TECHNOLOGY IN ACTION™

Learn RPGs in GameMaker: Studio

Build and Design Role Playing Games

—

Ben Tyers

Apress®

Learn RPGs in GameMaker: Studio

Build and Design Role Playing Games

Ben Tyers

Apress®

Learn RPGs in GameMaker: Studio: Build and Design Role Playing Games

Ben Tyers
Worthing, West Sussex, United Kingdom

ISBN-13 (pbk): 978-1-4842-2945-3 ISBN-13 (electronic): 978-1-4842-2946-0
DOI 10.1007/978-1-4842-2946-0

Library of Congress Control Number: 2017950737

Cover image by www.gamedeveloperstudio.com

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Technical Reviewer: Dickson Law
Coordinating Editor: Mark Powers
Copy Editor: Mary Behr

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

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

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

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484229453. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

Contents at a Glance

Contents

About the Author

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

About the Technical Reviewer

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

Acknowledgments

Special thanks to

Doug Morrison
&
Wayne Pinnow
&
Vadim
"YellowAfterlife"
Dyachenko

Thanks to the following for allowing use of their assets for the projects in this book

- Main Artwork: GameDeveloperStudio.Com/ Robert Brooks
- Additional Objects: KennyLand
- Main Character: Spyros Kontis
- Zombie Pirates, Card Characters, Main Character: gamedevmarket.net
- Cutscene Backgrounds: Purchased on 123rf.com - Klara Viskova/Roland Warmbier/Andrei Krauchuk/sergeiminsk/pixxart/Noel Powell/neyro2008/Sergey Breev/Sergey Pigulka/alexmstudio/Roman Dekan
- Ship: millionthvector.blogspot.co.uk
- Human Footsteps Sounds: OwlStorm Yoyodaman234 rocktopus punpcklbw Mikaelfernstrom - FreeSound.org, Creative Commons Attribution
- Fire Audio: midimagician - FreeSound.org
- Water Audio: InspectorJ - FreeSound.org
- Butterfly: GlitchTheGame.Com
- Wooden Pirate Ship: Bleed/OpenGameArt.org, Attribution 3.0 Unported (CC BY 3.0)
- Dice: JamesWhite/OpenGameArt.org, CC0 - Public Domain
- Tree: domsson/OpenGameArt.org
- Appo Paint Font: Grafito Design/Raul Perez

CHAPTER 1

Introduction

Some casual-style games may require a fair amount of design time.

RPGs, on the other hand, require copious planning and design. It is advisable to plan out the game in as much detail as possible before starting to code your game.

Designing and programming are two different disciplines. Constantly switching between them can reduce focus and creativity, and can also be mentally draining.

Planning is good for the following reasons:

- Allows you to brainstorm without having to think about programming
- Allows you to focus your creativity and get into the zone
- Provides notes that can be used later when coding
- Can actually be a fun activity in the overall process of design and programming
- Means you don't have to mentally store every idea (and risk forgetting the good ones)
- If you're working as a team, it allows you to share ideas with other team members

Sketching out your ideas on paper can also be beneficial. It allows you to have:

- Some time away from coding on your computer
- A break from having to think about your game
- The option of reviewing your ideas

At any point in the planning process, feel free to throw your notes into the nearest garbage bin, whether one page or all of them. You should think of your notes as a kind of organic life form that can change and adapt over time.

I recommend using one page for each section/element of your game, which makes changing/updating the ideas a slightly less painful affair when you do decide to make changes.

The rest of this introduction covers the main 20 points to consider when making an RPG, and the basic game template that some of the elements use as a template.

The main body of the book covers 40 commonly used RPG elements. I've done it this way so you can clearly understand what is relevant to each element. Some of these elements may be used as a template for other projects; this is indicated at the start of each element. The same game theme will be used throughout.

Hopefully, having each element separate should make it easy to understand what code is relevant to that section.

B. Tyers, *Learn RPGs in GameMaker: Studio*, DOI 10.1007/978-1-4842-2946-0_1

Programming/Source Code Notes

You may reuse the code in your projects.

The code used in this book is made for someone who is relatively new to Game Maker Language (GML). It is assumed that you have already read and learned what is taught in the book *Practical GameMaker: Studio Language Projects* or already have a basic knowledge of GML.

When coding, there is usually more than one method, using different GML or different approaches. The code used in this book is designed so that

- Someone relatively new to GML can understand it
- It's clear and logical
- Otherwise complex code appears clear and easy to read
- It's broken into understandable bite-size blocks
- You can easily reuse it in your projects
- It doesn't take a PhD to understand
- It looks good in print

Coding comments are used to explain what each section does, allowing a newbie to grasp the basics and understand what is going on

The source code is accessible via a download link at www.apress.com/9781484229453.

Story – Plot

Plot: The story of the game

Most RPGs follow some sort of story.

The premise for this one is that a character who lives in a windmill goes to the beach on a hot and sunny day, hires a boat, and gets caught in a storm. The boat sinks and she wakes up on an island run by pirates. She then needs to raise enough cash to get passage off the island.

