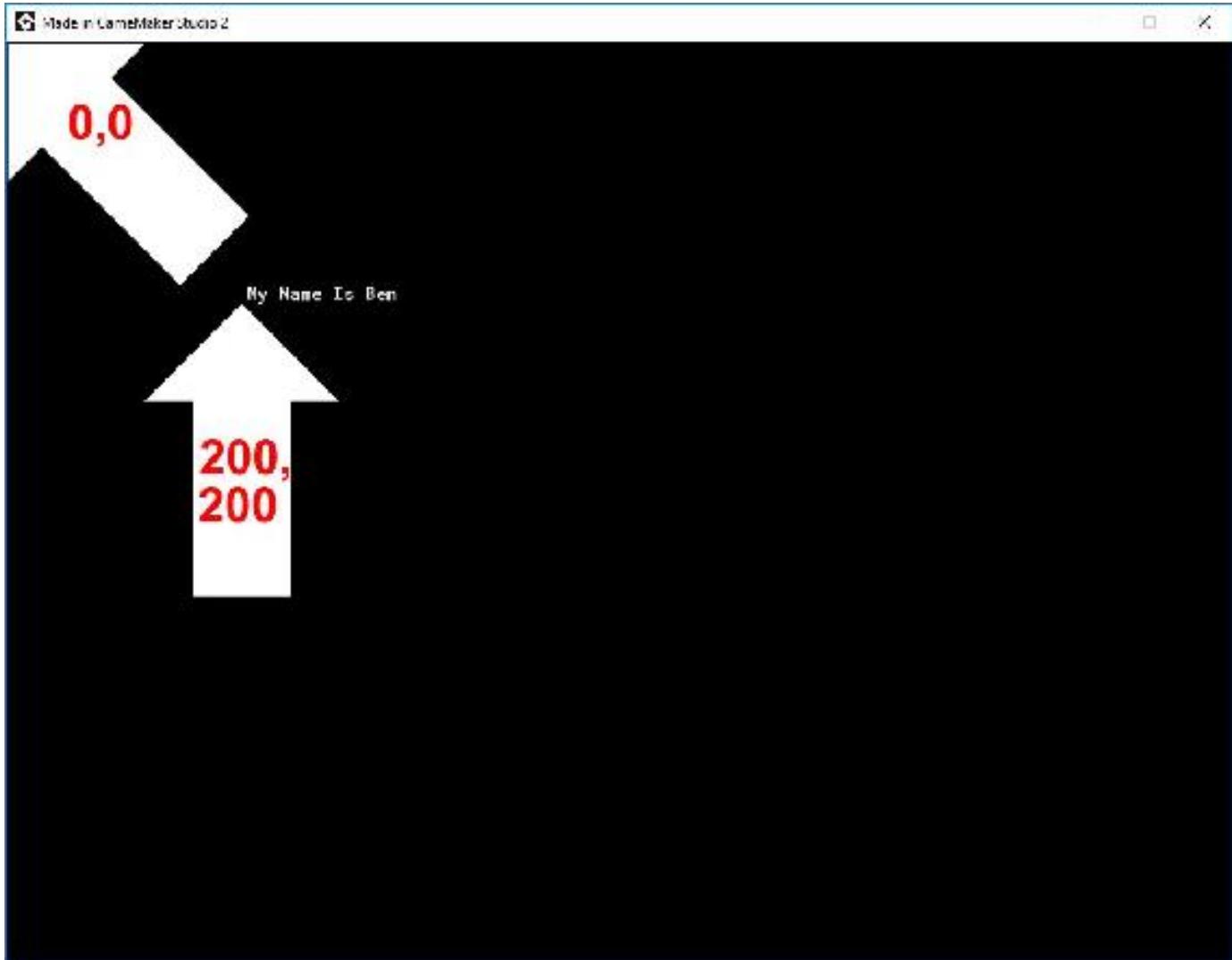


Figure A_1_1. Showing various locations in a room

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 top. Drawing can be relative to the room position or a view. This and the next section assume drawing in a standard room using default room settings without the use of views. See *Figure 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.

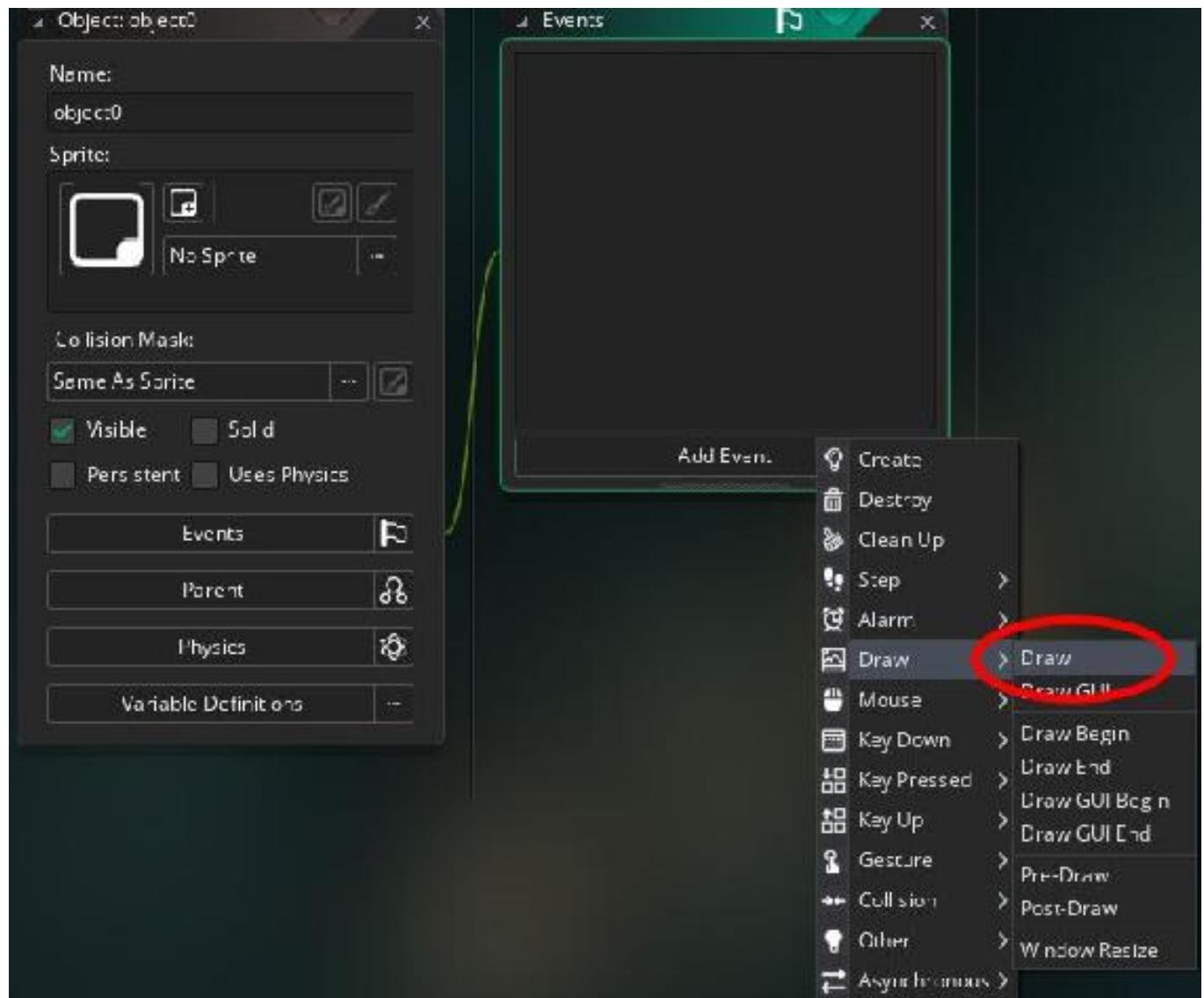


Figure A_3_1: Showing how to select draw event

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 color:

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_lonestrip);
draw_vertex(50,50);
draw_vertex(150,50);
draw_vertex(50,150);
draw_vertex(250,50);
draw_vertex(50,250);
draw_primitive_end();
```

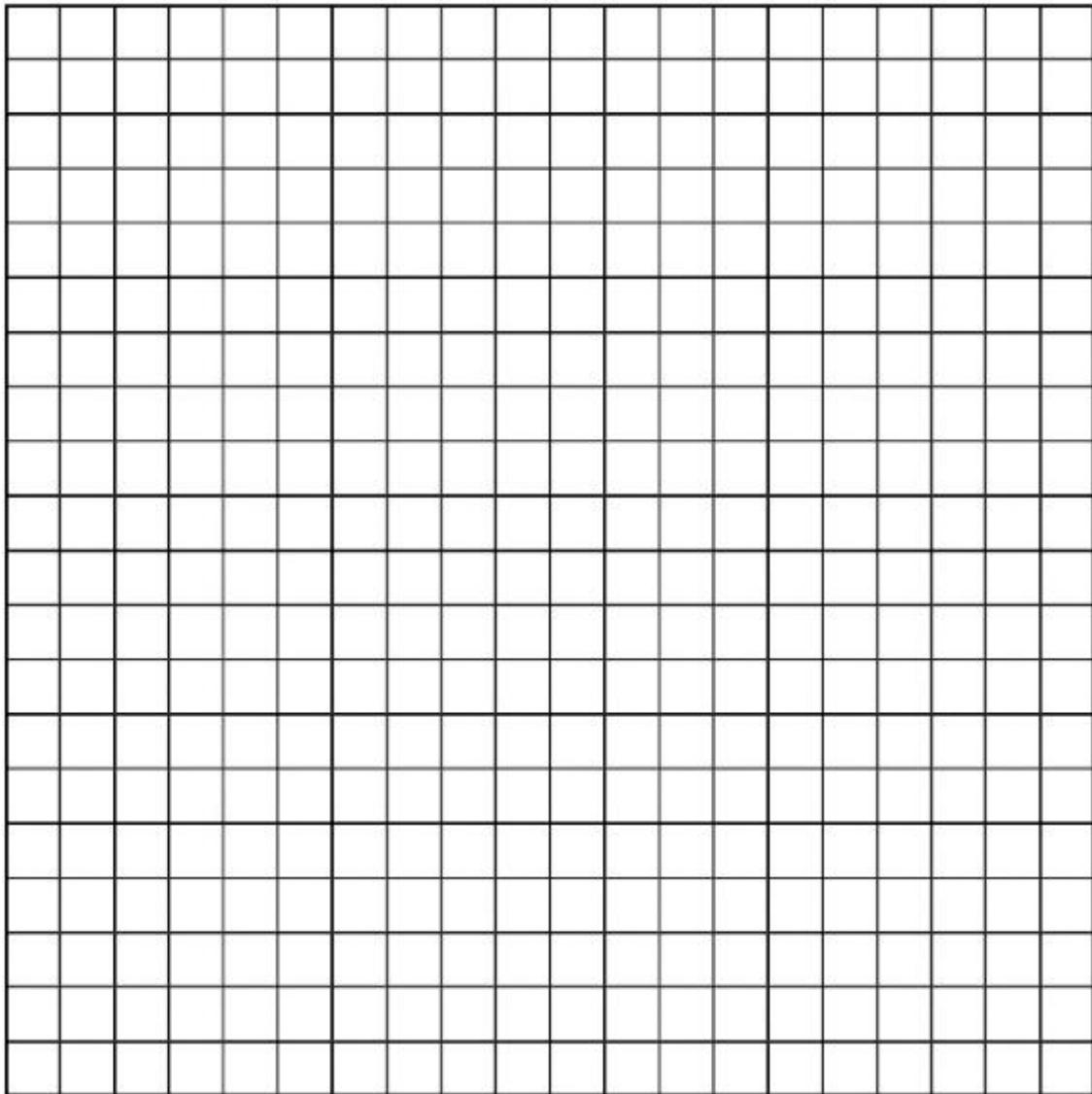


Figure A_3_1: Graph sheet for drawing on

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* :