See the chapter on cutscenes where this premise is made into an animated introduction story.

The story should then continue as the player proceeds in the game. This can be achieved through the following:

- Additional cutscenes
- Dialogue with other game characters
- The collection/use/interaction of objects
- The addition of mini-quests/mini-games
- Changing environments
- Unlockable locations

Locations that the player can visit can help to form part of the plot. When visiting new locations you can gradually provide more info for the player that helps progress the plot, but bear in mind that not all locations need to perform this action.

Having a good story not only engages the player, it sets the scene and gives the player some idea what they need to do. In a pirate-themed RPG, you know you will be treasure hunting and fighting pirates. You're unlikely to be dog-fighting an F-24 attack aircraft.

Most RPGs follow a plot throughout the whole game, for the simple reason that it works and is logical.

Story – Character Design

Character Design: Design of characters used by the story

Character design includes the main characters that the player will interact with; for example, in this story they are a shop-keeper, the ship owner, villains, and allies.

Your main character requires a bit of thought. Whoever plays your game will be spending a lot of time with them. An equal amount of thought should be given to the other main characters in the game. Spending too much time developing your main character can be counterproductive and may unbalance the game.

Consider things such as

- Appearance
- Voice (if they speak, which they probably will in an RPG)
- Quirks: Special things that make your character unique (in the example in this book, there are multiple voice variations when colliding with other objects, or after being static for any length of time)
- Movement: How the player is moved, and whether in four or eight directions or in 360 degrees
- Animation: Pre-rendered, made from separate parts, 3D rendered in real time, or just regular sprites
- Customizable Clothing: Via in-game upgrades or purchasable items
- What Type of Clan: Wizard, human, alien, zombie
- Special Skills: Flying, destroying walls, etc.
- How Much They Can Carry: Three items? Or eight if they find a bag?

At this point you should really decide on your art options. You can

1. Design and draw everything yourself
2. Commission an artist to do it
3. Use free/royalty-based art from the Internet

■ **Note** Free resources may only be free to use for personal projects. Commercial use may not be permitted under the terms of use. If unsure, it is advisable to check the license or seek permission directly from the creator.

Story – Enemy Design and Minions

Enemy Design: Mainly the design of non-character enemies and "nameless minions"

It is important that the main enemies are unique and have their own traits, such as a shopkeeper who stutters or a villain who chuckles.

Other characters in the game will also require some thought. There are two main types of characters: those that progress the story and plot, and those that don't.

For this example, the main enemy character will be a blue-bearded pirate.

The subcharacters will include a parrot, a pirate with a big nose, a pirate with a dagger, and a pirate with two daggers.

As for the main character, quirks are important. They could be

- A blue beard
- A wooden leg
- A hook
- Random voices and sounds

The parrot could fly randomly around the level. Upon collision with the parrot, the player might have to play a mini-game and win before proceeding.

Equally, there could be mini-games/challenges for the other characters, perhaps with the compensation of some gold coins. Limiting how often a player can play mini-games would be a good idea here.

Additional characters can help develop the plot or just be there for fighting, but the focus should be on the main game characters.

The game should ideally have several boss characters that the player must defeat before being able to proceed. Boss characters are usually strong, dangerous, and sometimes accompanied by nameless minions that also attack. The player should have to learn how the boss moves, where to shoot it, when to shoot it, and how to avoid its arsenal and minions. If it takes fewer than five attempts to learn the boss's movement and defeat it, it isn't tough enough. Make it harder.

As mentioned already, keep to the theme.

Story – Objectives

Objectives: The goals the player must complete to advance the story

The main objective of this game is to collect 1,000 gold coins so the main character can get off the island.

Providing a variety of ways to collect gold coins is the key here, such as

- Completing mini-games/quests
- Winning battles/fights
- Finding secret areas
- Small puzzles
- Digging for buried treasure

As before, restricting how often/quickly a player can get gold coins is the key here. For example, you may want to limit battles to once every 20 minutes the player plays the game, or charge the player one gold coin each time they dig (which prevents them trying to dig over the whole level).

Each enemy should have a different battle game with its own mechanics, to create variety.

Mini-quests could include retrieving 10 mushrooms in the level in five minutes or climbing a mountain to retrieve a diamond. The more quests you have, the more variety the player will have when playing your game.

Quests and mini-games can be part of the overall story of your game or independent. Some may require the player to successfully complete them in order to proceed: for example, killing a dragon to get the key that's around its neck, which can then be used to unlock a gate so the player has access to a new game area. A good mix of the two usually works well.

Even for mini-games that aren't part of the story, it's generally a good idea to keep to the theme of the main game. This game is based around a pirates' island, so things involving boats, ropes, sword fights, digging, map reading, exploring the island, building a sandcastle, shooting parrots, and that kind of stuff suit it well.

Story – Setting

Setting: Includes the general theme for graphics

The main theme will pretty much dictate all the graphics and audio used in your game (except for maybe a few mini-games or quests).

For the example in this book, the theme is pirate-based. Backgrounds, objects (both static and interactive), and audio should reflect and reinforce this theme. As the game is based on an isolated island, graphics and locations should reflect that too.

The following objects would be suited for such a pirate theme:

- Anchors
- Old wooden huts
- Wooden signs
- Barrels
- Palm trees
- Flying birds

Sound effects will all help create an immersive environment for the player, such as

- Birds
- Waves from the sea
- Footstep sounds on different surfaces

Themed locations could include

- The beach
- Forests
- Underwater sections
- Mini-platform game on an abandoned ship

Aesthetics – Art Style

Art Style: The style of art the game uses

One of the most important factors to decide upon is the art style of your game. There are tens if not hundreds of art styles and subgenres. The main four are pixel, vector, 3D, and photorealistic. Each has its own positives and negatives, as shown in Table 1-1.

Table 1-1. *The Main Four Types of Art Styles*

	Pixel	Vector	3D/Pre-Rendered	Photorealistic
Positives	Cheap to hire an artist. Lots of resources available for free or low cost. Short learning curve if you go at it alone.	Plenty of talented artists available. Can be adapted for size easily. Huge library online, both free and licensed. Can be pre-rendered to other formats.	Looks cool. Can be pre-rendered so it's quick. Lots of artists offering services. Multiple platforms use 3D.	Awesome eye-candy if done well. Limited by imagination only. Can create a truly immersive experience.
Negatives	If not integrated well, it can make a game look poor. Limited to how much information or detail you can portray. Not suitable for large sprites or backgrounds.	A steep learning curve on making vector graphics. Limited detail. Can be slower to process in real time. Using vectors in GameMaker requires some skill.	Extreme learning curve for modelling and importing into GameMaker. Expensive to hire artists. Minor changes may take days to get new sprites.	Can be extremely expensive. Not really suitable for an indie game developer. Huge game sizes. Requires special graphics cards.

Aesthetics – Choice of Art Style

Choice of Art Style: Must consider how the player sprite is going to be drawn (using either single or multiple layered sprites)

For this project, we're going for vector-style art.

The big question is how to implement it. The two main choices are a single sprite for each frame of animation, or creating the animation using separate parts.

Each has its own benefits and drawbacks.

If your art skills are somewhat limited, then outsourcing your artwork or obtaining pre-rendered sprites is the way to go. This may limit what your characters can do, but it is (usually) a cheap and effective method.

Using separate sprites allows for greater flexibility in what your characters can do, with movement limited only by your imagination. The drawback of this approach is that it requires a good understanding of math and learning other software such as Spine in order to do this in real time in the game.

For single sprites, there are a few options, such as buying custom-made or off-the-shelf sprites, or using specialist sprite creation software. If you choose the latter option, I'd go for one that supports skeletons. Sprite software is the middle road here: the basics can be learned within a day and the results can be visually awesome and very rewarding, but may lack the overall quality you get from hiring a professional or using premade sprites. Another advantage of going it alone is you can create your own animated sequences, rather than just the common walk, run, idle, and jump. If you want your character to do a somersault, you can make that happen. If you want it to rub its belly after eating food, you can also make that happen.

If you want something quickly for a mock-up of a game, then premade sprite packs are the obvious choice. You can always change the sprites to something more suited to your game dynamics as you proceed through your project.

Generating animations in real time has its advantages too. But imagine a highly animated game with 20 characters, each with 20 different animations with 20 frames each; that's 8,000 frames! At a decent quality that could be around 800MB. This is way too much for tablet- or phone-based games. Using skeletal animation could reduce that to around 8MB, which is a much better option.

It all boils down to the target platform and how much time you are prepared to put in to learn the required skills.

Aesthetics – Scening

Scening: How the story progression is going to be implemented in the game (usually via the use of cutscenes)

There are various styles of cutscenes. You can get a breakdown at https://en.wikipedia.org/wiki/Cutscene.

Cutscenes are a great way to provide info to the player. In a RPG it's now expected that there will be several cutscenes. Generally speaking, they are non-interactive animated sequences.

The game for this book will have an opening cutscene that explains how the main character ended up on the pirates' island. (See the chapter on cutscenes for a working example.)

Additional cutscenes could be added to provide extra information or to explain quests/tasks and mini-games.

Cutscenes are also commonly shown at the end of the game, as a kind of a reward to the player for completing it. Sometimes there is more than one possible end scene, depending on decisions made by the player in the game.

In the game for this book, the main task is for the player to acquire enough cash to get passage off the island. A great time to show a cutscene would be when the player finds the buried treasure chest.

Other major events could also warrant a cutscene, such as

- Killing a boss enemy
- Finding their first gold coin
- Visiting the in-game shop
- Conversations between a player and other characters
- A way of explaining complex instructions or tasks

A note of caution: not everyone is a fan of cutscenes. Overdoing them can annoy the heck out of people. You may wish to give players the choice of being able to skip cutscenes.