```
// @description Set some variables
name="Ben";
age=28;
country="England";
food="Pizza";
```

Figure A_4_1. Showing create event

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

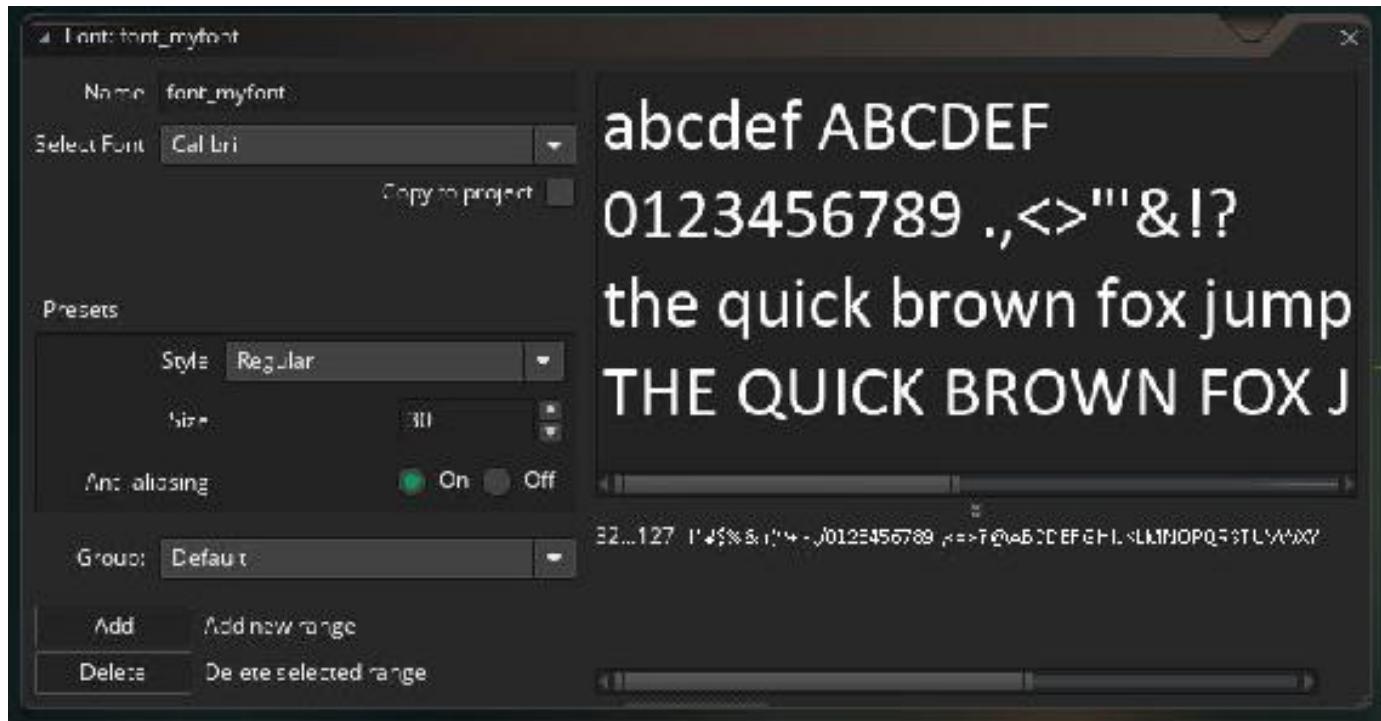
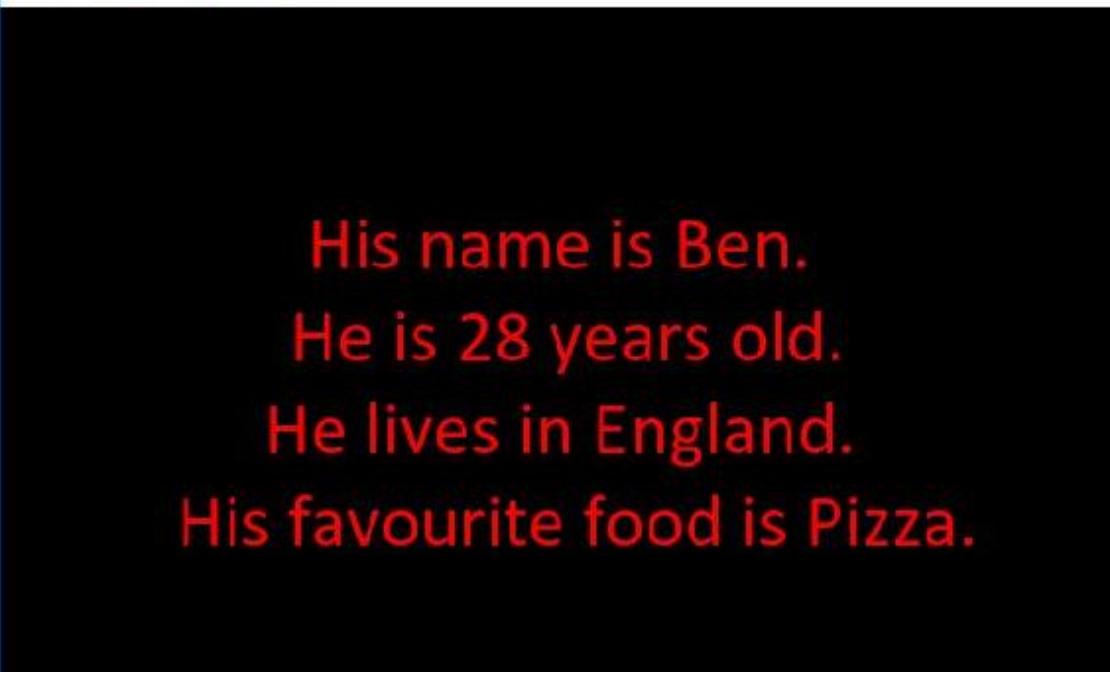


Figure A_4_2: Setting a font

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*:



His name is Ben.
He is 28 years old.
He lives in England.
His favourite food is Pizza.

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:

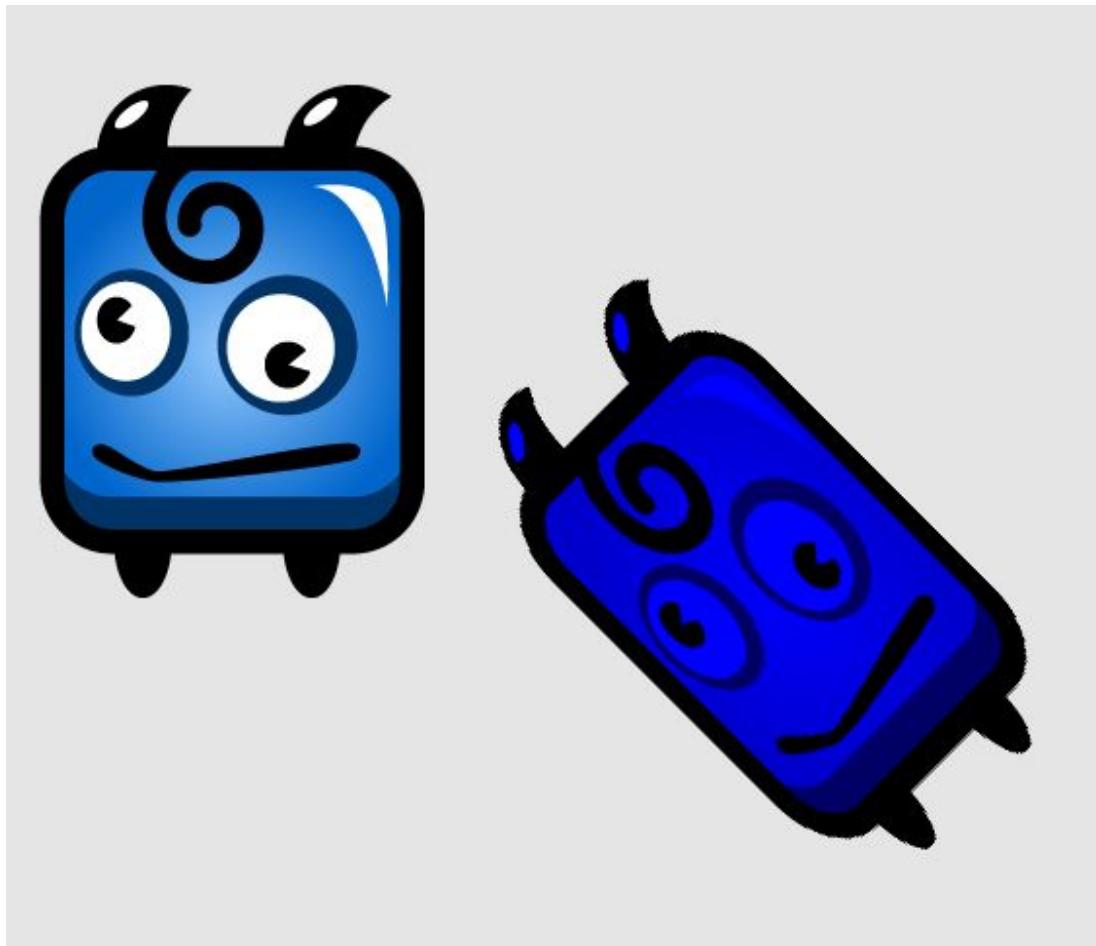


Figure A_4_4: Showing sprite drawn normally, and with `draw_sprite_ext`

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 previously. 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:

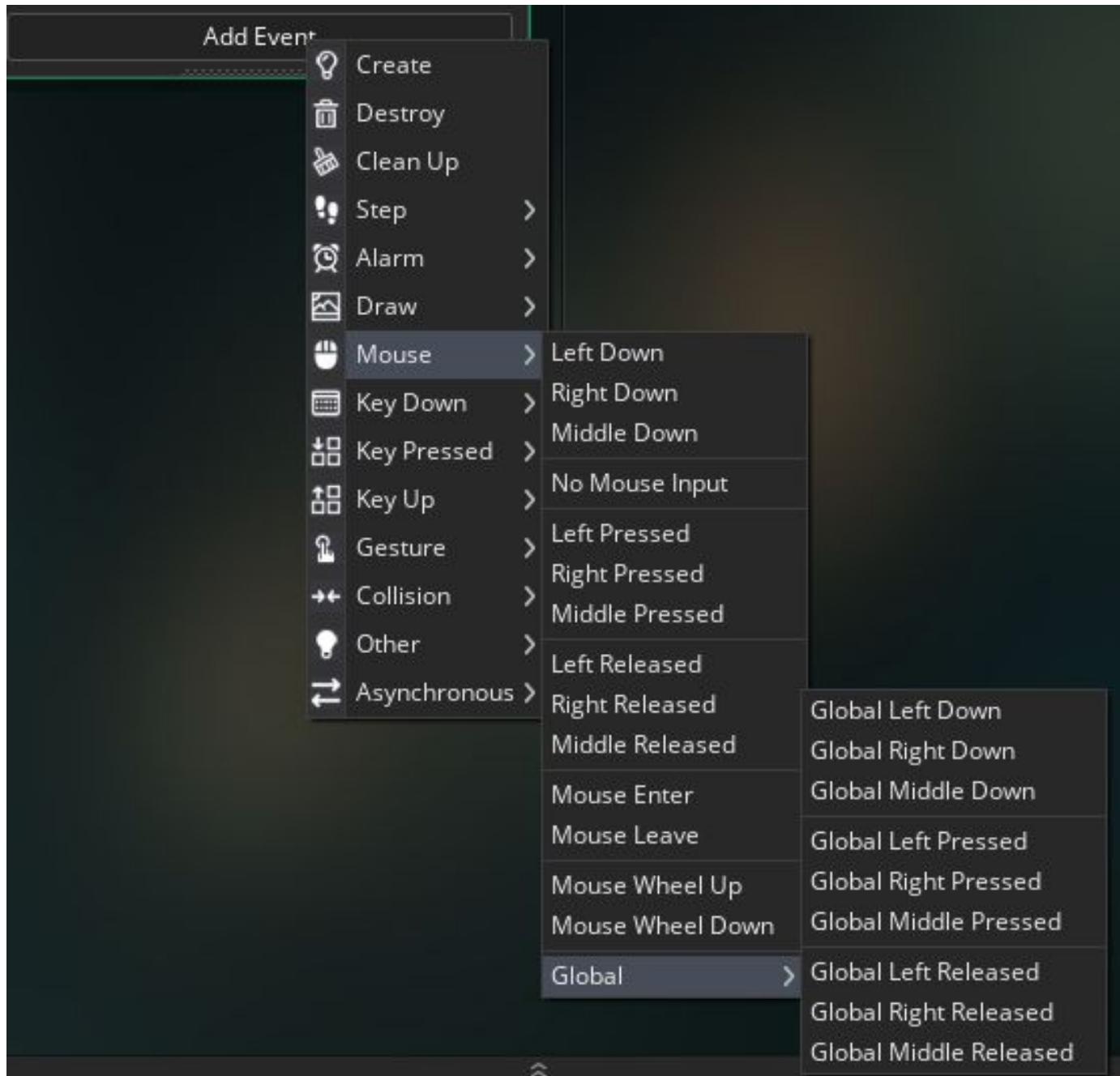


Figure A_6_1: Showing available mouse events

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 emphasise, 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* :

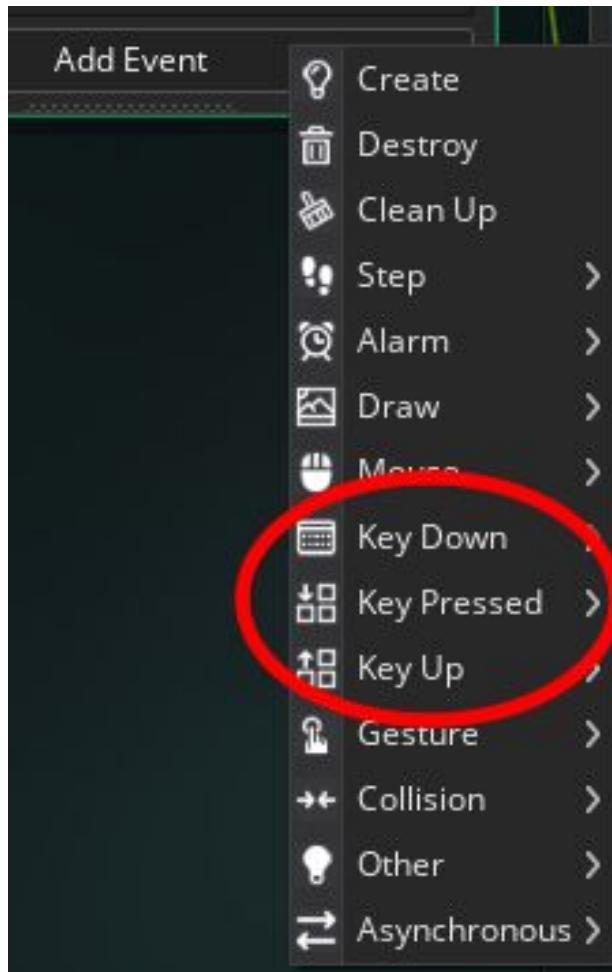


Figure A_6_2: Keyboard events

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*.

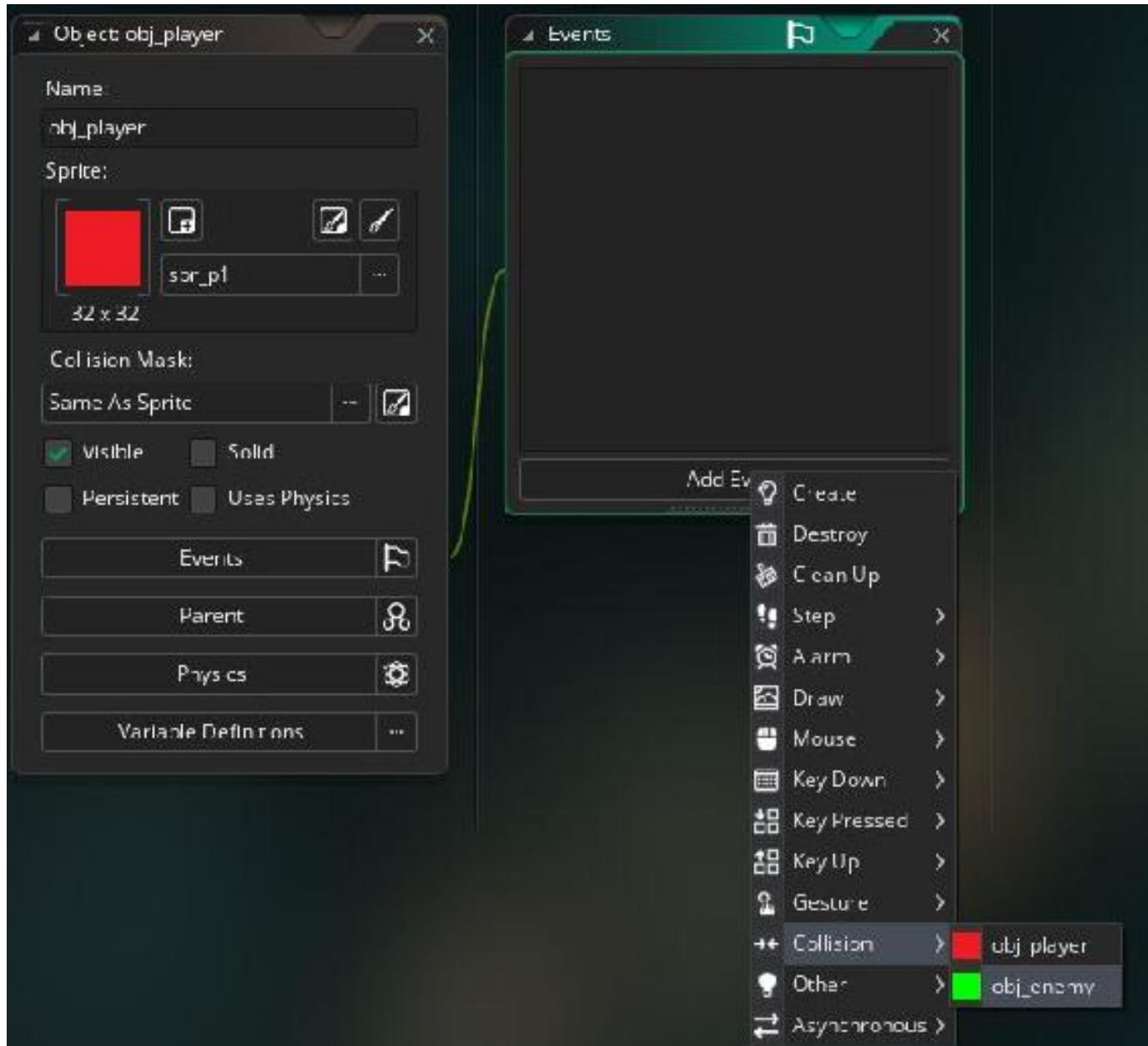


Figure A_6_3: Setting up a collision event

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.

Appendix 7 Sprites

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

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

You set an origin for a sprite. It is this point that will be used for displaying onscreen at an **X** and **Y** location. It is also worth noting that this point is also where transformations such as scaling and rotation are based around. It should be chosen carefully according to what the sprite will be used for. *Figure A_7_1* shows the sprite origin set as center.

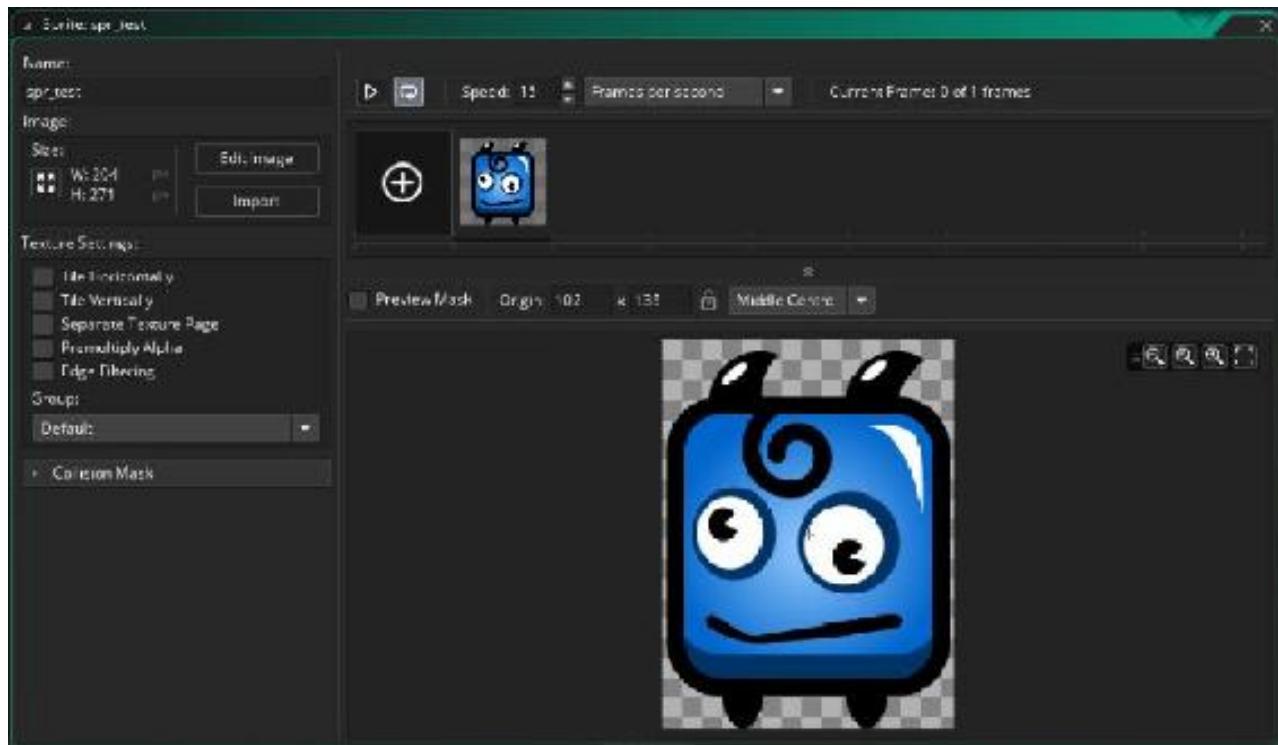


Figure A_7_1: Showing origin as center

Sprites can consist of single images or multiple images. Multiple images may be loaded in from separate images. Create a new sprite, **spr_coins_1**, and click import, as shown in

Figure A_7_2 :

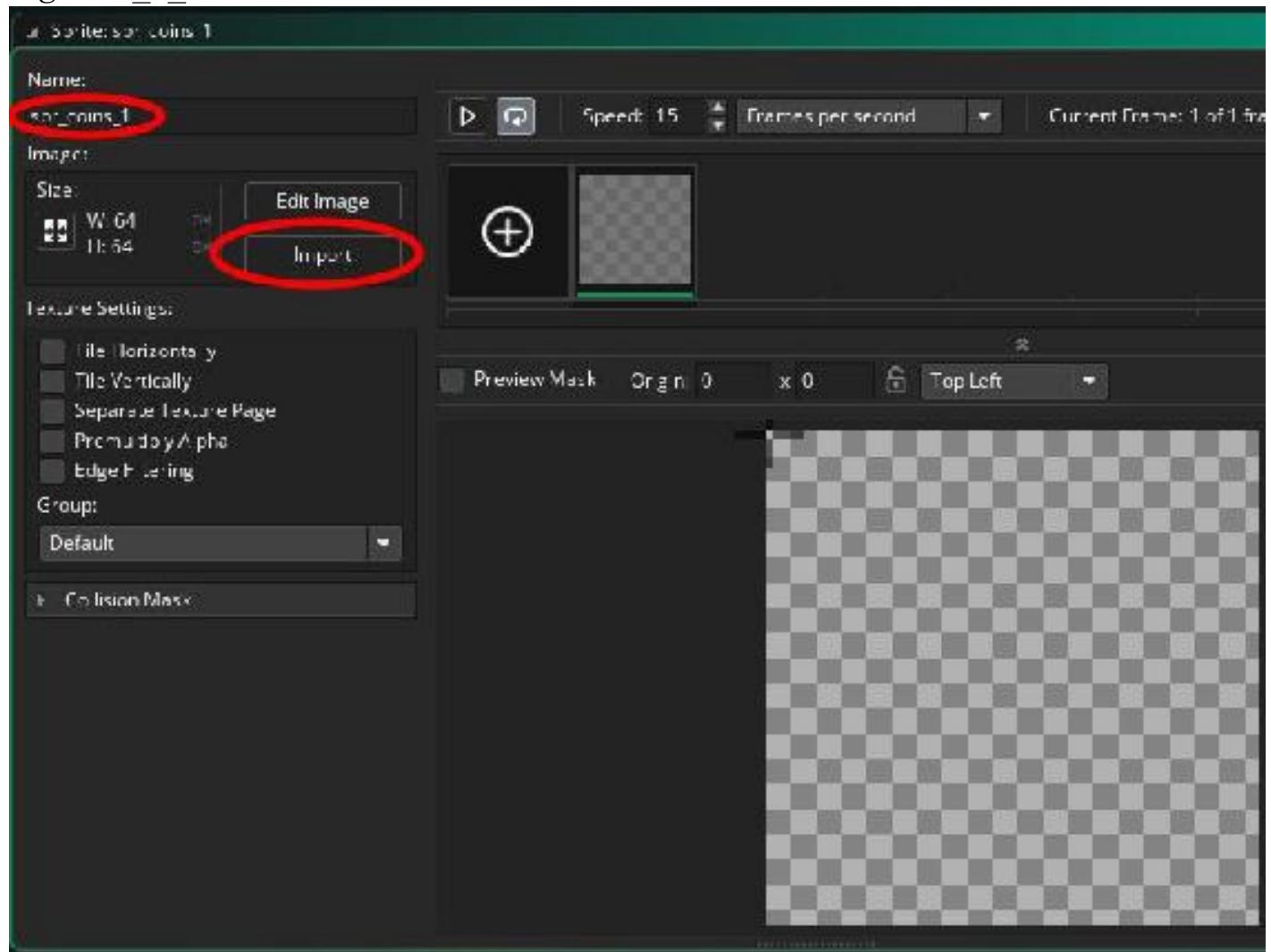


Figure A_7_2: Setting name and selecting import

You can then select multiple images by holding down shift. Select all 8 images for the single coin, as shown in *Figure A_7_3* :

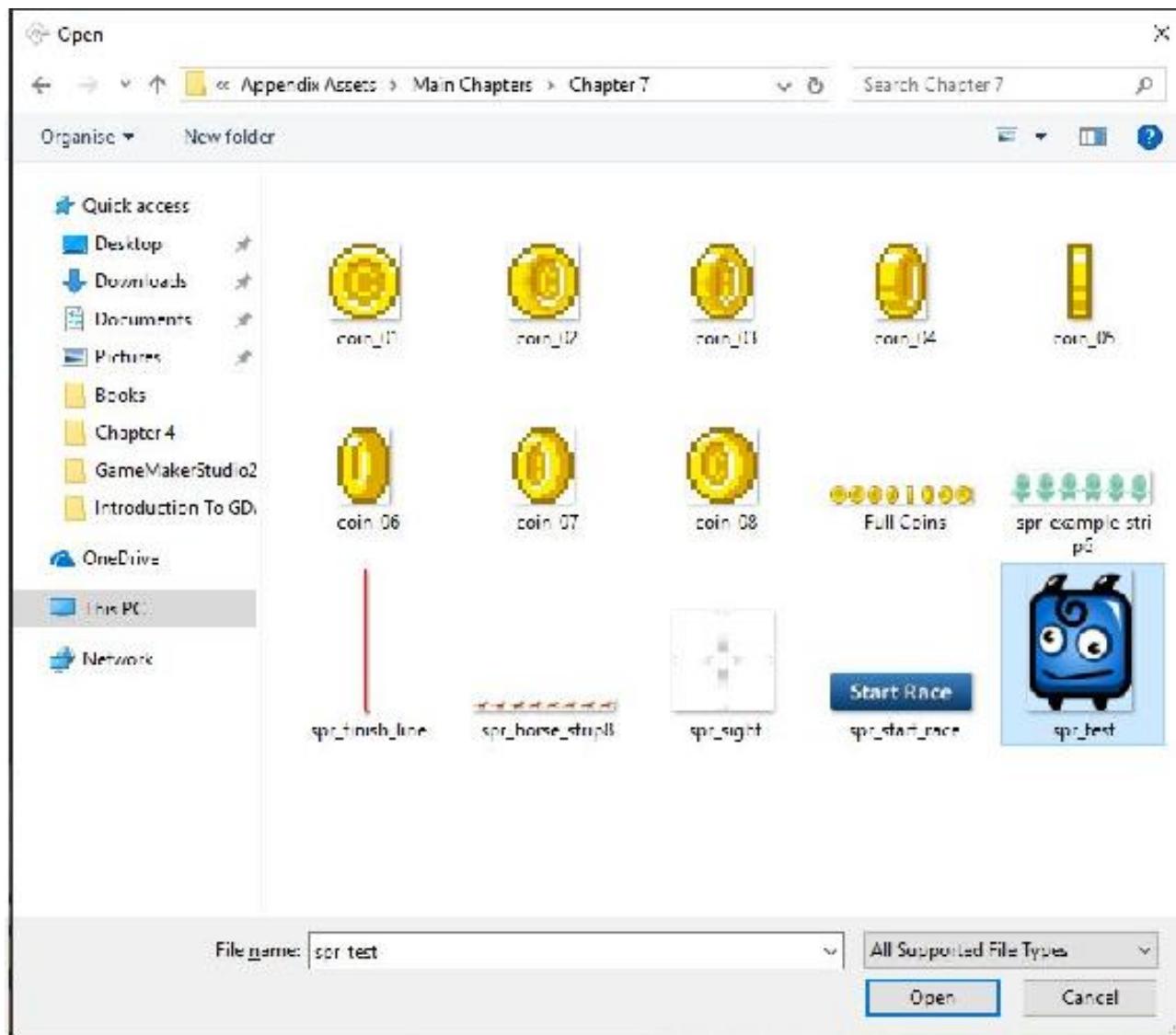


Figure A_7_3: Selecting multiple sub images

You can also import sprite sheets or strips. Create a new sprite, **spr_coin_2** and click on edit, as shown in *Figure A_7_4* :

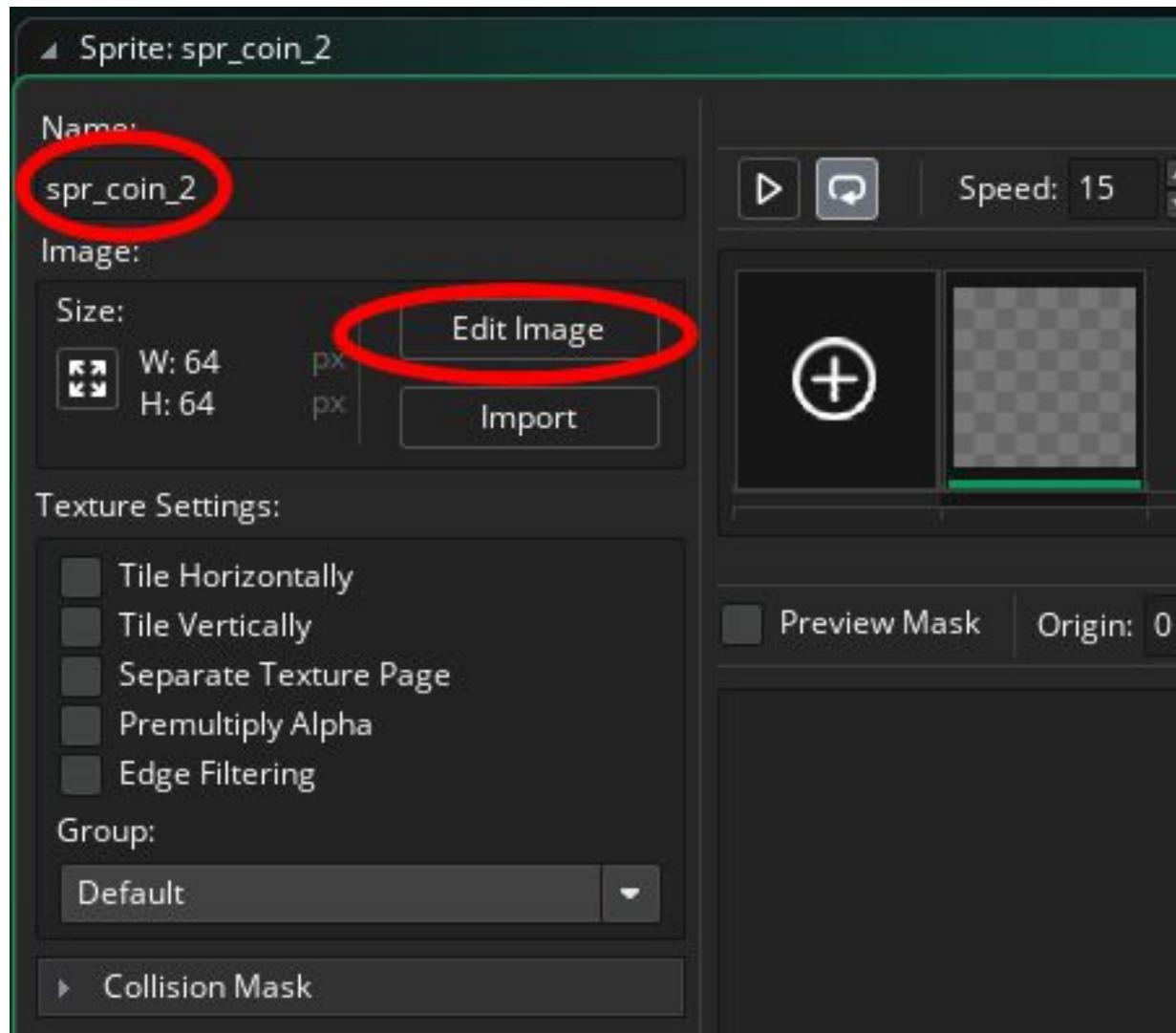


Figure A_7_4: Naming a sprite and selecting edit

Then click image and Import Strip Image as shown in Figure A_7_5 :

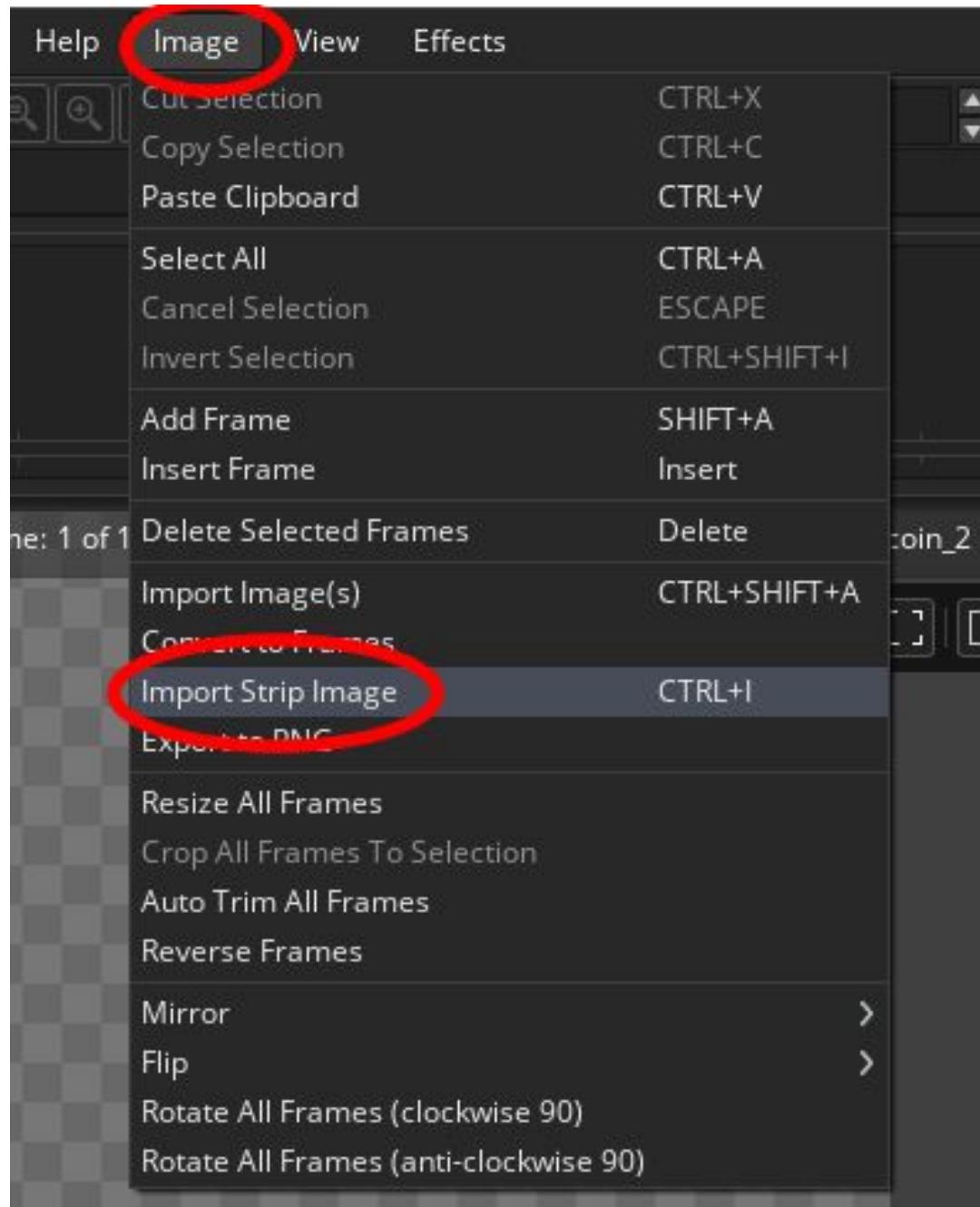


Figure A-7_5: Selecting import strip image option

You can then set as shown in *Figure A-7_6*:

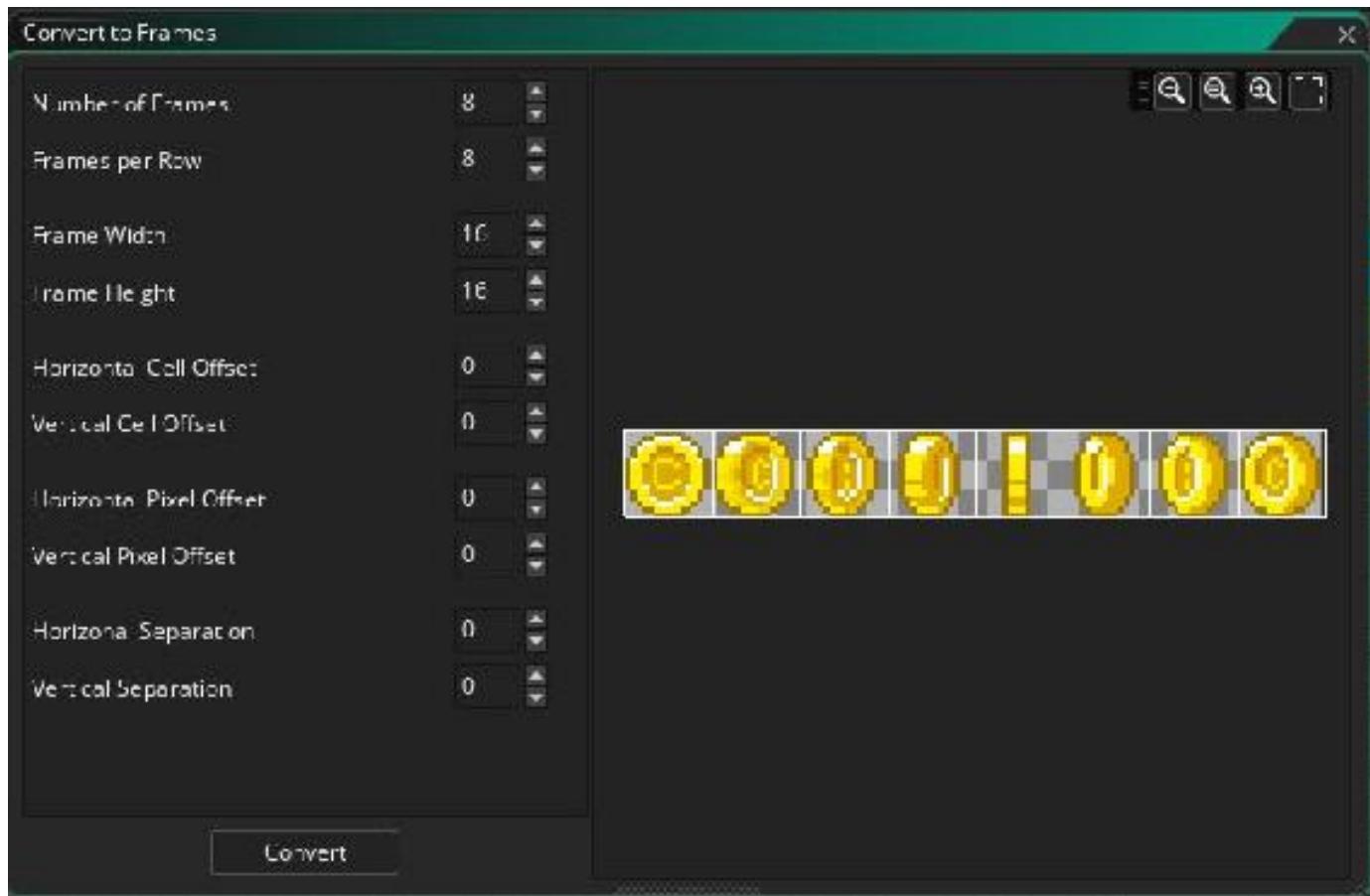


Figure A_7_6: Setting the import properties

You can now delete the sprite **spr_coins_2**, by left clicking on it in the resources tree and selecting delete, as it won't be used in the following example.

Set the origin to center for both the sprites.

Next create an object, **obj_player**, and assign the sprite **spr_test** from the resources folder for this appendix.

Place the following code in the **Step Event**:

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

That is all for this object.

Create an **obj_coins_1** and assign the coin sprite, then place the following code in a Collision Event with **obj_player**:

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

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

Place one of each object in the room **room0**. Now test this game.

An example YYZ for this available in the resources folder

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

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

image_speed – How quickly a sprite's sub images animate.

image_index – Set a specific sub image.

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

draw_sprite_ext(); - Allows drawing with additional settings.

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