That said, when done well and at the right times, cutscenes can add a lot to your games.

Aesthetics – Sound Design

Sound Design: Which basic sound effects the game will need (for example, footsteps can be used for a more serious tone, etc.)

I have a great idea! Let's make sound effects for everything:

- Different footsteps on different surfaces (like gravel, grass, water, leaves, snow)
- Make a sound when a player picks up/puts down an object
- Ker-ching noises when collecting coins
- Alarms, bells, and whistles when finding a treasure chest
- Squawking noises when the parrot flies overhead
- Lots of voice instructions
- Voices in dialogues
- Repetitive background music on loop

Repetitive multiple sound effects playing at the same time can be annoying. When used correctly, however, sound effects can add a lot to the game and provide audio feedback based on what is happening on screen. When computers first had the ability to make good quality sound effects, it was pretty cool. (I'm thinking back to Tomb Raider and Alone in the Dark). These days, not so much.

Overusing sound effects is an easy way to alienate your players. Try to use sound effects sparingly and for emphasis. For example, footsteps on grass could be quiet, or some water sounds would be OK.

Provide an options screen where the player can select/deselect/change the volume of various in-game audio effects. They will thank you for that.

When used well, in-game audio can provide an immersive experience for the player. The quality of your sounds and voices is also important. There are plenty of sites out there where you can hire professional voice-over (VO) artists at reasonable prices. If you decide to go the voice route yourself, invest in a quality microphone and DAW (digital audio workstation) software; you can get set up for less than $200 (£150).

Aesthetics – Projection

Projection: The angle from which the player sees the game world (first person, top down, third person, etc.)

Different game genres have different view styles: platform games are usually side-on views. RPGs can be top-down (4/4), isometric, various forms of 2.5D (also known as fake 3D, semi-top, or 3/4 perspective), or full 3D.

Generally speaking,

- Isometric games look pretty cool, but can be a pain to program.
- Top-down looks OK and is quite easy to program.
- Semi-top looks good, is easy to program, and allows you to provide more info than top-down.

As you can see from Figure 1-1, in the top-down view you can see the roof of a building, but you are unable to tell what type of building it is. The semi-top view shows part of the roof and the front of the building; this immediately allows the player to know what the building is. Hey, you could even hang a sign with the word "SHOP" on it. Portraying this information in top-down is trickier.

Isometric views can not only provide the information on the type of building, but also give a feeling of depth and, if done well, look pretty cool.

For the purpose of this book, the project will be using semi-top, for the following reasons:

- A lot of artwork in this style is available, both free and paid.
- Creating your own graphics in this style requires only a gentle learning curve.
- Movement and object interaction is fairly simple (when compared with isometric).
- The math level required for interactions and calculations is reasonably low.
- Low CPU overhead; hey, GameMaker Studio is great for 2D games.
- The player can easily understand what they see on-screen.
- No transparency effects needed (e.g. walking behind a building in isometric view).

Obviously there are also negatives in 3/4 view compared with isometric view, but 3/4 view is a great place to start.

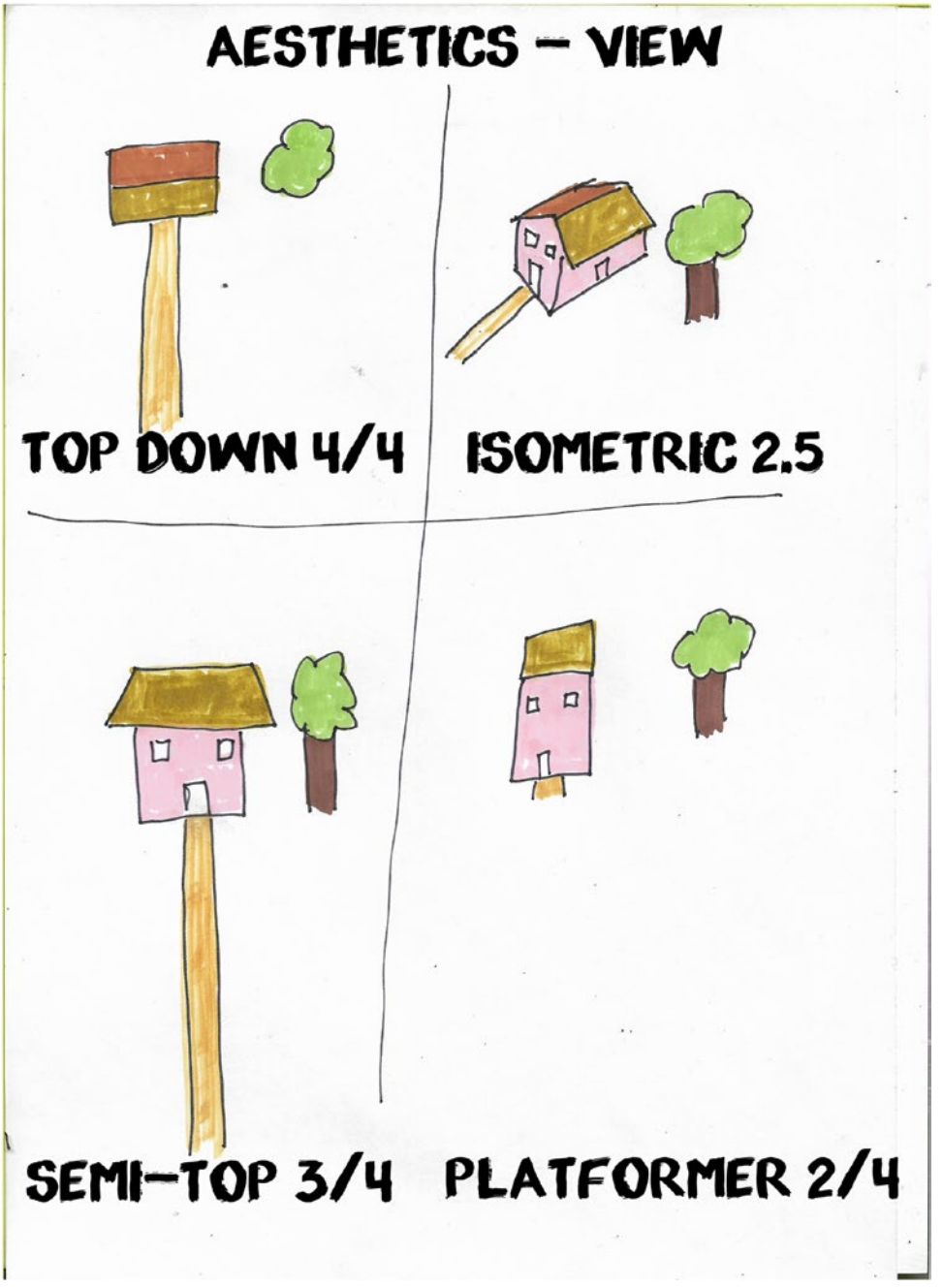

Figure 1-1. *Types of game views*

Core Game-Play – Battle

Battle: The main provider of the challenge in the game (for example, Pac-Man's battle aspect is the avoidance of the ghost creatures)

RPGs are synonymous with battles and fights, with dialogue-driven story lines currently very popular. They are usually a mix of different fighting/battle games; in my view the more variety, the better. The main ones used are

- Boss Battles: Which I won't discuss here because they were mentioned in a previous section
- Turn-Based: Characters take turns attacking each other
- One-On-One: Full-on fighting in real time, like old-school arcade games
- Random: Player just clicks and hopes
- Avoidance: Having to survive an amount of time or reach a goal without hitting an enemy
- JPRG (Japanese RPG): Turn-based variation
- Card-Based: Play a card and hope it beats the enemy's choice
- Hybrid: A combination of one or more of the above

Some general guidance for battle systems:

- Make use of the whole screen
- Use sounds and background music
- Make it clear what the player must do
- Make use of objects that the player has collected
- Have different enemies use different attack styles
- Make the characters face each other
- Allow a skilled player to always be triumphant

Core Game-Play – Ending

Ending: How the player can achieve Game Over (by dying, completing certain objectives, or finishing the story)

Personally, I think the main goal of an RPG should be clear from the onset. The initial cutscene is a great place to put this goal. In this game, this will be reinforced by placing the player near the exit on game start. The player can walk up to and chat with a pirate who will explain that they'll need 1,000 gold coins if they want passage on his boat. I don't feel this distracts from the game-play, as there will be many subplots for the player to uncover on their own.

Exactly how much information you give the player and when is up to you. You may, for example, just give info for the next quest/mini-game, and provide core information later on.

Whether your game takes hours, days, or weeks to complete, reward your player. They've committed their time and money (if they bought it), and they expect something in return. An immersive cutscene is a great way to end the game.

You may wish to throw in a false ending, such as if they've got their 1,000 gold coins, the pirate sails them home. On the way home, they come across another pirate ship and they have a full-on battle with cannon balls, explosions, etc.

Ending credits are also good: you can thank everyone who helped on your game, where you sourced your graphics and audio, and code you got from other places. You could go for simple scrolling text, or full graphics with effects, whichever suits your game style.

Depending on how you planned your game, it may have multiple endings. For example, rather than finding the 1,000 gold coins, your player got together the tools and resources to make a small rowboat. Different endings will get players coming back and playing it more often, which is great if you have an ad-based revenue system.

There is also a school of thought that games should not have multiple endings, as they may somehow punish the player for not doing something or not visiting a location in game.

Core Game-Play – Exploration

Exploration: How players will travel the game world (by exploration or level select screens)

So you've designed the main characters and the basics of your game play. Next is something that is often overlooked or not given enough consideration: how your player gets from A to B.

You can add a little variety without distracting from the fact that it's an RPG.

Mainly the character will be walking, whether on solid ground, grass, or through puddles.

You can also add variety by offering some alternatives.