```
image_index=0;
```

And when it's moving right:

```
image_index=1;
```

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

```
image_speed=0;
```

```
image_index=0;
```

And when it's moving right:

```
image_speed=0;
```

image_index=1;

Basic Projects

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

Advanced Project

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

Appendix 8 Health, Lives & Score

Most casual games will have some sort of health, lives, or score system. Players usually start off with full health and lose some of it when they get hit by an enemy bullet, land on spikes, or collide with something they shouldn't. Once a player loses all their health, they usually lose a life and are transported back in the level to a re-spawn site.

Different games have different goals, though with a lot of them the aim is to achieve the highest score possible. Fortunately GameMaker Studio 2 makes it really easy to set up such a system described above.

health, **lives**, and **score** are built-in global variables. Which means you don't need to put "global." before them to access or change the value from any instance.

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

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

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

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

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

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

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

Health

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

Some examples:

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

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

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

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

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

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

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

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

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

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

```
if health<=0
```

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

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

There is also a function for drawing a health bar, **Draw the health bar**. For example:

```
draw_healthbar(x-50,y-40,x+50,y-20,  
(health/100)*health,c_red, c_green,c_green,0,true,true);
```

Which would draw a health relative to the instance drawing it. Sometimes, if not relative to an instance you may use a control object to draw the player's health, lives and score.

Lives

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

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

You can also draw *lives* graphically using a bit of code, for example

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

This will space out the lives at 50 pixel intervals; this will look like what is shown in *Figure A_8_1*.

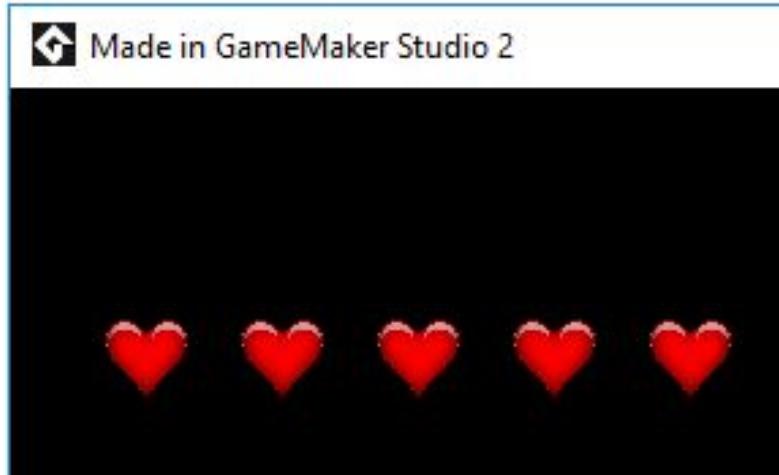


Figure A_8_1: Showing lives drawn using code

Score

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

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

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

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

Step Event

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

Draw Event or Draw GUI Event

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


Basic Projects

- A) Draw a health bar across the whole of the top of the game window, draw *lives* under this as images. Allows keys Q and W to change health value, and A and S to change lives. Limit health to a maximum value of 100.
- B) Draw lives as images, but make the sprite grow and shrink. Do this through code or editing sprites.

Advanced Projects

- C) Draw a bar at the top of the screen that draws the current score mod 1000. For each 1000 score increase the level by 1. Also draw score and level as text. Gains a level for each 1000 points. Allow key Q to increase the score/
- D) Create 3 level buttons that each become clickable for every 1000 score points. Show they are clickable using different sub images.

Appendix 9 Mouse

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

Mouse input can be used for:

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

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

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

```
if mouse_check_button(mb_left) // check for being held down
{
    //do something
}

if mouse_check_button_pressed(mb_left) //activates one time only when button is pressed
{
    //do something
}
```

```
if mouse_check_button_released(mb_left) // activates  
one time only when button is released  
{  
    //do something  
}
```

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

The position of the mouse can be found using:

mouse_x the **X** position of the mouse in the room.

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

For mouse buttons actions you can use, for example:

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

Where *button* can be any of the following:

- mb_left
- mb_right
- mb_middle
- mb_none
- mb_any
- mb_middle means middle button
- mb_none means no button
- mb_any means any mouse button

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

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

```
if (mouse_check_button(mb_left))  
{
```

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

An example YYZ for the above is available in the resources.

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

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

You can detect scroll down in a similar manner:

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

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

cursor_sprite=spr_name;

when **spr_name** is a sprite that exists.

You can hide the default windows cursor using:

window_set_cursor(cr_none);

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

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

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

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

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

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

Basic Projects

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

Advanced Projects

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

Appendix 10 Alarms

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

Alarms can be used for the following:

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

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

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

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

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

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

In total you can have 12 alarms for each object:

alarm[0] alarm[1]

....

alarm[10] alarm[11]

When an alarm activates (alarm goes off / runs out of time) you can detect this using an **Alarm Event**, which can be selected as shown in *Figure A_10_I* :

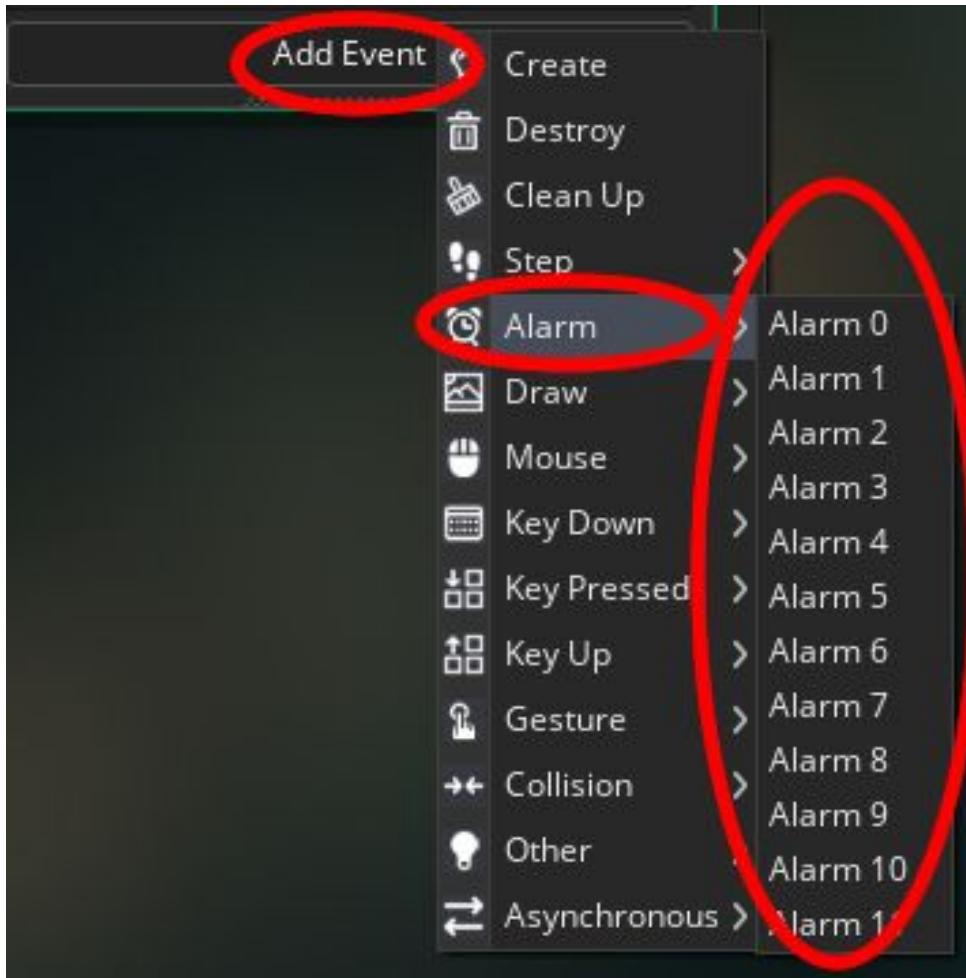


Figure A_10_1: Showing alarm events

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

alarm[0]+=1*room_speed;

Alarms run until they reach 0, which is when they trigger, and then count one more to -1, so you can stop an alarm by setting it to -1.

Here are some examples:

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

alarm[0]=5*room_speed;

Alarm[0] Event:

room_goto(room_menu);

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

Create Event:

```
can_shoot=true;
```

When player shoots, by pressing the Z key:

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

Alarm [1] Event:

```
can_shoot=true;
```

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

Note: Be careful about holding alarms open by constantly setting them.

Use a flag to prevent this, as was done with can_shoot in the example above.

Basic Projects

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

Advanced Projects

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

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

```
ball=instance_create_layer(x,y,"Instances",obj_ball);
ball.speed=2; ball.health=50;
```

Appendix 11 Collisions

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

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

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

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

Most of the time it's sufficient to use the **Collision Event**. You can create a **Collision Event** as shown in *Figure A_11_1*; just select the object you want to check a collision with.

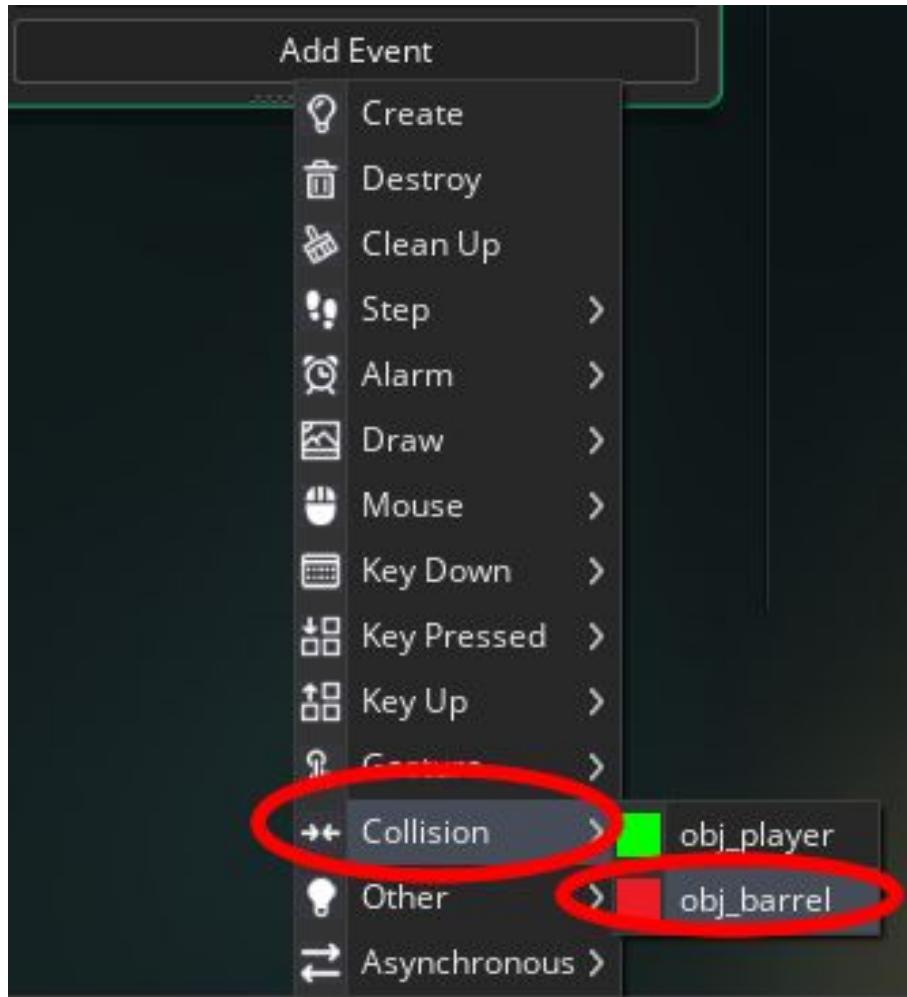


Figure A_11_1: Setting up a collision event

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

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

place_empty(x, y);

As above, but checks for solid objects only:

place_free(x, y);

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

place_meeting(x, y, object);

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

position_meeting(mouse_x, mouse_y, id);

You can destroy all objects at a location:

position_destroy(x, y);

Or the following, which finds the instance ID:

instance_position(x, y, obj);

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

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

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

Collision Line

For this example you'll need three sprites, **spr_player** in green and **spr_enemy** in red and **spr_wall** in blue. A size of 32x32 with origin set as center for each will be fine.

In the **Step Event** of **obj_player** and assign the sprite, put the following; this will allow you to move the player object using the arrow keys:

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

Create an object **obj_enemy** and assign the sprite . In the **Draw Event** of **obj_target** put the following code, which will draw a line between **obj_target** and **obj_player** if there is a direct line of sight (i.e., no walls in the way):

draw_self();

```

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

```

Place one instance of **obj_player**, and a few each of **obj_target** and **obj_wall** into a new room and test. *Figure A_11_2* shows this in action:

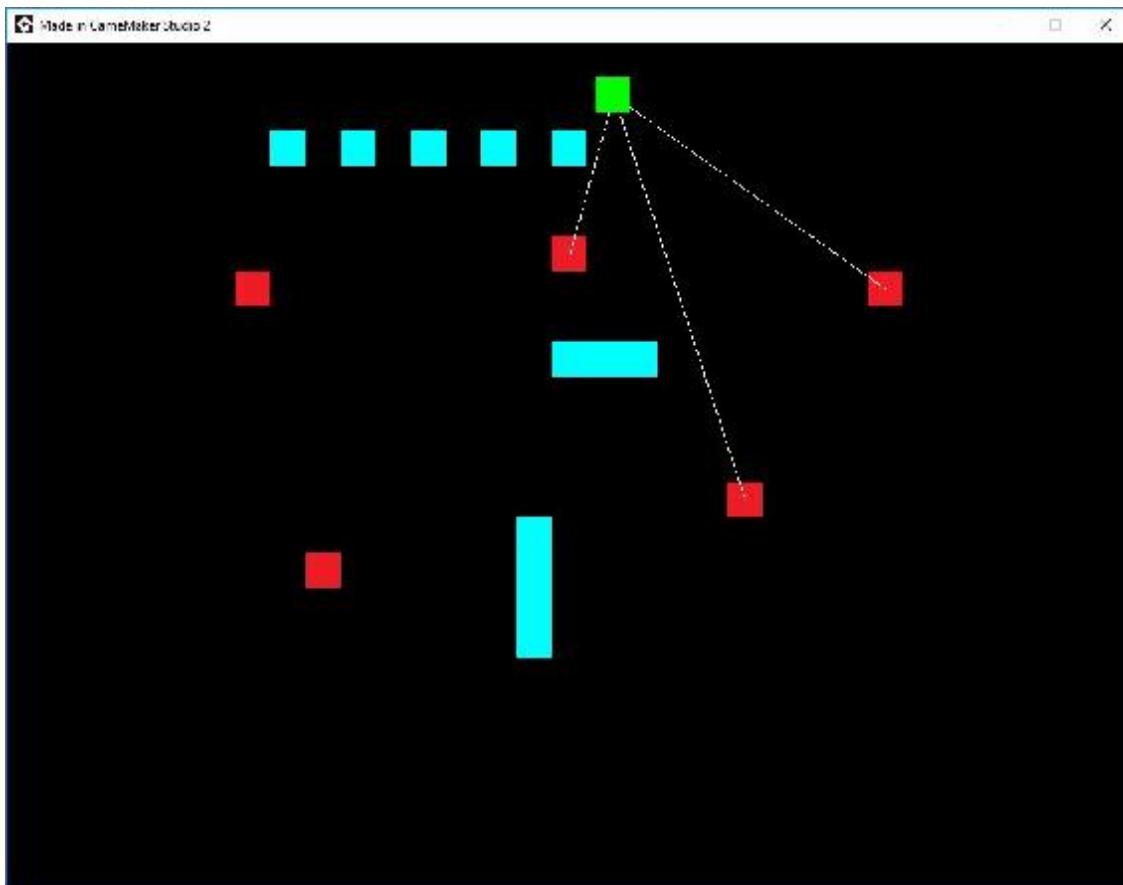


Figure A_11_2: Showing example room with objects added

A YYZ for the above is available in the resources.

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

Basic Projects

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

Advanced Projects

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

Appendix 12 Rooms

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

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

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

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

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

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

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

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

Create a new GameMaker Studio 2 project, and load in a new background.

Backgrounds are loaded in the same way as sprites, for example as shown in *Figure*

A_12_1 :

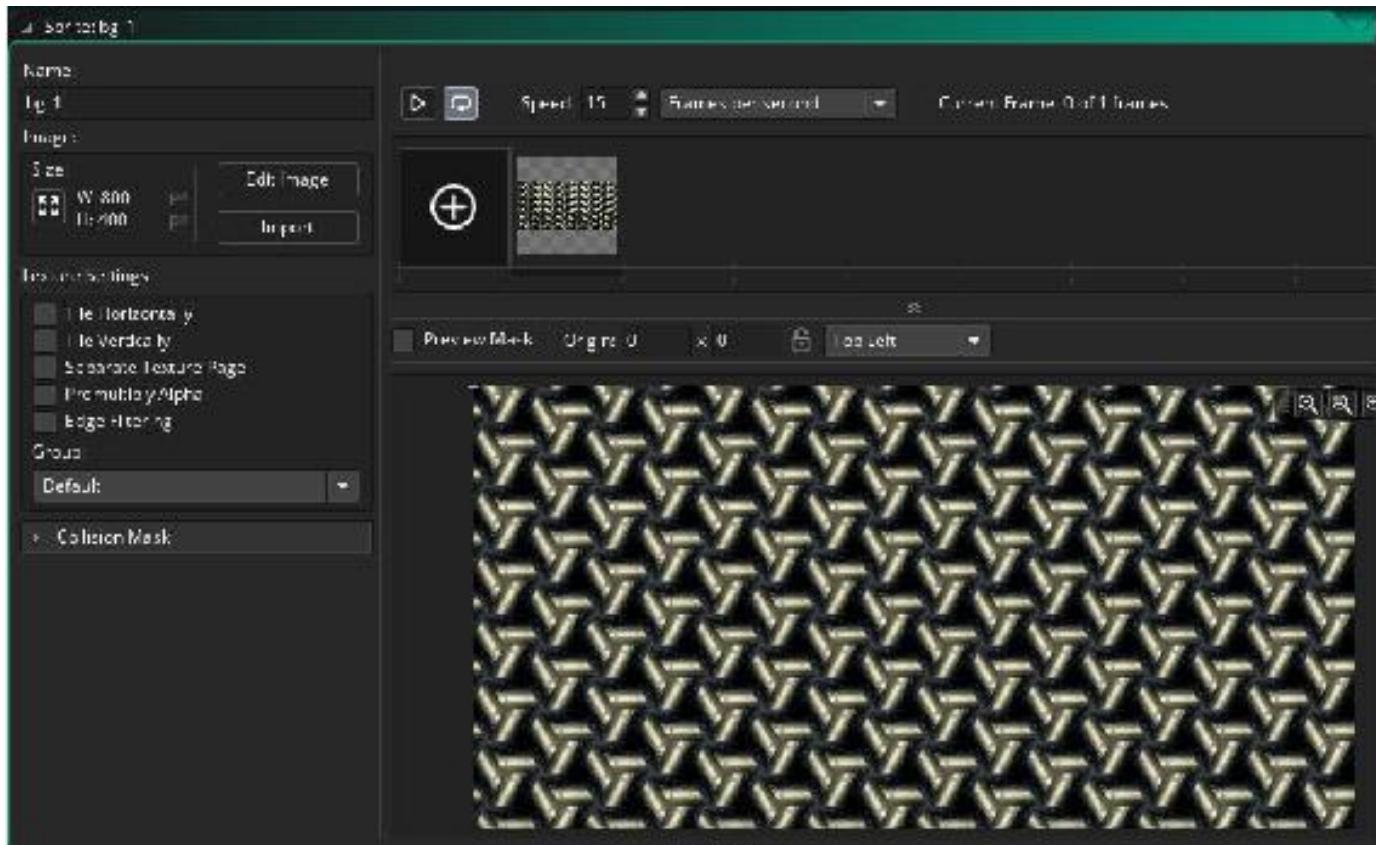


Figure A_12_1: Loading a sprite to use as a background

Note: For naming purposes, use something `bg_1` so you know it will be used as a background

You can set this background in a room. Create a new room, **room_example**, and select the background layer, and set the room dimensions as 800 by 400, set the background image as the background you just loaded in, shown in *Figure A_12_2* :

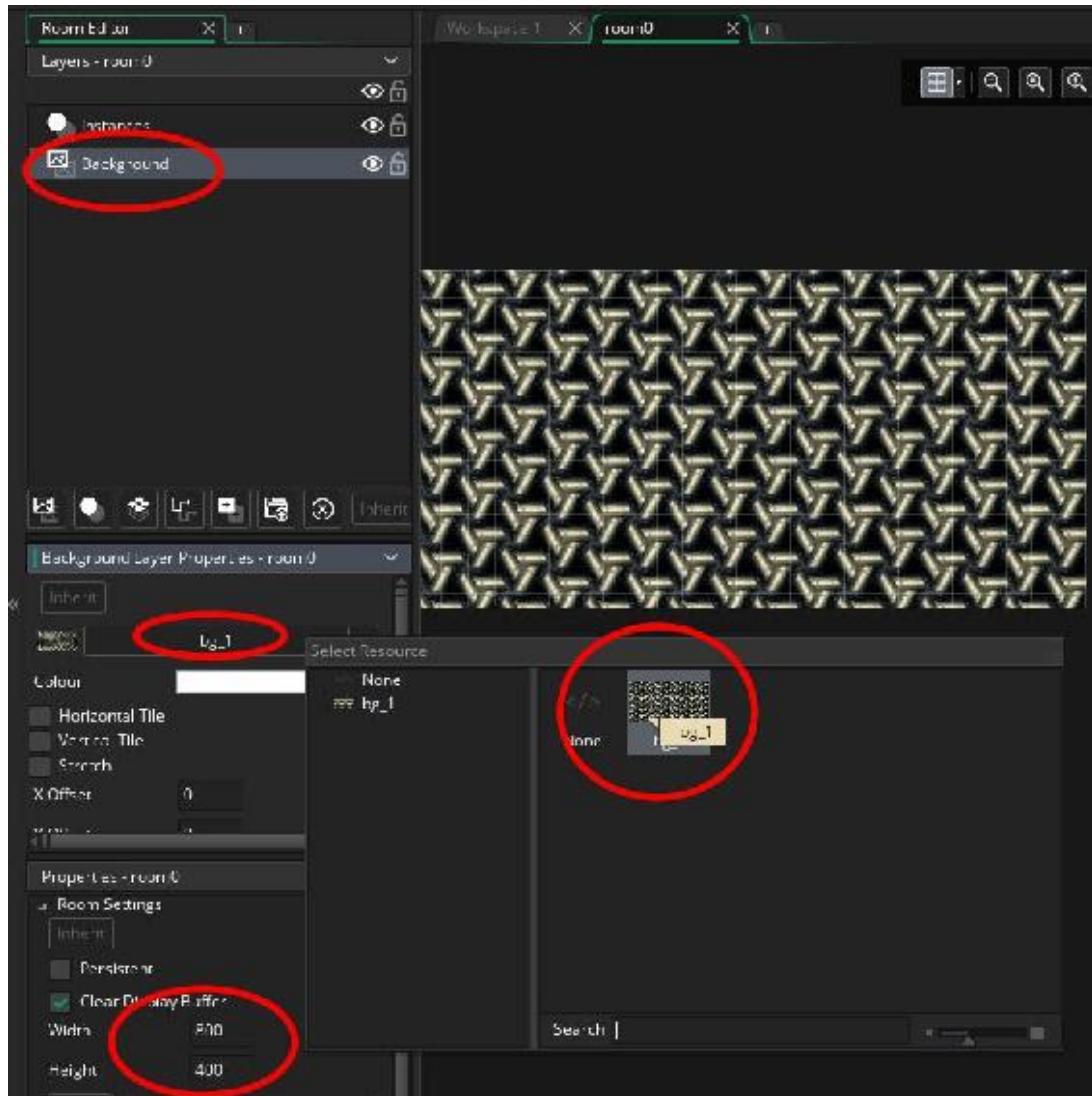


Figure A_12_2: Setting a background

Backgrounds are dealt with in more detail in appendix 14.

Next we'll look at the **views** tab. Start a new project.

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

Create a new project.

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

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

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

Create another object, **obj_wall**, and create and assign a blue sprite for it. No code is needed for this. Open **room0** and set the room width and height to 2000 each. Place one

instance of **obj_player** and multiple of **obj_wall**. It doesn't matter too much where you place them. An example is shown in *Figure A_12_3*.

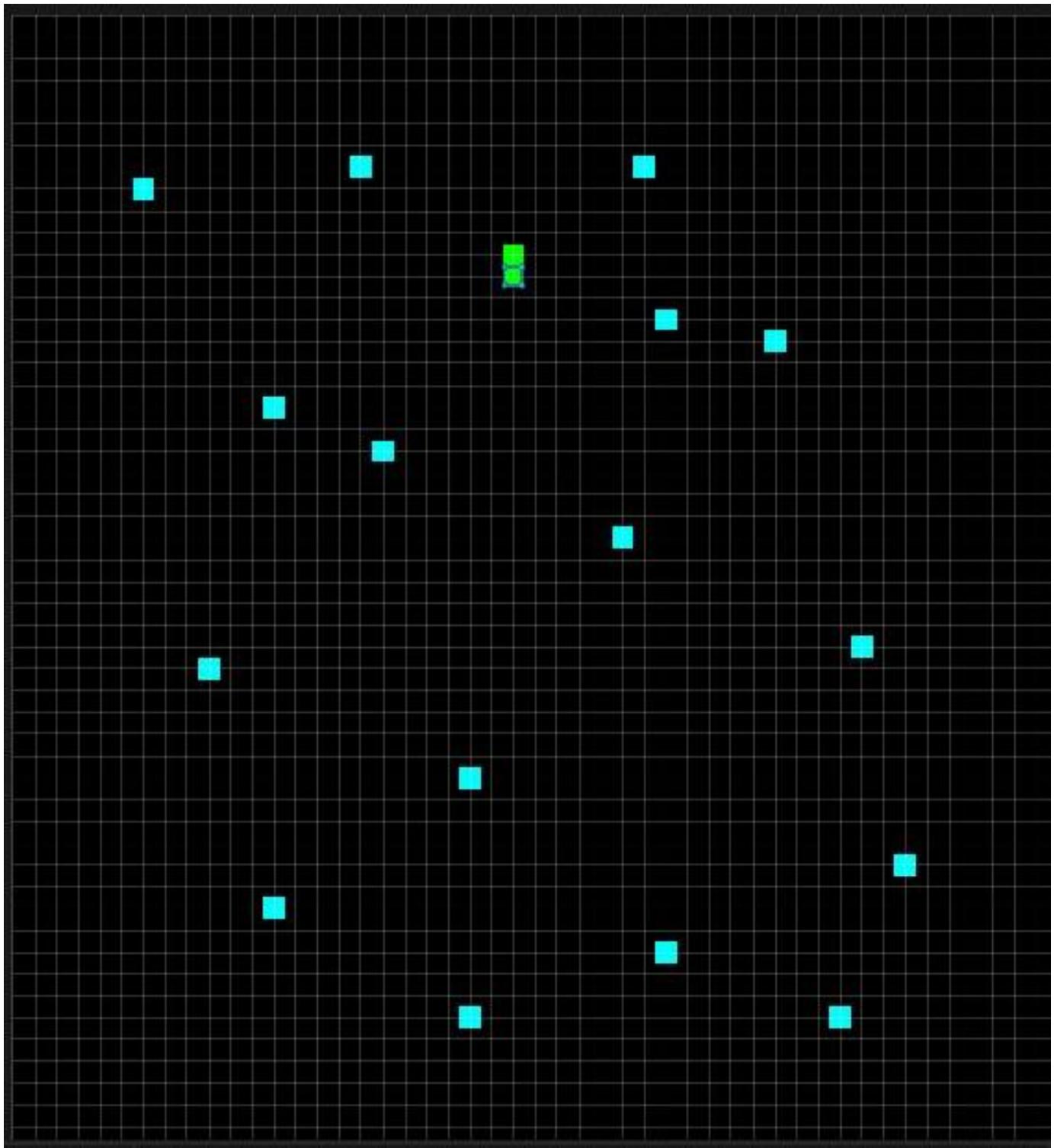


Figure A_12_3: Showing instances placed in room

Set up a view, as shown in *Figure A_12_4*. This will create a view that keeps the player visible on the screen.

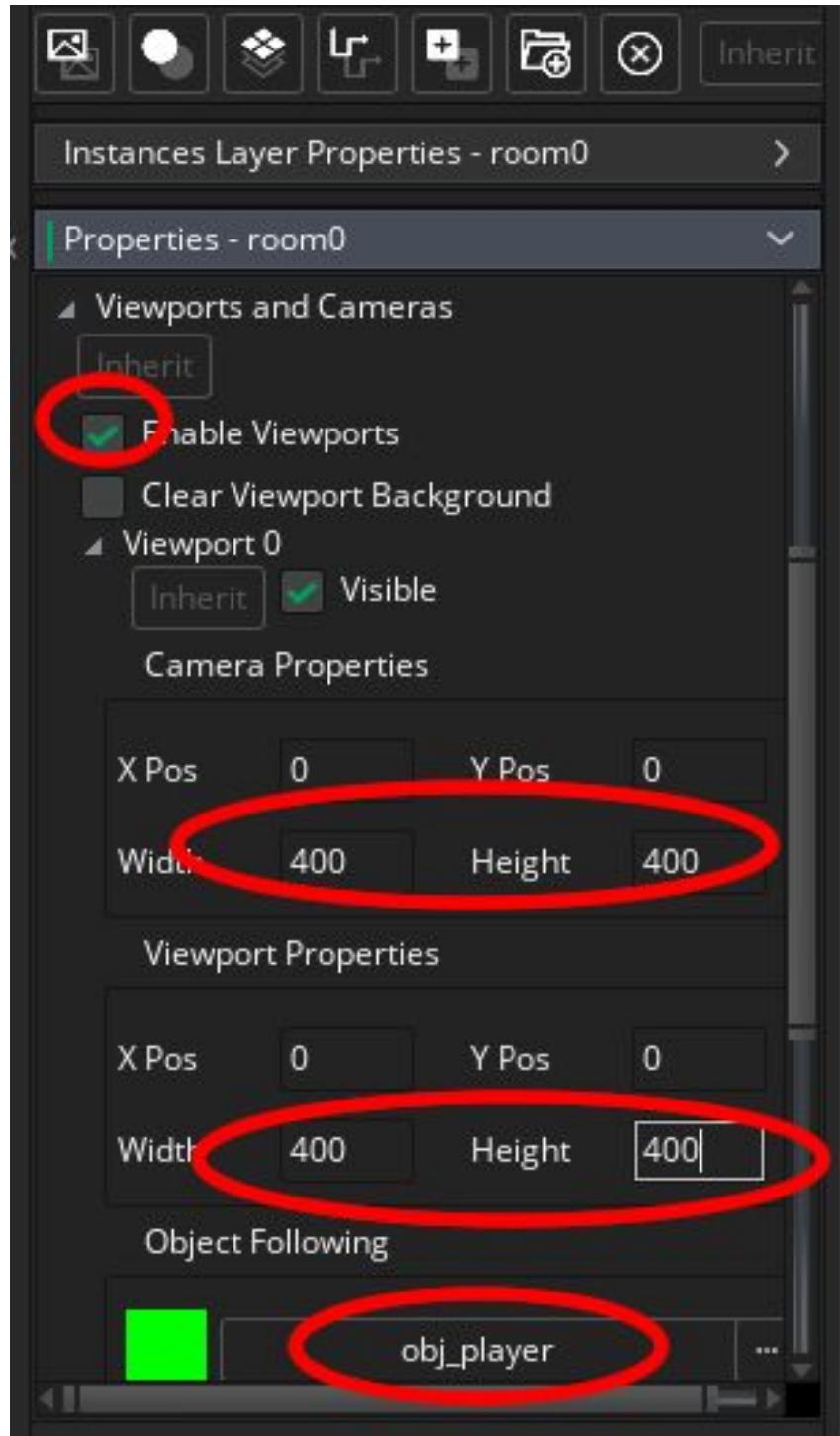


Figure A_12_4: Setting up view to follow an object

An example for this is in the resources folder for appendix 12

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

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

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

This one will take you to a named room:

room_goto(room_name);

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

room_goto_next();

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

room_goto_previous();

You can restart the room using:

room_restart();

New rooms can be created by right clicking on Rooms in the resources tree.

You can name a room by right-clicking on it and selecting rename.

Basic Projects

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

Advanced Project

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

A	B
---	---

Make the player wrap up and down in each room.

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

Appendix 13 Backgrounds

Backgrounds are one of the basic resources that are used for rooms in which the game takes place. Backgrounds can be made of one large image or tiles of a smaller image. Additionally, you can have static and moving backgrounds in GameMaker Studio 2. These backgrounds can be set using the room editor, or in GML. You can also set and change backgrounds using GML.

Backgrounds are images that show in the room but do not have any direct interaction with objects. You can combine two or more backgrounds to create a parallax scrolling effect. Some uses of backgrounds include:

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

Let's make an example by creating a new project. We are going to create a moving background. This is useful as it creates a more professional look to the game, and it is useful for scrolling type games. Load in a background from the resources, and name it as `bg_wire`. Set it as shown in *Figure A_13_1*.

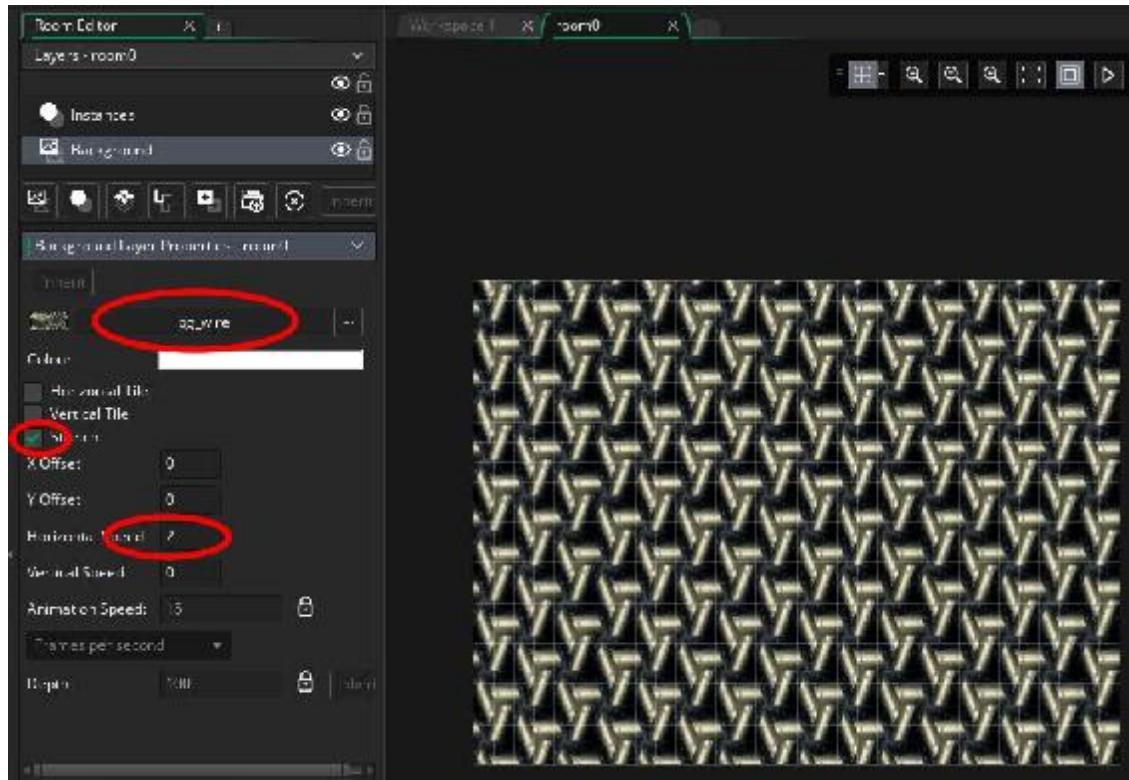


Figure A_13_1: Setting up a background

If you now test you will see that the background now moves to the right.

If you wanted to move the background to the left, you can set the Horizontal Speed to a negative value, for example -2.

You can tile a background as shown in *Figure A_13_2* :

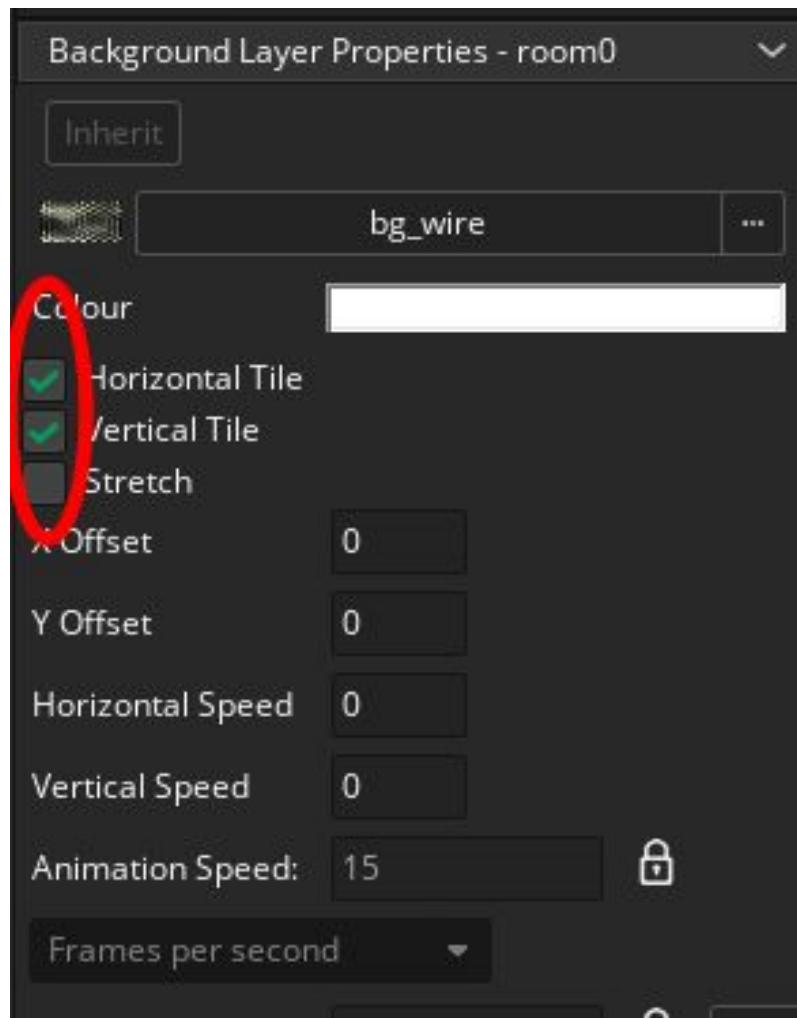


Figure A_13_2: Setting background to tile

You can change the horizontal and vertical (x and y) positions using code. For example
Create Event:

```
/// @description setup  
yy=y;
```

Step Event:

```
/// @description Move bg  
yy=yy+2;  
layer_y("Background",yy);
```

The above code would move by 2 pixels per step.

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

You can have multiple backgrounds, which can be used to great effect.

You can create a new layer, by clicking where shown in *Figure A_13_3* :

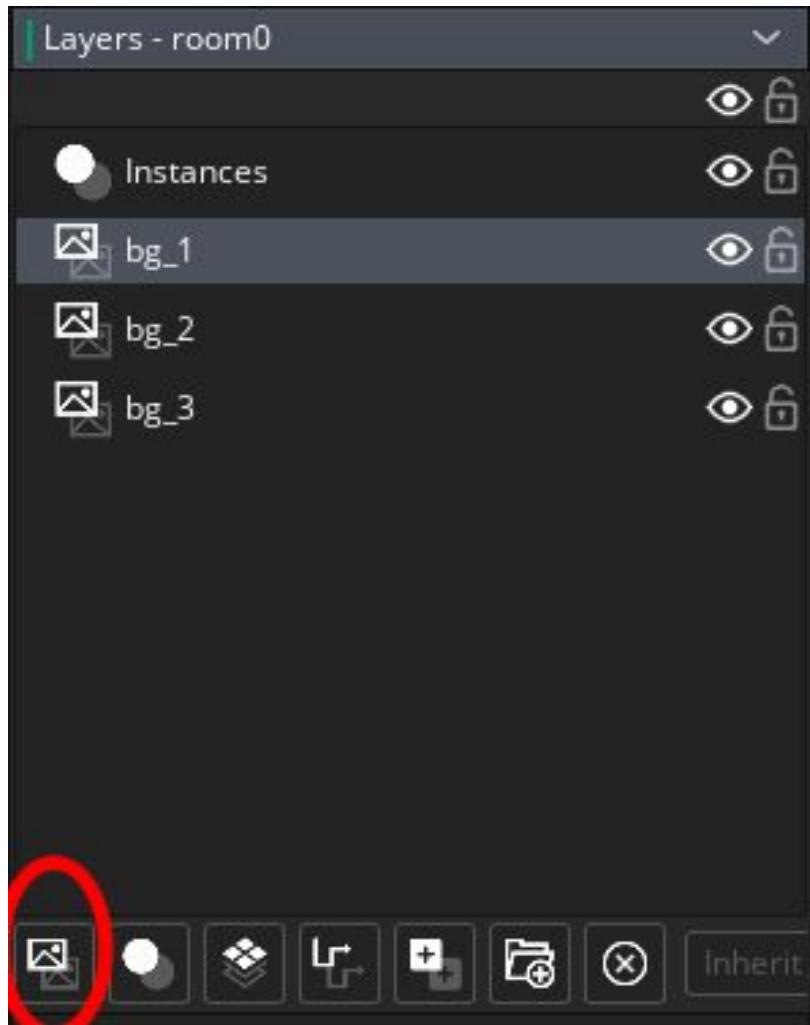


Figure A_13_3: Creating a new background layer

You can set a background visible or not with:

layer_background_visible(bg_name,true);

Basic Projects

- A) Make a program that changes background shown if keys 1-3 are pressed.
- B) Make a tiled background that scrolls up and left.

Advanced Project

- C) Create a tiled background that moves in the opposite direction to arrow key presses. (effects like this are used in a lot of styles of games)

Background assets are available for this in the downloadable resources.

Appendix 14 Sounds

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

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

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

You can create a new sound by right clicking where shown in *Figure A_14_1*:

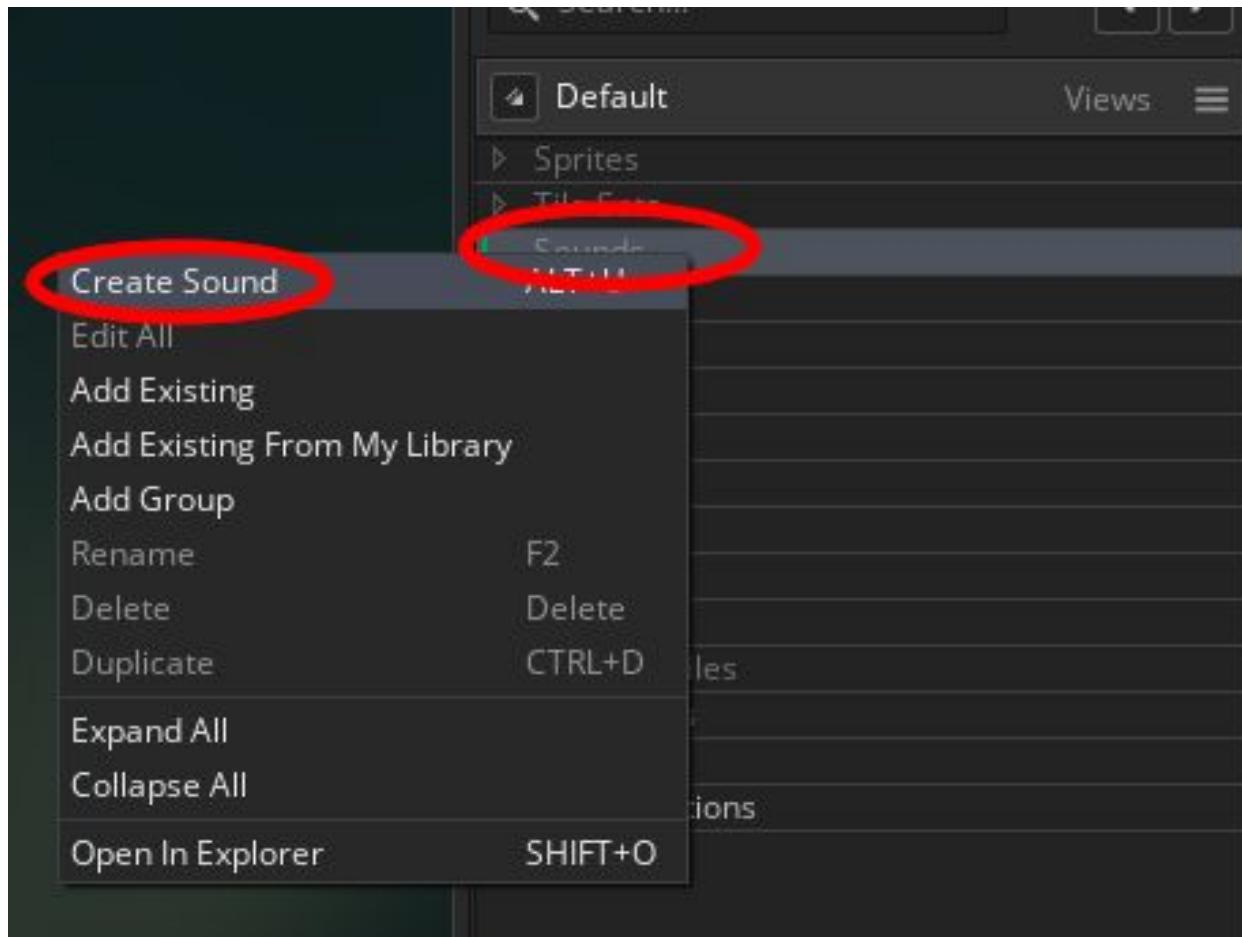


Figure A_14_1: Creating a new sound

Load in the sound effect, **bells** from the resources folder for this appendix, and name the sound **snd_bell**; this step is shown in *Figure A_14_2*.

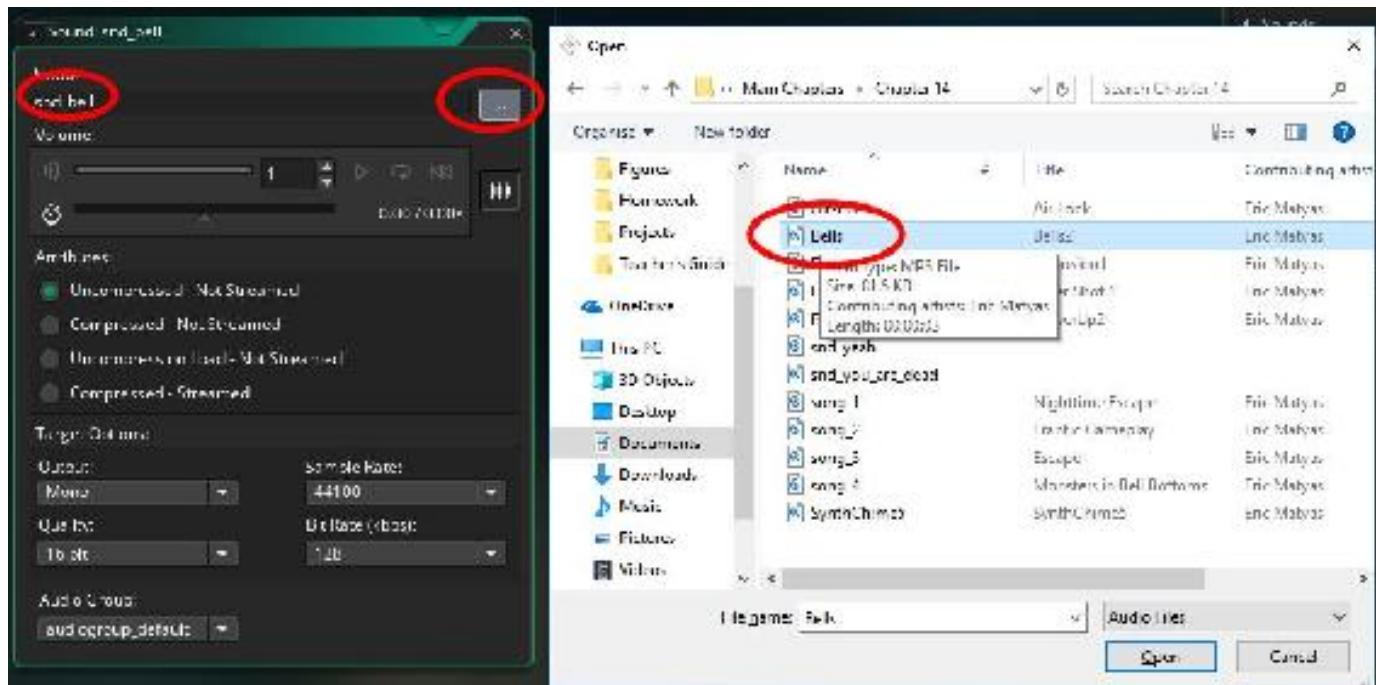


Figure A_14_2: Naming and loading a sound

Create an object and place this in the Step Event:

```
/// @description Play sound on X
if keyboard_check(ord("X"))
{
    audio_play_sound(snd_bell,1,false);
}
```

The **snd_bell** is the sound file to play, **1** is the channel priority, and **false** stops the sound from looping.

Place one instance of this object in **room0**.

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

Add a new sound, **snd_music_1** and load **song_1** from the resources. Put the following code in to a **Create Event** of object **obj_example**.