- Parrot: You catch a parrot that will fly you to isolated locations not accessible by foot
- Swimming: The ability to cross a vast river to otherwise inaccessible locations
- Vehicle/Transport: Perhaps your player can collect items to make a boat or cart. Then they can use it in a mini-quest against another pirate
- Weapons: Starting off the game with a character with no weapons is also a good way to go. Perhaps when they have a sword/cutlass they can cut down bushes or trees to allow access to other areas
- Running: Perhaps for short time after eating food the player can run faster and jump farther, allowing the player to traverse ravines, for example
- Skiing: Perhaps the island has snow-covered mountain peaks. Skiing down them would make a great mini-game: one-on-one against a pirate

Keyboard movement in RPGs is generally done with WSAD (key presses of the keys W, S, A, and D). This is mainly so you can control the character on-screen and still be able to use a mouse; additionally arrow keys are sometimes available.

Core Game-Play – In-Game Text

In-Game Text: How players will receive information from the game (also dialogue)

Text messages are a core component of RPGs. For the purpose of the game made with this book we'll be using the following:

- Text on Scroll: Used during cutscenes to convey information
- Pop-Up Text Box: When conversing with other characters in the game

- Info Text: Kind of a quirk; semi-random sentences when you collide with objects in game. For example, one of ten sentences at random for each object.
- Hints/Tips: Pop-up messages that guide the player through the RPG/story or what they should do next

In RPGs, text is commonly shown in a typewriter-style effect, one letter at a time. For the purposes of this book, I'll be following that convention.

Text is also usually accompanied by spoken words. Each speaking character will have their "own" voice. This is something you can do yourself or outsource it. I recommend going it alone because it can be fun and rewarding. There's plenty of software out there to do this: as a starting point, I recommend Audacity, which is free.

Interactions with main characters should add something to the overall game, such as advancing the plot or giving tips to the player.

Core Game-Play – Scoring

Scoring: How the scoring system of the game will work (this is also used to plan for XP in RPG games)

One way of showing progress to a player in an RPG is their stats/scoring. For example,

- XP (experience points): This stat usually goes up when the player wins a fight or completes a task. It may also be split into levels
- Gold: In the game for this book, the aim is 1,000 gold pieces. Displaying how many the player has will give a good indication of their progress
- Health: As a constant side-plot, the player may need to be on the constant look-out for food or water; otherwise they may die
- Skills: The skills the player currently has, perhaps starting with four and adding extras as they progress through the game
- Time: An indication of how long a player has to complete the current task/challenge/mini-quest
- Mental Well-Being
- Strength, which may be reduced during physical exertion and replenished with rest
- Designing a layout that displays all of this information for a heads-up display (HUD) is a skill unto itself. It is dealt with in its own chapter

Other things that you may want to keep track of, perhaps as a pop-up so as not to overcrowd the HUD, are

- Total distance travelled
- Total gold coins collected/spent/used
- Total battles with some stats
- Total time played

Additional player attributes could include awareness, luck, ego, etc.

Extended Game-Play – Collectables

Collectables: These include secondary objectives that will be used to enhance the game's lifespan

What's a good RPG without collectables? Collectables make an RPG an RPG.

Whether for instant gratification, food or water, or for advancing the story, collectables are it.

For the game designed in this book, the main objects are

- Food and Water: Needs to be constantly sourced and has no effect on the overall story (unless the player dies and needs to restart)
- Gold: The main collectable for this game, the player needs enough gold to leave the island. Gold may also be spent in the shop, reduced if they lose battles, or spent if they want to dig
- Keys: Can unlock areas and help advance the story
- Shovel: Once the player has this item, they can start digging for the buried treasure chest. In this game, each dig will cost one gold coin to prevent cheating by digging everywhere
- Mushrooms: Used in a mini-quest where the player needs to find 10 mushrooms in a set time
- Battle Skills: Certain objects add battle skills
- Sword: Allows the player to participate in battles
- Satchel: A bag that, when collected, allows the player to carry more items
- Birdseed: Allows the player to trap a parrot for hidden areas

An RPG is likely to have tens if not hundreds of separate items. Take time to design/plan each item and how it works within the game.

Having notes to work from is an essential aspect of the design process and will allow for quicker and easier game development.

Extended Game-Play – Further Considerations

Further considerations: This includes the inventory, items, and power-ups that the player can use to increase game depth

The next stage is important. It's how you relate the game back to the player.

Inventory, as mentioned previously, may start with two slots and increase to four when the player finds a bag. Obviously you show two empty slots in the inventory section of your HUD. The other slots could have a big red X in them. On mouse over, you could display the text "Find Bag to Hold More Items," perhaps accompanied by spoken audio, but make sure not to use too much audio at once, or you may alienate the player.

For health, you could use some kind of bar or container that shows current health. In the book's example, we'll use a chemistry bottle with liquid that changes fullness based on health. The player can keep their health up with a constant supply of food and water. A visual bar for each would suffice. Rather than just having items scattered through the level, force the player to give some thought (i.e. running into a tree may drop a coconut, crouching next to a stream may give water).

Showing what battle skills the player has and needs to find can be shown graphically: full colour for current skills, greyscale for those yet to find.

One of the quirks for this game is lots of objects, some of which can be collected, but all of which have multiple sentences that are displayed and spoken when the player collides with them. Some of these sentences will provide hints and tips on how/where to use the item, and other sentences are just for fun and variety. Encouraging the player to collide multiple times is the main aim here.

When it comes to mini-games, variety is the key. Mini-puzzles could be based on anything: flying a parrot through a forest while avoiding branches, crossing a river without being eaten by a crocodile, running across the deck of a ship while avoiding sword fights. For this game, plan to use ancient weapons, like a bow and arrow or a trebuchet.

Extended Game-Play – Mini-Games

Mini-Games: Such as the lock-picking games that many games now use

Here's a rundown of a few mini-game ideas:

- Hunt for Mushrooms: Collect 20 mushrooms from a forest within a set time limit
- Destroy Buildings: Use a trebuchet to throw rocks at a building
- Swimming: Swim in the sea collecting seaweed, but avoiding fish
- Maze: Get out of a maze in a set time limit
- Lock-Picking: Pick a lock using the mathematical clues provided
- Matching Game: Match the cards
- Rowing Race: Race against a pirate over a water-based course
- Find The Treasure: A variation on the classic game Minesweeper
- Tap to Music: A variation on the tap-the-button-to-the-music style game
- Match Three: Swap over objects to get three in a row

As you can see, mini-games are really only limited by your imagination: as long as it has some similarity to the main game, you can't go too far wrong.

Although mini-games are usually distant from the main game story, they should reward the player. For this game, a few gold coins would be sufficient.

You may want to limit how often a player can play each mini-game.

Extended Game-Play – Quirks

Quirks: Unique or strange game-play mechanics that you want to use to make your game stand out from the others

Quirks are things that make your game special. In this game it's going to be the multiple variations in dialogue/text when speaking to in-game characters, colliding with various objects, or when purchasing from the shop. Additionally, sound effects and music will be used for emphasis.

There will also be diversity in the range of the quests and mini-games. I intend to make some so good and engaging that they could stand on their own as a separate game.

Other quirks could include

- Mundane objects as toys (would make good cutscenes)
- Dancing to music if not moved for a time
- Correcting a pirate's speech for not using proper English
- Burping after eating or drinking

Extended Game-Play – Saving

Saving: Saving and loading game files extends game life by allowing the player to return whenever they want.

Saving is an important aspect of RPGs and is relatively easy to do.

With modern devices, players may leave the game part the way through, for example to take a call or check emails. This makes timing when to save a bit of a hit-or-miss affair.

The approach I favor is big signs throughout the game with the words "Save Here" written on them. No mistaking that: on collision with the player, the save script is executed and a message is shown to the player that the game has been saved. Make it so this can only be done when not a mini-game or quest.

I prefer the use of an INI system to save data. In an RPG, this is relatively straightforward because there are only a limited number of things that need to be saved and a limited number of objects in-game.

There is a whole chapter dedicated to saving data in this book.

Basically for an RPG you'll need to store

- Player stats
- Location of objects
- Which mini-games/quests have been played and how often
- Location of player
- Location of other characters
- What items the player has in the inventory
- Location of other in-game objects

Game Base

To make things a little easier, and to prevent me showing the same code in every chapter, most of the RPG topics will use a basic movement code for the player. The GMZ project file for this is in the download resources and is named `Game_Base`. It's a very basic movement and animation example that will be expanded upon in other sections. For each project, it will be noted whether this or another GMZ file is used as the base template. You could use emums for directions and movement, but the method used allows for easier adaptation of the code in later elements where it is used as a base template. This base will provide four directional movements, controlled by arrow keys. If you prefer movement via WSAD, this is a simple change you can make yourself.

The name of the player object is **`obj_player`**. The **Create Event** code sets the initial values. It uses a number of variables so you know what direction it is/has been moving in and whether it is currently moving (see Listing 1-1).

Listing 1-1. Setting Up States and Initial Variables

```
///set up
enum player_state {
    idle,
    up,
    down,
    left,
    right
    }

dir= player_state.down;
is_moving=false;
image_speed=0.5;
```

The **Step Event** consists of three blocks. The first block detects keypresses and updates the values accordingly (see Listing 1-2).

Listing 1-2. Changing States Based on Keypresses

```
///keypress code
if (keyboard_check(vk_left))
{
    dir= player_state.left;
    is_moving=true;
}
else
if (keyboard_check(vk_right))
{
    dir=player_state.right;
    is_moving=true;
}
else
if (keyboard_check(vk_up))
{
    dir=player_state.up;
    is_moving=true;
}
else
if (keyboard_check(vk_down))
{
    dir=player_state.down;
    is_moving=true;
}
else
{
    is_moving=false;
}
```

The second block will make the player move if the flag `is_moving` is `true` and `dir` has a value (see Listing 1-3).