```
/// @description Play music on loop
audio_play_sound(snd_music_1,1,true);
```

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

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

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

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

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

An example for this, including pause and resume, is in the resources.

Some additional functions include:

audio_pause_sound
audio_resume_sound
audio_stop_all

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

Create Event:

audio_play_sound(snd_music_1,1,true);

Step Event:

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

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

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

Create Event:

audio_play_sound(snd_music,1,true);

Step Event:

```
if (keyboard_check_pressed(ord("X")))
```

```
{  
    audio_stop_all();  
}
```

Basic Projects

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

Advanced Projects

- C) Play some music. Adjust the volume based on y's mouse position.
(see `audio_sound_gain`)

Appendix 15 Splash Screens & Menu

A splash screen is usually shown when a game starts up. It's an ideal time to initialize variables and load external resources. A splash is usually a single room and a control object. It will load / set the data and then take you to another room, for example a menu or the main game level. A good way to show a splash screen is to create a new room, assign a background or sprite, and then move it to the top of the room's resources tree, for example, as shown in *Figure A_15_1*.

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

A completed game rooms tree may look something like this:

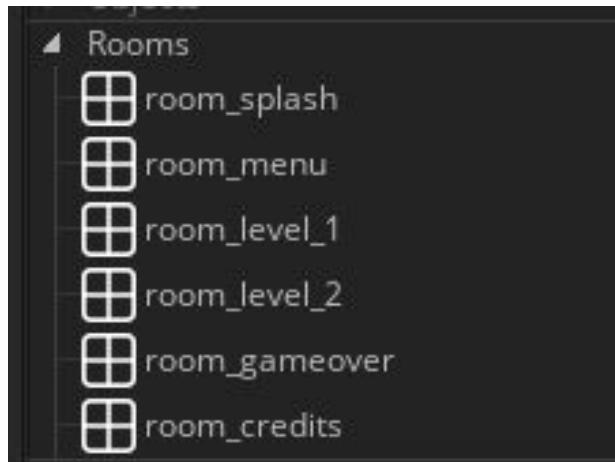


Figure A_15_1: Showing an example of room order in resource tree

Go ahead and these rooms as shown in the figure above.

As you will see above, in *Figure A_15_1*, each room has a distinct name. The room at the top of the tree will be loaded first.

Start a new project within GameMakerStudio 2.

Create a new object, **obj_splash_screen**. With the **Create Event** code and splash sprite loaded and assigned (in resources for this appendix):

```
global.level=1;  
lives=5; //Set initial lives to 5  
score=0; //Start with a score of 0  
global.bonus_score=0; //Set with a value of  
health=100; //Start with full health
```

```
alarm[0]=7*room_speed; //Set alarm for 7 seconds
```

Alarm[0] Event code:

```
room_goto_next();
```

Place one instance of this object in **room_splash** and test.

Start a new project.

Go ahead and load in the project file HW5B.

Rename **room0** as **room_level_select**. Delete **obj_control**.

Create some extra rooms so the resource tree looks like that in *Figure A_15_2* :

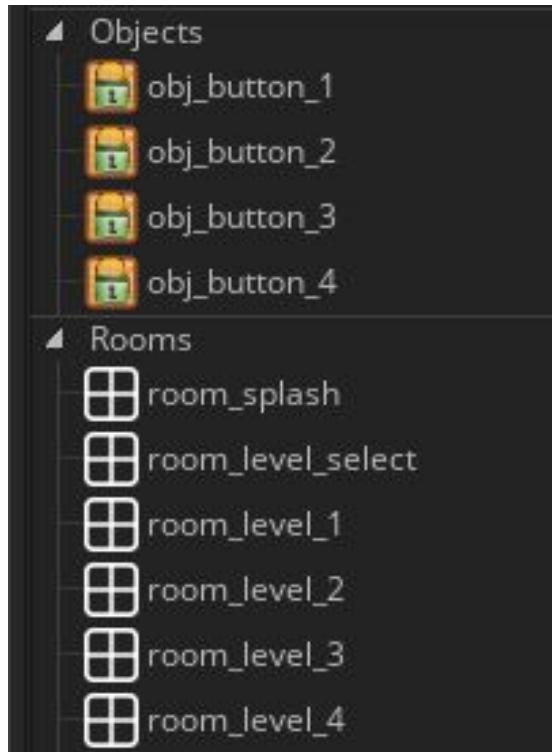


Figure A_15_2: Showing changes to resources tree

Create an object **obj_splash** with the **Create Event** code:

```
/// @description Set up
global.level=2;
room_goto_next();
```

Place one instance of this object in **room_splash**.

Change the **Step Event** of **obj_button_1** to:

```
/// @description control
```

```

if position_meeting(mouse_x,mouse_y,id) &&
mouse_check_button_released(mb_left)//if mouse
pressed over button
{
    if my_level-1>=global.level//check local viable against
level
    {
        show_message("locked");
    }
    else
    {
        show_message("unlocked");
        // @description Goto room
        room_goto(room_level_1);
    }
}

```

```

if my_level-1>=global.level//check local viable against
level
{
    image_index=0;//set image index
}
else
{
    image_index=1;
}

```

Add similar code for the other buttons, changing the **room_level_1** to **room_level_2**, **room_level_3**, and **room_level_4**, accordingly.

An example project is in the folder for this appendix.

If you save and test, you will see the game starts with level 1 and 2 unlocked. In appendix 18 you will learn how to save and load level values so they present when a game is started.

Basic Projects

- A) Create a splash screen and place an object with a sprite animation. Set it to go to the next room 5 seconds after the animation has ended. (use code or animation end event to check for animation ending)
- B) Create a menu room with background music with a button for it to go to an instructions room (with button to return to menu room).

Advanced Project

- C) Create an object that becomes unlocked only if user types in a password on the keyboard. Make the password “bacon.” Display visibly whether the object is locked or unlocked.

Appendix 16 Random

Randomization can be used for a lot of things:

- Choosing a random sound effect
- Spawning an enemy at a random location
- Adding diversity to your game
- Selecting an effect to use
- Making AI move / change direction
- Choose an attack from a selection
- Choose a random sound effect / backing music
- Move an object to a new location

A static game with the exact same pattern of movement doesn't have much replay value. This type of game can be memorized. Using randomness in your games to create variety in game play greatly increases the game replay value.

Strictly speaking there is no such thing as a true random number on a regular PC, but it can be simulated and approximated for game play.

For the purposes of making games easier to create and test, GameMaker Studio 2 will always generate the same random numbers each time a game is played. For example, if in your game you rolled a dice 5 times and got 5 3 1 6 3 2, the next time you played from the Studio IDE you'd also get 5 3 1 6 3 2. Depending on your game you may want different numbers each time when testing, so remember that a final executable will not display this behavior. To do this, use **randomize()**, which only needs to be called once at the start of the game.

For example, in the **Create Event** of an object in your splash screen you may have:

randomize();

Then in your game you may have:

starting_x=irandom(800);

starting_y=irandom(400);

Now each time the game is run, starting_x and starting_y will have different random values. Randomness can be used for such things as the following:

An example of a random function:

attack=choose(1, 1, 1, 2, 2, 3);

This will choose a number at random. There will be a 50%(3/6) chance of getting a '1,' a 33%(2/6) chance of getting a '2,' and a 17%(1/6) chance of getting a '3'. Weighted randomness like the above code is a common feature in a lot of games.

You can also use other things as well as numbers: for example, objects, sounds, colours:

music_track=choose(snd_track_1, snd_track_2, snd_track_3);
text_colour=choose(c_green, c_red, c_blue);
spawn_enemy=choose(obj_enemy_1, obj_enemy_2);

You can create a real random number:

number=random(50);

This will choose a random number between 0 and 50 exclusive, with decimal places.

An example of a value returned by the random() function: 44.7768221937 (which may contain more digits than this).

If you are generating real numbers with decimals, the following is useful; however floor is generally used for randomized values:

floor(4.2) would return 4

ceil(4.2) would return 5

round (4.2) would return 4

A method more suited to most applications, as it creates whole integers is:

number=irandom(50);

This will choose a whole number integer between 0 and 50 inclusive.

An example of this value is 38.

You can also choose a whole random integer number between values, for example, between 40 and 50 inclusive:

irandom_range(40, 50);

There may be times that you want the randomization to be the same each time, even when the game has been compiled to the final executable for distribution: for example,

when you have a randomly generated level. To achieve this, you can use the following function:

random_set_seed(number);

If you want a random outcome each time.

Note: This should be placed just before any deterministic results are generated.

Basic Projects

- A) Play a random sound every time the player presses the spacebar.
- B) Make an object jump to a random position, no closer than 50 pixels near the edge of the window, when clicked with the mouse.
- C) Make an object move randomly around the room without going off of the edges. Make it change direction every 5 seconds. If it meets edge of room, make it change direction away from the edge.

Advanced Project

- D) Create a lottery game that chooses six different numbers between 1 and 49. Display them in ascending order, inside a circle. If the number is between 1 and 9 make the circle white, 10-19 blue, 20-29 green, 30-39 red, 40-49 yellow.

Appendix 17 AI

Most casual games involve a player and some form of computer AI (Artificial Intelligence). Basically an AI object will interact with what's onscreen and what the player does. For example, in a top-down game the AI object may move toward the player avoiding other objects and shoot a bullet toward the player if they're within a certain range. Usually in the first few levels of a game the AI object will be more forgiving to the player, but higher levels may make the AI more aggressive.

Some of the functions the AI may have:

- Move toward player
- Fire a bullet / missile toward player
- Decide which card to play next
- Punch the player if they're not blocking

In most cases you'll be checking whether a variable equals some number, or if a function is **true** or **false** – and use this to create the illusion of AI.

This starts with a very basic AI and progressively makes it more intelligent.

We'll create a basic AI system that makes an enemy move toward the player, if there is a direct line of sight. **Start a new project.**

Create three new objects, **obj_wall**, **obj_player**, and **obj_enemy** and assign a sprite for each, each 32x32 in size..

Place the following code in the **Step Event** of **obj_player**. This code will allow the player to move so long as it does not collide in the direction it is traveling into an **obj_wall** instance:

```
// @description movement
hor=4*(keyboard_check(vk_right)-
keyboard_check(vk_left));
vert=4*(keyboard_check(vk_down)-
keyboard_check(vk_up));
if !position_meeting(x+hor+(sign(hor)*16),y,obj_wall)
{
    x+=hor;
```

```
}

if !position_meeting(x,y+vert+(sign(vert)*16),obj_wall)
{
    y+=vert;
}
```

The above code will check that there is not a wall in the direction that the player wants to move, if there isn't then it will move. That is all for this object. Create a basic room and check that the player cannot move through the walls, as shown in *Figure A_17_1*:

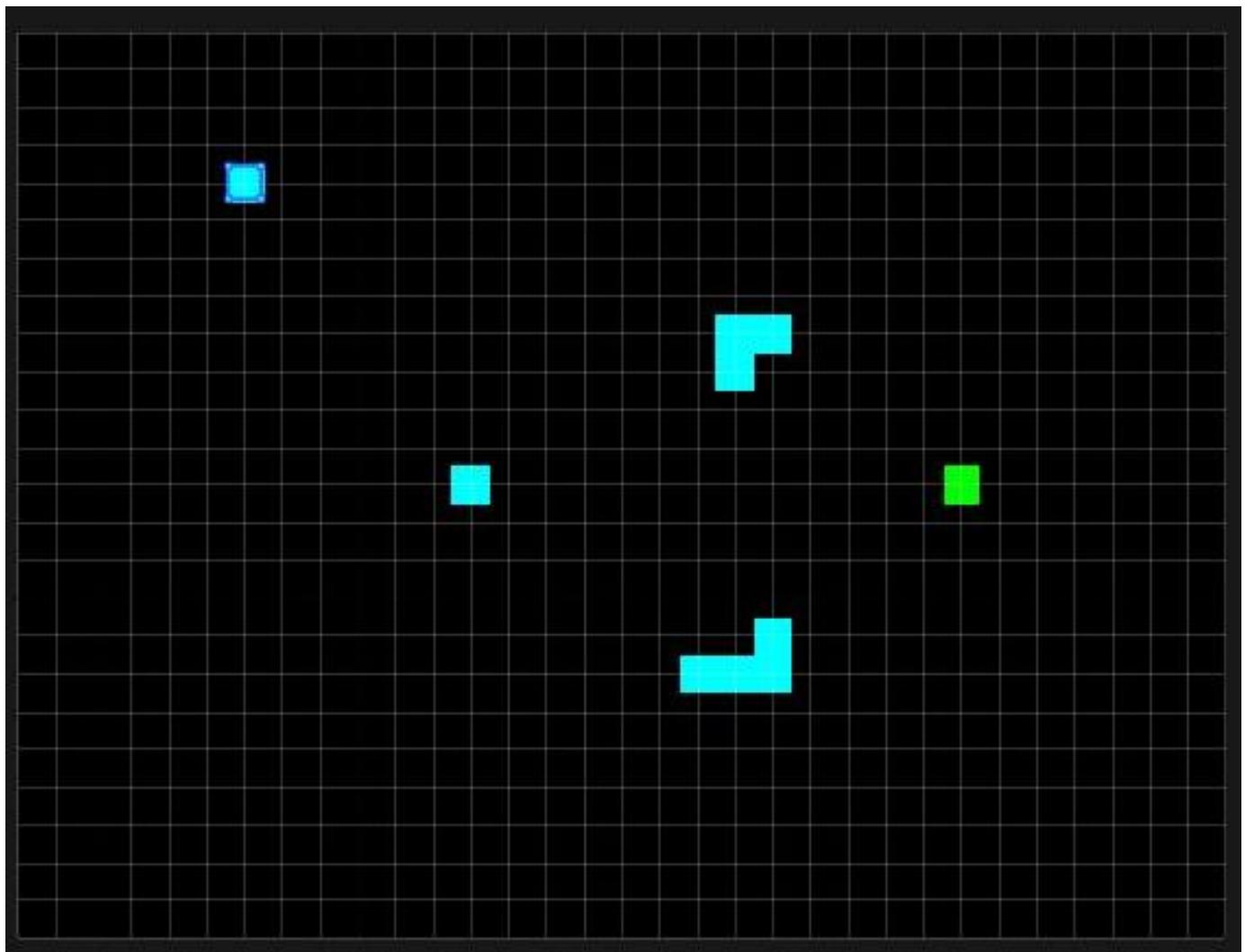


Figure A_17_1: Room for testing

A project file for this, example_1, is in the resources folder for this appendix.

Open up object **obj_enemy** and put the following code in a **Create Event**. This will create a flag that can be changed and tested upon, with true if it has a direct line of sight to the player, or false otherwise:

```
/// @description Set Flag  
can_see=false;
```

In the **Step Event** of this object put:

```
//if there is a direct line of sight between the player and  
the enemy then set can_see to  
true, otherwise false  
if (collision_line(x, y, obj_player.x, obj_player.y,  
obj_wall, true,false))!=noone  
{  
    can_see=false;  
} else  
{  
    can_see=true;  
}  
if can_see  
{  
    mp_potential_step( obj_player.x, obj_player.y, 2,  
true);  
}
```

collision_line returns whether an imaginary line between it and the player instance collides with an instance of **obj_wall**. It returns false if it collides with **obj_wall**.

mp_potential_step(obj_player.x, obj_player.y, 2, true);
moves the object 2 pixels towards the
player for each frame.

Finally create a **Draw Event** for this object with the following code:

```
draw_self();
if can_see
{
    draw_line(x, y, obj_player.x, obj_player.y);
}
```

This will draw its own sprite, and a line between itself and player if **can_see** is true.

Place one instance of **obj_enemy** in the room and test. It will look like that in *Figure A_17_2*:

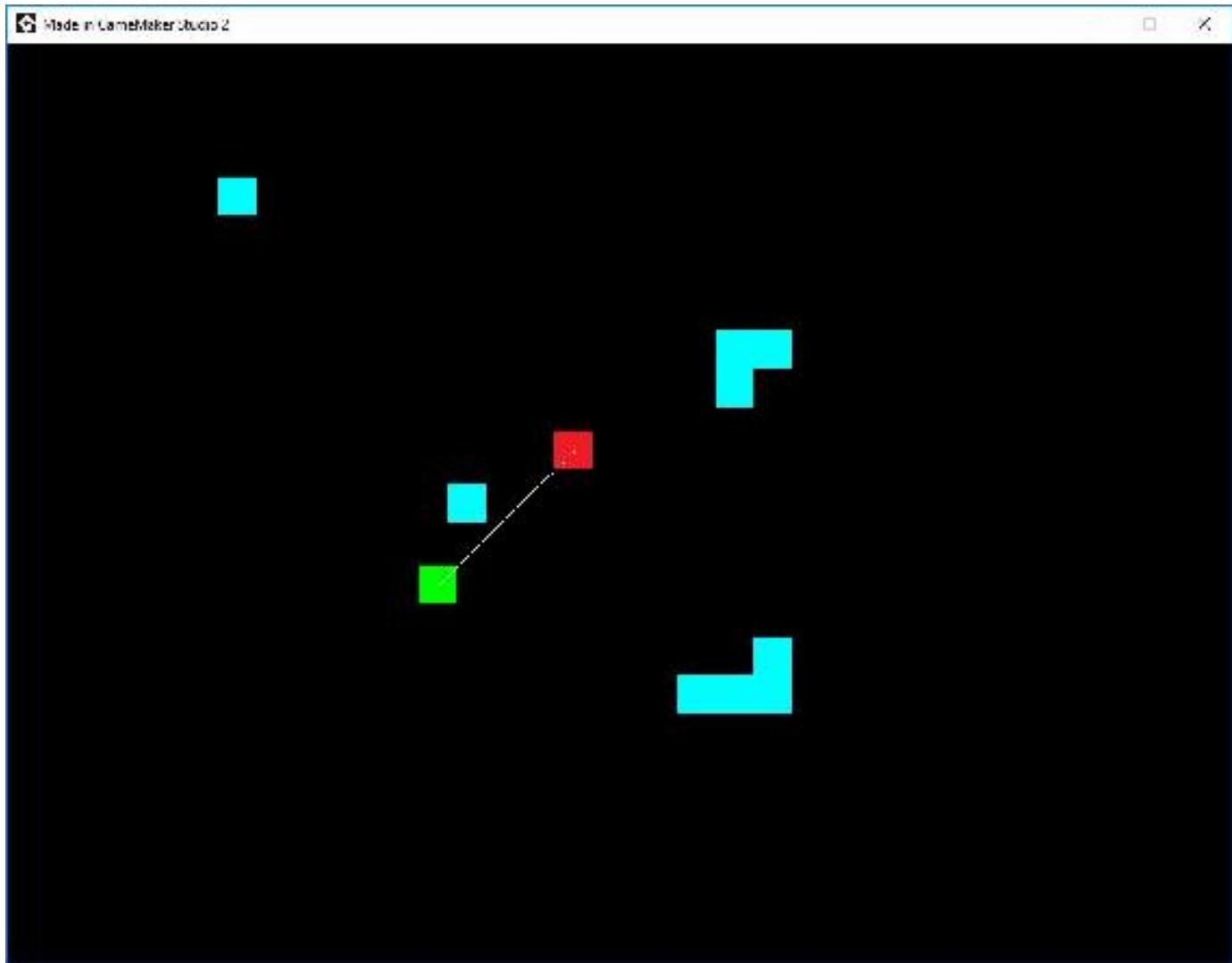


Figure A_17_2: Showing room being played

Load **snd_ouch** from the resources of this appendix.

Place the following in **obj_enemy** with a **Collision Event with obj_player**:

```
/// @description On collision
audio_play_sound(snd_ouch,1,false);
instance_destroy();
```

The above code will play the sound on collision with the player instance, then destroy itself.

Instead of playing a sound you could do things such as removing a life or making a graphical effect.

example_2 in the resources folder shows has the progress so far.

Next create an sprite **spr_bullet** which is 8x8 in size. Assign to an object **obj_bullet**.

Put this code in a **Collision Event with obj_player**:

```
// @description On collision  
audio_play_sound(snd_ouch,1,false);  
instance_destroy();
```

Change the **Step Event** for **obj_enemy** to:

```
//if there is a direct line of sight between the player and  
the enemy then set can_see to true, otherwise false  
if (collision_line(x, y, obj_player.x, obj_player.y,  
obj_wall, true,false))!=noone  
{  
    can_see=false;  
} else  
{  
    can_see=true;  
}  
if can_see  
{  
    mp_potential_step( obj_player.x, obj_player.y, 2,  
true);  
    timer++;  
} else  
{  
    timer=0;  
}  
if timer==10  
{
```

```

bullet=instance_create_layer(x,y,"Instances",obj_bullet);
    bullet.direction=point_direction(x,y,obj_player.x,obj_player.y);
    bullet.speed=5;
    timer=0;
}

```

This change will make the variable timer count up if it can see the player instance. If it reaches 10 it will create a bullet instance and reset the timer. If the player is hidden, the timer will be set to 0.

You can now save and test the game. An example project **example_3** is in the resources folder.

This next part continues on from the previous example; do not start a new project.

Create an object, **obj_food** and assign a 32x32 sprite to it.

In a **Collision Event** with **obj_player** put:

```

/// @description Jump to random position
score++;
x=irandom(800);
y=irandom(400);
while(!place_free(x,y))
{
    x=irandom(800);
    y=irandom(400);
}

```

Place 3 instances of **obj_food** in the room and save and test this. There is an example in folder, **example_3**.

If it collides with the player it will award a point and find a new free location.

Change the Collision Event code of **obj_bullet** with **obj_player** to: **/// @description On collision**

```
audio_play_sound(snd_ouch,1,false);
score--;
instance_destroy();
```

Finally create an object **obj_score**, and place the following in the **Draw Event** (or **Draw GUI Event**):

```
draw_text(100,50,"Player:"+string(score));
```

Place one instance of this object in the room.

You can now save and test.

An example project **example_4** is in the resources folder.

Basic Projects

- A) Create a control object that moves randomly left and right across the top of the window, wrapping as needed. It creates a bomb every 5 seconds that then falls toward the bottom. Player gets a point for every one collected. Player can only move left and right at the bottom of the screen, without being able to leave the window. Use the control object to draw the score in the top left of the window, remembering to set a font, drawing style, and colour.
- B) Make a movable player object (using arrow keys) and a static enemy object. Make the enemy shoot at the player, getting faster as the player gets closer. Prevent the player from leaving the window. Ensure the bullet gets destroyed when leaving the room; use room_width & room_height for this.
(See distance_to_object in the help file).

Advanced Projects

- C) Create an enemy that changes direction every 5 seconds to move away from the player, and wraps around the room if it goes off of the edge.

Appendix 18 INI Files

With casual games that people may only play for a few minutes at a time, the ability to store the player's progress is a required feature. Initialization or INI files provide an easy way to save and load data.

With INI files you can pretty much save any data, but the main ones you're most likely to save are these:

- Player's name
- Health
- Score
- Lives
- High Score
- Distance traveled
- Enemies killed
- Player cash

INI files involve two main elements:

[section] and **key=data**

A simple ini file would look like this:

```
[player]
top_score=100
name="Bob"
```

You can save this as an INI file (use a text editor and save as *savedata.ini*) and add it to the included files on the project tree. You can do this by right-clicking on **Included Files** and selecting **Create Included File**.

Note: On most exports it is not required to include an INI as an included file. This is not the case in HTML5, in which you must include one.

To load data from this file you can use:

```
ini_open("savedata.ini");
global.top_score=ini_read_real("player","top_score",0);
global.player_name=ini_read_string("player","name","");
ini_close();
```

What the above code does:

Looks for a file named *savedata.ini*

Sets the variable **global.top_score** to the value stored in the INI file. If the file or section and key do not exist it sets global.top_score to a default value, in this case 0 (the value after the last ,).

Sets the variable **global.player_name** to the value stored in the INI file. If the file or section and key do not exist it sets global.player_name to a default value, in this case "" (the value after the last ,).

It then closes the open INI file.

You can also write variables to an INI file. For example:

```
ini_open("savedata.ini");
ini_write_real("level_section", "level", global.level);
ini_write_real("distance_section", "distance",
global.distance);
ini_close();
```

This will store the current values of global.level and global.distance. If the values were 3 and 287, then the created INI file would look like this:

```
[level_section]
level=3
[distance_section]
distance=287
```

ini_read_real() and ini_write_real() work with real numbers. ini_read_string() and ini_write_string() work with strings.

Note: It is standard practice to close the INI file immediately after you have saved or loaded data.

Basic Projects

- A) Create two rooms, room_splash and room_game. Create an object for room_splash that loads any data from an INI file to two global variables and takes the player to room_game. If no INI file is present, set the starting location the value of each to 100. Make this object clickable to go to room_game. Create a movable object for room_game, which starts in the position stored in the INI file. Pressing X saves the location to the INI file and restarts the game.
- B) Create a counter that keeps track of how many keypresses of space the player makes in total, and save/load the value of this counter to keep track of presses over multiple games. Use a splash screen with an object to load from the INI file. Show the presses amount on screen. Allow pressing X to save value and restart.

Advanced Project

- C) Create an object that takes in five people's names, ages, and favorite food. Display this data on screen. When the game is restarted, and an INI file exists, give the option to display the current INI file or enter new data. Create a splash and main room as required. Allow pressing of X to restart. This example should use arrays, which haven't been taught yet; this project is for the more adventurous.

Appendix 19 Effects

Eye candy is quite important in modern graphical games. Your backgrounds, sprites, and buttons need to be of high quality in a finished game. You can also create an assortment of graphical effects using the built-in effects function. The advantage of using the inbuilt effects is they are easy to program and update if required. In its most basic form you can create an effect using (“ef” designates effect):

```
effect_create_above(ef_cloud, 100, 150, 2, c_green);
```

This would create an effect above the current object. The effect will be a cloud at position x 100 and y 150 in the colour green. The 2 denotes the size of the effect, which is large. You can also create an effect below the current object using

```
effect_create_below(type, x, y, size, colour);
```

You can try out these effects. Create an object, **obj_example**, and put this in the Step Event:

```
if (mouse_check_button(mb_left))
{
    effect_create_above(ef_cloud, mouse_x, mouse_y, 2,
c_green);
}
```

Create a room and place one instance of this of object in it. You can test the effect using the left mouse button.

You can use the following types of effects:

ef_cloud, ef_ellipse, ef_explosion, ef_firework, ef_flare, ef_rain, ef_ring, ef_smoke, ef_smokeup, ef_snow, ef_spark, and ef_star.

You can introduce some randomness into it, for example, change the code to:

```
if (mouse_check_button_pressed(mb_left))
{
    effect_create_above(ef_firework, mouse_x, mouse_y, 2,
choose(c_green, c_red, c_white, c_yellow));
}
```

Now test again.

Try the following code:

```
if (mouse_check_button_pressed(mb_left))
{
    effect_create_above(choose(ef_smoke,ef_ring,
ef_explosion, ef_firework),mouse_x,
mouse_y, 2, choose(c_green,c_red,c_white,c_yellow));
}
```

You can also create a mouse trail using effects. Create a new object **obj_trail**, and put in the following code in the **Step Event**:

```
repeat (2)
{
    effect_create_above(ef_star ,mouse_x, mouse_y, 2,
choose(c_green,c_red,c_white,c_
yellow));
}
```

Now test the game again. You will see an effect like that shown in *Figure A_19_1* :



Figure A_19_1: Showing effect trail

Combining various effects at the same time can create some awesome graphics.

Note: Effect code does not need to be in a draw event, it can be in a step event, key press event, etc.

Basic Projects

- A) Allow the user to change the weather by pressing W. Change between a snow and rain effect.
- B) Create a firework display each time the mouse left button is pressed. Have fireworks going off in more than one place, different sizes and colours.

Advanced Projects

- C) Create a line of effects, 20 pixels apart, which start at the top of the screen and fall down to the bottom, then start from the top again.
- D) Create an effect that spreads out from a location when the mouse is clicked. Make the effect move out every 10 degrees from the starting point. Destroy any effects after 3 seconds.

Appendix 20 Loops

A common need in programming is to repeat the same code multiple times. This functionality is available through using loops. There are four main loop functions in the GameMaker Language: ***do***, ***while***, ***for***, and ***repeat***. Each of these has its own strengths and weaknesses.

Loops are great for:

- Process data in ds lists or arrays
- Performing the same action multiple times
- Performing a calculation until it is true
- Drawing data

A loop can result in many similar actions being executed in a very short notice.

Note: It is important to make certain that loops run the correct number of times and then stop. It is possible through a coding mistake to create an infinite loop that runs forever. This will cause the game to stop working. Care needs to be taken to avoid this from happening.

repeat Loop:

repeat is a very simple control structure, which will repeat an assigned action for a specified number of

times. The following code creates five enemies in a single step, all at random positions in the room.

```
repeat(5) //Repeat the following code block 5 times
{
    instance_create_layer(irandom(room_width),
    irandom(room_height) ,”Instances”, obj_enemy);
}
```

while Loop

The ***while*** loop will repeat an action as long as the expression assigned to it is true or until you call a break. For example the following checks for an empty space randomly, and when it finds a free space the loop stops.

```
while (!place_free(x, y))  
{  
    x = random(room_width);    y =  
    random(room_height);  
}
```

for Loop:

The ***for*** loop will execute different actions, and after each one it will check if its expression is true. If the expression is not true, the loop will end, and none of the following functions will be executed. Like before, we'll use it to create a 50-point array in a matter of milliseconds. In almost all cases, you'd want to use var i here, as the loop counter generally has no need to be used outside of the block scope.

```
for(var i=0; i<50; i++)  
{  
    array[i]=i;  
}
```

The first statement (*i*=0) initializes the loop and sets a starting value. After initialization, the loop will check if *i* is smaller than 50, and if it is, it will increase the value of *i* using the *i+=1;* statement; this can also be written as *i++;*. After each step, the array's length will increase by 1 and add *i* to that array's position.

*Note: Alternatively you can reduce a value by 1 using *i--;*.*

To iterate *n* times with a for loop:

If you start with *i*=0, check *i*<*n*.

If you start with *i*=1, check *i*<=*n*.

do Loop:

This will repeat until an expression is true. For example, the following will repeat until it finds an empty position:

```
do
{
    x = random(room_width);
    y = random(room_height);
}
until (place_free(x, y));
```

Not too hard to understand, right? It can be used in many ways and it's one of the most practical functions in the GameMaker Language, so don't forget about it.

*Note: The **do** loop will always execute at least one time, but in **for** or **while**, it can be skipped.*

Basic Projects

- A) Place an object in a room at a random position. Create another object that finds a random location within 100 pixels of the first object. Use a while loop for this.
- B) Add 100 random numbers between 1 and 100 to a ds_list, then sort them into order, highest value first. Display onscreen in 4 columns of 25. Use for loops for this.

Advanced Projects

- C) Create four random points in the room. Get an object to visit each point in order. Display a message when all the have been visited.
- D) Store the names of students in your class in an appropriate way. Display names in alphabetical order, one at a time on the screen for 5 seconds, with a gap with no name for 2 seconds.

Appendix 21 Arrays

Arrays are a useful way of storing data in an organized format. They allow similar information to be stored together. This allows for easy access, manipulation, and displaying of data.

You can store real numbers, integers, strings; and the indexes of sounds, sprites, and objects in a single array.

Some use applications for arrays include the following:

- Storing weapon information for multiweapon games
- Keeping lists of data
- Storing data in order created
- Storing facts and information

There are both one-dimensional arrays and two-dimensional arrays in GameMaker. An example of setting a 1D array:

```
name[0]= "Bob";  
name[1]= "Claire";  
name[2]= "Steve";  
name[3]= "Nigel";  
name[4]= "Sue";
```

A visualization of this would look something like this:

Location	Value
0	Bob
1	Claire
2	Steve
3	Nigel
4	Sue

The following would draw the string stored in the array "name" at index 2, "Steve," at position 100, 100.

draw_text(100, 100, name[2]);

An example of a 2D array, storing name, age, and country of residence.//name

```
info[0, 0]= "Bob";
info[0, 1]= "Claire";
info[0, 2]= "Steve";
info[0, 3]= "Nigel";
info[0, 4]= "Sue";
//ages
info[1, 0]=27;
info[1, 1]=19;
info[1, 2]=52;
info[1, 3]=40;
info[1, 4]=102;
//country
info[2, 0]= "America";
info[2, 1]= "Spain";
info[2, 2]= "Brazil";
info[2, 3]= "Canada";
info[2, 4]= "France";
```

A visualization of this in table form would look like this:

	0	1	2
0	Bob	27	America
1	Claire	19	Spain
2	Steve	52	Brazil
3	Nigel	40	Canada
4	Sue	102	France

So the value at cell **[1, 3]** would be **40**.

You can use and change the values of an array as you would with any variable.

Arrays come into their own when processing data, especially with 2D arrays, as each array entry can be different lengths. Put the code shown previously into the **Create Event** of an object.

For loops are great ways to sequentially process data. The example below will repeat i three times and each time repeat j five times each time for i; this is known as a nested loop. It will then draw the value of the array cell at that position.

In a **Draw Event** put:

```
for (i=0; i < 3; i++)
{
    for (j= 0; j < 5; j++)
    {
        draw_text(i*70, j*80, info[i, j]);
    }
}
```

This code will draw the values of information onscreen in a table format, as shown in *Figure A_21_1*:



Made in GameMaker Studio 2

Bob 27 America

Claire 19 Spain

Steve 52 Brazil

Nigel 40 Canada

Sue 102 France

Figure A_21_1: Showing output of data

An example YYZ is available in the resources.

You can add to an array variable just like a regular variable. For example:

```
info[1, 3]+=1;
```

Which would increase Nigel's age by 1. You can compare array values:

```
if (info[1, 0]>info[1, 1])
{
    text_to_show=(info[0, 0]+ " is older than "+info[0,
1]);
}
```

Which sets **text_to_show** to *Bob is older than Claire.*

That is all for this example.

An application of this in GameMaker Studio 2 as an example; arrays are very useful for holding data for multiple weapons, as shown:

<u>Weapon Name</u> <u>Strength</u>	<u>Cost</u>	<u>Current Ammo Sound</u> <u>Effect</u>	<u>Image</u>	<u>Bullet Object</u>
Gun	1	1 200	snd_gun spr_gun	obj_bullet_gun
Machine Gun	5	10 400	snd_mach_gun spr_mach_gun	obj_bullet_mach_gun
Rocket Grenade	250	300 8	snd_rocket spr_rocket	obj_bullet_rocket
Nuke	1000	5000 2	snd_nuke spr_nuke	obj_bullet_nuke

You can create a data array for the previous code using:

```
// @description Put data in array
global.cash=100000;
global.selected_weapon=0;
```

```
//declare array
```

```
//weapon name
weapon_info[0,0]="Gun";
weapon_info[0,1]="Machine Gun";
weapon_info[0,2]="Rocket Grenade";
weapon_info[0,3]="Nuke";
```

```
//weapon strength
weapon_info[1,0]=1;
weapon_info[1,1]=5;
weapon_info[1,2]=250;
weapon_info[1,3]=1000;
```

```
//weapon cost  
weapon_info[2,0]=1;  
weapon_info[2,1]=10;  
weapon_info[2,2]=300;  
weapon_info[2,3]=5000;
```

```
//weapon ammo  
weapon_info[3,0]=200;  
weapon_info[3,1]=400;  
weapon_info[3,2]=8;  
weapon_info[3,3]=2;
```

```
//weapon sound effect  
weapon_info[4,0]=snd_gun;  
weapon_info[4,1]=snd_mach_gun;  
weapon_info[4,2]=snd_rocket;  
weapon_info[4,3]=snd_nuke;
```

```
//weapon sprite  
weapon_info[5,0]=spr_gun;  
weapon_info[5,1]=spr_mach_gun;  
weapon_info[5,2]=spr_rocket;  
weapon_info[5,3]=spr_nuke;
```

```
//weapon bullet object  
weapon_info[6,0]=obj_bullet_gun;
```

```
weapon_info[6,1]=obj_bullet_mach_gun;
weapon_info[6,2]=obj_bullet_rocket;
weapon_info[6,3]=obj_bullet_nuke;
```

Create an object, **obj_example**, and place the above code in a **Create Event**.

The resources needed for this example are in the downloadable resources, **example_1**.

You can set up an easy weapon select system using the following in a **Step Event**.

```
/// @description control
if (keyboard_check_pressed(ord("0")))
{
    global.selected_weapon=0;
}
if (keyboard_check_pressed(ord("1")))
{
    global.selected_weapon=1;
}
if (keyboard_check_pressed(ord("2")))
{
    global.selected_weapon=2;
}
if (keyboard_check_pressed(ord("3")))
{
    global.selected_weapon=3;
}

if (mouse_check_button_pressed(mb_left)) &&
weapon_info[3, global.selected_weapon]>=1
{
```

```
    audio_play_sound(weapon_info[4,
global.selected_weapon], 10, false); //play firing sound
    weapon_info[3, global.selected_weapon] = 1; //reduce
ammo
    instance_create_layer(mouse_x,
mouse_y, "Instances", weapon_info[6,
global.selected_weapon = 0]); //create bullet
}
```

```
x = mouse_x;
y = mouse_y;
```

In a **Draw Event** put:

```
/// @description drawing
draw_sprite(weapon_info[5, global.selected_weapon], 0, x
, y);
//draw info
draw_text(10, 20, "Strength:
"+string(weapon_info[1, global.selected_weapon]));
draw_text(10, 40, "Cost:
"+string(weapon_info[2, global.selected_weapon]));
draw_text(10, 60, "Current Ammo:
"+string(weapon_info[3, global.selected_weapon]));
```

Which is easier to understand.

An example is available in the resources, example_2.

Basic Projects

- A) Create a 2D array with data relating to four students in your group. Include variables name, age, height, eye colour, and favourite food. Display this data onscreen.
- B) Make a 1D array with the values of 10 types of food. Display one at random each time the spacebar is pressed.

Advanced Projects

- C) Create an array to store the starting positions in a game of chess, use letters to represent each piece, that is, K for King, Q for Queen, N for knight, B for bishop, and C for castle; all other squares to have a value of 0. Use UPPER CASE for black and lowercase for white. Represent this board on the screen. Try to use one or more loops to populate the array. Draw a chessboard with the correct value in the middle of each square.

Appendix 22 DS Lists

ds Lists (Data Structure) are effectively one-dimensional arrays that allow you to add data sequentially. However, they are much more flexible than a one-dimensional array, and you can perform lots of cool functions with them. They are great for the following:

- Sorting data alphabetically
- An inventory system
- Shuffling items
- Card games
- Music management
- Message management
- Sort the contents
- Delete contents, which will shift the remaining contents
- Adding instance id's and processing them (for example by distance or hp)
- Queuing up variables, objects or sprites to process them in order
- Add content at the start, anywhere in the middle, or at the end, which again can shift contents
- Find where something occurs in the list

Note: ds_lists start with an index of 0, so if you wanted to insert at position 5, the actual index would be 4.

ds_lists are awesome for making an inventory system. You can easily add items and remove them. So in an RPG type game you may pick up a key, then remove it when you collide with a door. To start you need to declare a **ds_list**, for example:

example_list = ds_list_create();

Then add something to the list:

```
ds_list_add(example_list, "cheese");
ds_list_add(example_list, "bacon");
ds_list_add(example_list, "mushroom");
```

```
ds_list_add(example_list, "ham");  
ds_list_add(example_list, "tomato");
```

Note: it's also possible to add many values within one line of code.

*For example: ds_list_add(example_list, "cheese", "bacon",
"mushroom, "ham", "tomato");*

This would give the following result:

Index	value
0	"cheese"
1	"bacon"
2	"mushroom"
3	"ham"
4	"tomato"

You can sort the list:

```
ds_list_sort(example_list, true);
```

Where **true** will be ascending or **false** for descending. Strings are sorted alphabetically, lowest to highest for values.

This would then look like:

Index	value
0	"bacon"
1	"cheese"
2	"ham"
3	"mushroom"
4	"tomato"

You can remove a value:

```
ds_list_delete(example_list, 2);
```

Would remove the value at position 3 (index 2). The list will then look like:

Index	value
0	"bacon"
1	"cheese"

2	"mushroom"
3	"tomato"

You can insert a new value:

ds_list_insert(example_list, 1, "egg");

Would insert "egg" at position 2 (index 1). The list will then look like:

Index	value
0	"bacon"
1	"egg"
2	"cheese"
3	"mushroom"
4	"tomato"

After adding an element you may want to sort your list again.

Other things you may want to do, such as finding where in the list something appears:

position = ds_list_find_index(example_list, "cheese");

You can return a value to a variable, for example, the following, which would set word to "mushroom":

word=ds_list_find_value(example_list, 3);

You may need to find the size of a list, for example:

list_size=ds_list_size(example_list);

Which is great to use before you try and find a value at a position. You can shuffle (randomize) the values with:

ds_list_shuffle(example_list);

You can also use the accessor | that will allow you treat the **ds_list** as an array, for example, the following code would draw the value in index 3 at 100,100 on the screen:

draw_text(100,100,example_list[| 3]);

You can look up how to use the following in the manual:

ds_list_clear()

ds_list_empty()

ds_list_replace()

ds_list_copy()

ds_list_read()

ds_list_write()

When you have finished using a *ds_list* it's essential to destroy it. This helps prevent memory leaks. For example:

ds_list_destroy(example_list);

Basic Projects

- A) Create an inventory system for five objects. Allow keypress of X for user add an additional item, add it to the end of the list, then remove the top item. Draw this onscreen.
- B) Create a *ds_list* with five fruits. Player enters a fruit; if it matches a value in the *ds_list*, remove it, and tell player they made a correct guess. Player wins when all five fruits are guessed.
- C) Create a list with the names of students in the class. Sort them in ascending order and draw on the screen.

Advanced Project

- D) Add the names of all playing cards to a list. Shuffle them. Create four new *ds_lists* to represent player's hands. Deal and remove the top card from the main list and deal to each of 4 players until each has five cards. Draw the values of each player's hand on the screen. Represent value and suit like: AS, 9H, 2D etc.

Appendix 23 Paths

Sometimes you want to have repeated object motion similar to how code is repeated with scripts. Paths are a useful way to make objects move in a set predefined motion (they can also be created dynamically through code). You can create paths using the built-in path editor or using GML. This section covers both.

You can create a new path by right clicking where shown in *Figure A_23_1*:

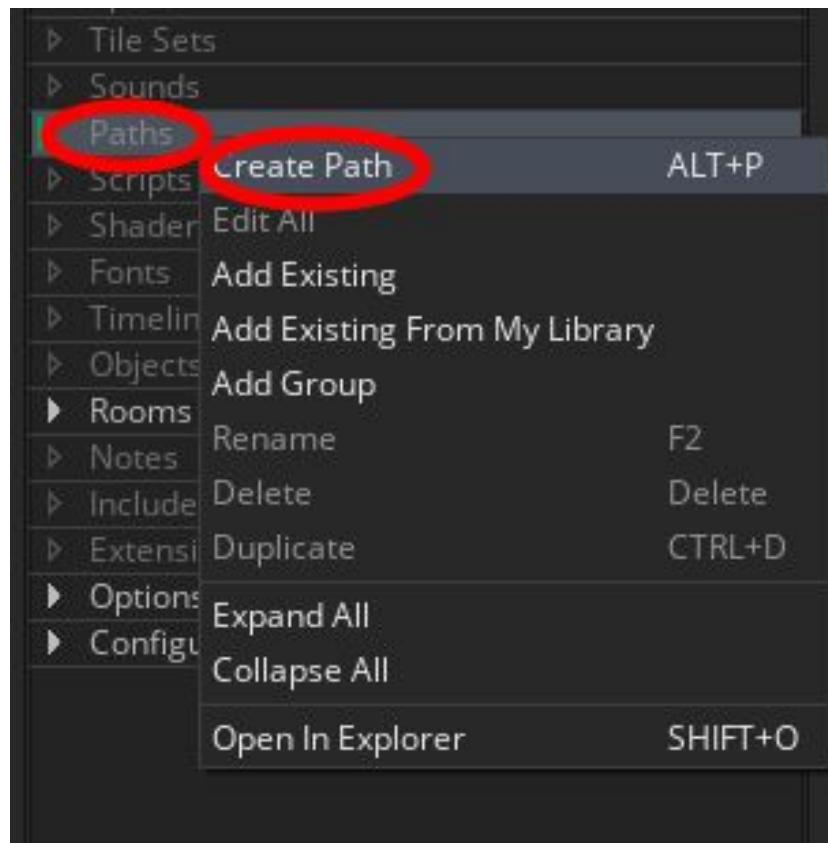


Figure A_23_1: Creating a new path

You can then create the path, and add points by clicking. *Figure A_23_2* shows a path with four points added, with **Closed** and **Smooth curve** checked. sp means the speed.

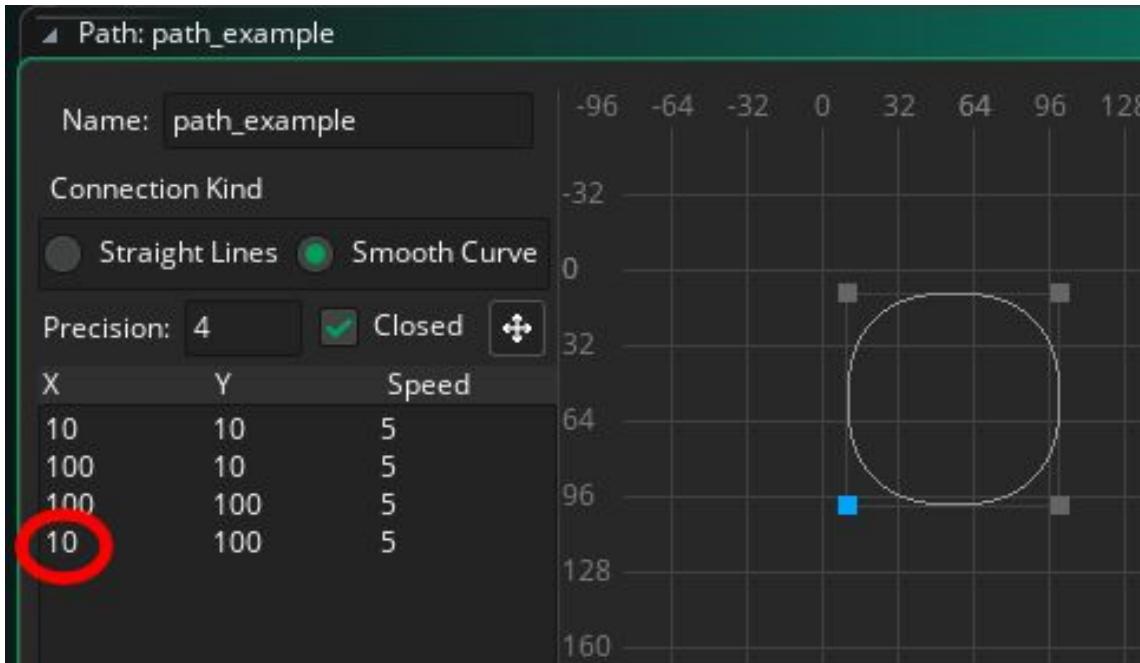


Figure A_23_2: Showing an example path closed and smooth curve

You can edit the location of a path point by right-clicking, for example where shown in the Figure above.

If you were to create the same path using GML, you would use the code below:

```
path_example=path_add();
path_add_point(path_example,10,10,5);
path_add_point(path_example,100,10,5);
path_add_point(path_example,100,100,5);
path_add_point(path_example,10,100,5);
```

This creates the path and stores it in instance scope, not global scope as the path editor would have.

This would create a memory leak if the instance is destroyed later without calling

path_delete().

The 5 at the end relates to the objects speed, though this is usually superseded when starting the path using code.

The above example could be used to make a sentry move around and protect a building.

You can then set whether the path is straight or curved: use **0** for straight, **1** for curved. So in this example we'll use:

path_set_kind(path_example, 1);

Then set that the path is closed with:

path_set_closed(path_example, true);

Then start the path with:

path_start(path_example, 50, path_action_restart, false);

path_example is the name of the path you created in the editor.

50 relates to the speed. **path_action_restart** tells what should happen when the last path location (end) is reached. The last part can be **true** or **false**. **false** places the path at the absolute position within the room (i.e., where you defined it in the path editor) and **true** positions it relative to the instance (i.e., the start position will be at the current instance x/y position).

There are several end path actions. They are:

Action name	Path Action Value	What It Does
path_action_stop	0	Stops at the end of the path
path_action_restart	1	Restarts path from first position
path_action_continue	2	Continues the current path
path_action_reverse	3	Reverses along current path

A neat feature that should be pointed out is that you can preview and change a path in the room editor, you can To use this feature, click as shown in *Figure A_23_3*. This would allow you to create a path around any instances that you have placed in the room.

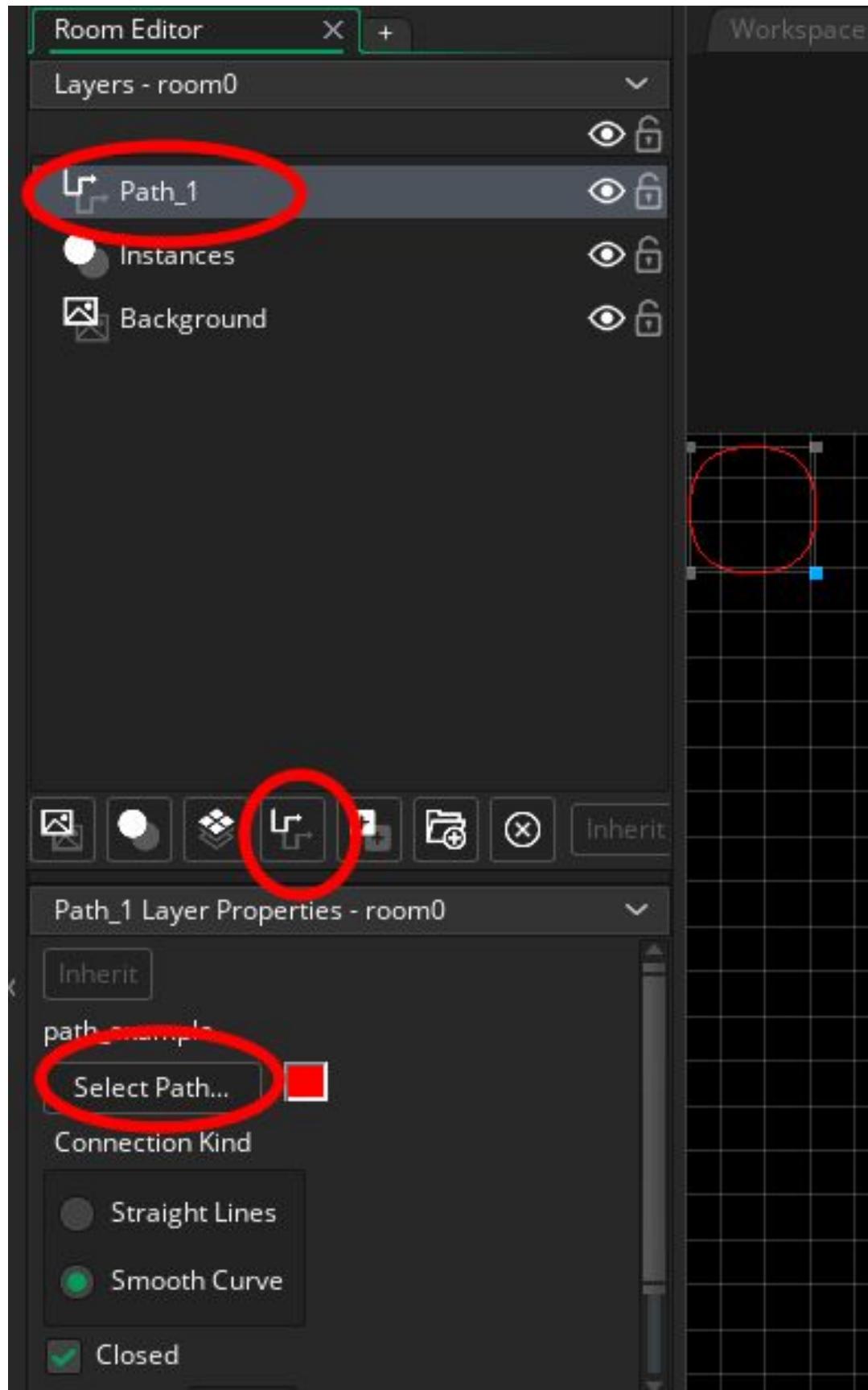


Figure A_23_3: Setting a path layer in the room editor
Some other useful functions for dealing with paths include the following:

Note: These should not be used for paths currently being followed.

You can insert a new point in a path using the following, which would add a new point to the path **path_example** at position 3 with the X and Y at with a speed of 5:

path_insert_point(path_example,3, 50,50, 5);

You can change a point's location, or speed using the following, which would change the path's point at position 4 to an X and Y of 25 with a speed of 5:

path_change_point(path_example, 4, 25, 25, 5);

You can get the X or Y point of a path's position using, for example, the following, which give the point of X location at position 2:

xpoint=path_get_point_x(path_example, 2);

You can set how precise a curved path is using **1**(low) to **8**(high):

path_set_precision(path_example, 4);

You can draw the path on screen, using for example (which must be within a **draw event**) which is very useful for debugging and testing:

draw_path(path_example,x,y,true);

Note: Set the drawing colour before using this code, to make the drawn path stand out.

direction holds the value in degrees of the direction an instance is moving along a path.

Basic Projects

- A) Create a path for an object, and make it move continuously in a circle. 1 Point
- B) When the player left-clicks the mouse add the location as a new point on the path. Draw the path on screen.

Advanced Projects

- C) Make an object point in the direction of movement when following a path. Use project 23 B as a basis.
- D) Create a random path of 10 points, and save the points to an INI file.

Appendix 24 Scripts

Reusable code makes it easy to update program objects at the same time. Additionally, it makes it easier to organize and understand how your code works. GML scripts are useful in processing data, especially if you will be doing the same calculation again and again. This can include sending data to the script and then returning a value if required. If you are using the same code twice or more anywhere in your program, then you should consider using a script. This allows you to make just one change to update your code. Imagine a game that had over 100 enemy monsters with their own code; changing the code for each would take many hours, and be prone to errors. Using a script you could do this in a few minutes. It also allows for nice and tidy code. Scripts also allow you to easily share code between different game projects (which is must if you go on to a career in game making) – saving you potentially a lot of time.

Some examples for scripts:

- Doing a math calculation and returning the answer, even if only used once; it means that your code is easier to read through and understand.
- Setting a drawing or font type, formatting, and colour – makes code easier to read.
- Playing sound effects and voices – you can send through which asset to play.
- Sending through an object and returning the closest instance – great for complex weapon systems.
- Drawing code that's used multiple times – allowing you to quickly update it.
- Recording bullet hits against multiple different objects.
- Adding things to a DS list

Any other GML that's used more than once.

You can create a new script by right clicking where shown in *Figure A_24_1* :

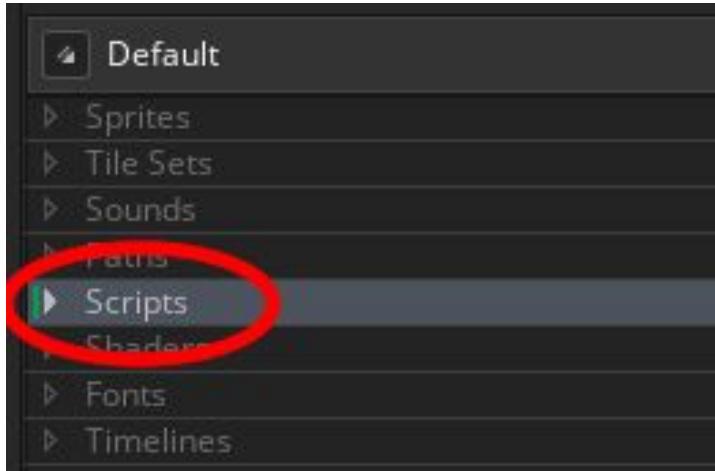


Figure A_24_1: Creating a new script

Name the script `scr_example`. Put in the following code:

```
/// @function scr_add(value,value,value)
/// @param {real} value
/// @param {real} value
/// @param {real} value
return argument0+argument1+argument2;
```

The comments in the above script perform an important function, as you type the script name when calling it, it will partially auto complete, reminding you what variables are needed.

Create an object `obj_example`.

In its **Create Event** put (so we don't try and draw a nonexistent variable):

```
answer=scr_example(12,18,7);
show_message(answer);
```

This will send the values **12**, **18**, and **7** to the script. The script will perform its GML and then return the value of total.

You can return strings, integers, real numbers, Boolean (true or false) and indexes of sounds, objects, rooms etc.

For example you can send through some objects: Create a script and name it `scr_pos`:

```
/// scr_pos(obj1, obj2)
//compares heights on y axis
```

```
if (instance_exists(argument0) &&
instance_exists(argument1))
{
    if argument0.y < argument1.y
    {
        return true;
    }
}
return false;
```

This will return **true** if **obj_1** is higher up the room than **obj_2**, by checking the Y location of each, or false otherwise.

Note that when you call **return** you are exiting the script, and no further code will be run from it.

You do not have to return a value, as shown in the following example. Another example, this script will use views to keep two objects in view, and set the border size. It takes the X and Y locations of two objects and adjusts the view so that both can be seen at the same time.

You can send through positions of an instance or of the mouse, for example, without returning. The following would draw a laser between two endpoints.

```
scr_laser(start_x,start_y,end_x,end_y)

draw_set_color(make_color_rgb(irandom(255),irandom(255),irandom(255)));
draw_line_width(argument0, argument1, argument2,
argument3, 5);
draw_set_color(c_lime);
draw_line(argument0+1,argument1+1,argument2,argu
ment3); draw_line(argument0+1,argument1-
1,argument2,argument3);
draw_line(argument0-
1,argument1+1,argument2,argument3);
```

```
draw_line(argument0-1,argument1-1,argument2,argument3);  
draw_line(argument0,argument1,argument2,argument3); effect_create_above(ef_spark, argument2, argument3,  
1, choose(c_red,c_orange));
```

An example of using this script, which would draw a laser effect from the object to the mouse position:

```
scr_laser(x, y, mouse_x, mouse_y);
```

Arguments can be accessed in two ways, for example:

```
name=argument0;
```

Or

```
name=argument[0];
```

Note: You cannot mix the two argument formats shown above.

Basic Projects

Create a script to do each of following, and display any result onscreen visually, as required, remembering to set up any text drawing.

- A) Find the average of five numerical values and round to the nearest whole number. 1 Point
- B) Work out if the player is within 200 pixels of an enemy object, and return true or false.
1 Point
- C) Draw given text, in white with a red shadow, at given position. 1 Point
- D) Create three different fonts. Create a script that allows you to quickly draw text, using font, alignment, colour, and position.

Advanced Projects

- E) Create a script that finds an average point between two objects, and draws a star effect at that position.
- F) Create a script that takes and calculates an angle between two objects and draws that direction as text as the angle were on a compass needle, that is, North or South West. The direction is that from the first object to the second. North is up.