Listing 1-3. Moving the Object Based on Its State

```
///movement
if is_moving
{
    switch (dir)
    {
        case player_state.up:
        y -= 4;
        break;

        case player_state.down:
        y += 4;
        break;
```

```
        case player_state.left:
        x -= 4;
        break;

        case player_state.right:
        x += 4;
        break;
    }
}
```

The block used for animation checks whether the player is moving or not and sets the appropriate sprite (see Listing 1-4).

Listing 1-4. Setting the Appropriate Animation Based on Direction and Whether Moving or Not

```
///animation
if is_moving
{
    switch (dir)
    {
        case player_state.up:
        sprite_index=spr_walk_up;
        break;

        case player_state.down:
        sprite_index=spr_walk_down;
        break;

        case player_state.left:
        sprite_index=spr_walk_left;
        break;

        case player_state.right:
        sprite_index=spr_walk_right;
        break;
    }
}
else
{
    switch (dir)
    {
        case player_state.up:
        sprite_index=spr_idle_up;
        break;

        case player_state.down:
        sprite_index=spr_idle_down;
        break;

        case player_state.left:
        sprite_index=spr_idle_left;
        break;
```

```
        case player_state.right:
        sprite_index=spr_idle_right;
        break;
    }
}
```

As you will see from the code, it will animate walking when moving, or show an idle animation if not walking.

There is also a set of four images for each direction, both idle and moving. You can find them in the folder path `Resources ➤ Images and Audio ➤ Game Base`. The origin for the sprites is 32,60. They can be created by clicking the Create Sprite button, as shown in Figure 1-2.

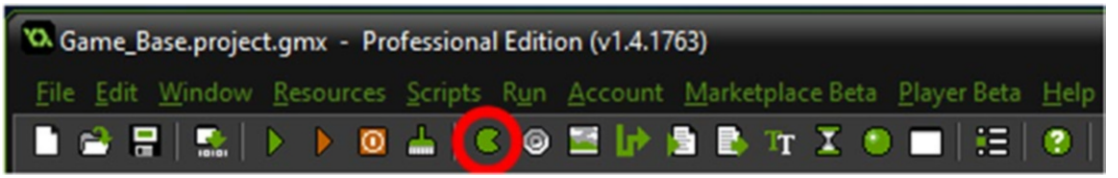

Figure 1-2. *Creating a sprite*

Next, name your sprite **`spr_idle_left`**, and click Edit Sprite ➤ File ➤ Create from file, as shown in Figure 1-3.

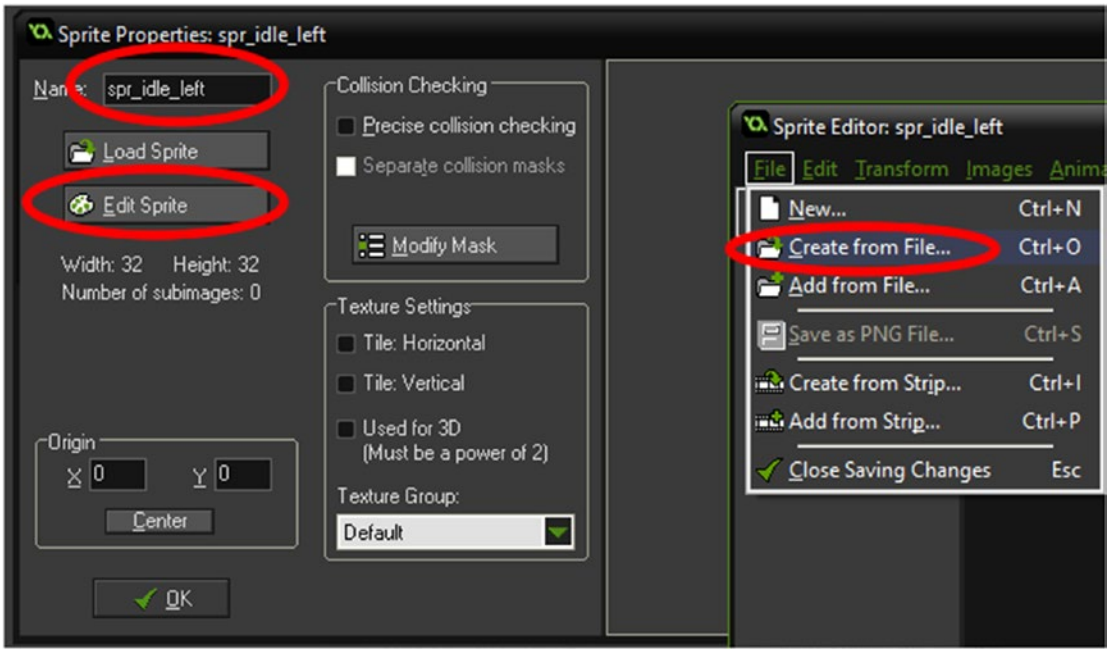

Figure 1-3. *Creating a new sprite from a file*