User Handbook

1.Introduction

Welcome to the world of *Yarnspin*, a friendly and approachable game engine designed to help you create your very own choose-your-own-adventure story games! Drawing inspiration from home computer game development tools from the 1980s and 1990s, Yarnspin aims to provide an accessible and enjoyable experience, whether you're a seasoned developer or just starting out. In this introductory chapter, we'll cover the essential information you need to get started with Yarnspin, including system requirements, how to make the most of this manual, and what you'll find in the Yarnspin package. So, let's dive in!

System Requirements

Yarnspin games will run on Windows, MacOS, Linux and web browsers.

The Yarnspin development tools runs on Windows, MacOS and Linux, but it is not possible to use a web browser to develop Yarnspin games.

In addition to the Yarnspin tools, you will also need a text editor - any text editor will work, as long as it can save plain text files. Sublime (www.sublimetext.com) is a popular choice that runs on all three operating systems.

It might also be useful to have a paint program or image editor, as well as sound and music programs.

Using This Manual

This manual is designed to be your comprehensive guide to Yarnspin, providing you with all the information you need to create engaging choose-your-own-adventure story games.

In addition to explaining key concepts, the manual will walk you through the process of creating a simple story game, or "yarn". By breaking down a sample yarn, you'll gain an understanding of how the various components of Yarnspin work together to create an interactive narrative experience.

As you progress through the manual, you'll find numerous examples, tips, and best practices that will help you become proficient in using the Yarnspin engine.

What's in the package

When you open up the Yarnspin package, you'll find a set of folders and files. The folders already contains files used for the tutorial game, but as you make your own

yarns, you will replace most of these files with your own content.

Here's a quick overview of what each folder contains:

- `build/`: This folder contains support files used internally by Yarnspin to build stand-alone redistributable packages of your games. Normally, you would never have to touch the files in this folder.
- `faces/`: In here, you will save portrait pictures of all characters in your game.
- `fonts/`: A collection of fonts to give your game that perfect look and feel. A bunch of fonts are included, but feel free to add your own.
- `images/`: Here you put all images used to represent the different locations of the game.
- `palettes/`: Yarnspin games can use a palette to give the game a distinct look. This folder includes some palette definitions, but you can also add your own.
- `scripts/`: Every part of a yarn is defined in script files, and they are placed in this folder. The names or extension of the files in this folder doesn't matter - they will all be loaded and compiled into a single game.
- `sound/`: Yarnspin games can have music and sound effects, and those are placed in this folder.

And of course, you'll find two essential files:

`yarnspin.exe`: The heart of Yarnspin! This engine processes, compiles, and runs and packages your yarns.

`yarnspin.pdf`: The very documentation you're reading right now, guiding you step by step on your Yarnspin journey.

About the Author

My journey into game development began in the 1980s with a love for home computers and programming. Starting with a Commodore 64, and later moving on to the Atari ST, I spent countless hours learning to code first in BASIC and later in assembler, making games and running tabletop RPGs with friends.

Later in life, I worked as a game developer, mostly in the engine teams for games like Crackdown, Bodycount and Battlefield 4. In 2014, I decided to leave the professional game development world to return to my roots, focusing on making games as a hobby and fueling my love for retro games.

With Yarnspin, I aim to share my appreciation for game development and empower others to create their own games. Drawing inspiration from the friendly and approachable nature of 80s and 90s game development tools, I hope Yarnspin will enable you to embark on your own creative journey.

2. Getting Started

In this chapter, we'll help you dive into the world of Yarnspin by guiding you through the sample game, understanding the basic concepts of Yarnspin, and creating your very first yarn. By the end of this chapter, you'll have a solid foundation in Yarnspin and be well on your way to creating interactive stories of your own.

Running the Sample

Before you start crafting your own game, it's useful to get familiar with the look and feel of a Yarnspin game by running the provided sample. The tutorial game is a good way to understand how the engine works and visualize the concepts we'll be discussing in the next section.

To play the tutorial game, simply launch the `yarnspin` executable in the root folder of the yarnspin package.

Take your time to explore the tutorial and don't hesitate to revisit it later as you progress through this manual. It will serve as a valuable reference when working on your own projects.

Basic Concepts

Now that you have experienced a Yarnspin game first-hand, let's discuss some fundamental concepts that will help you understand the underlying structure of Yarnspin projects.

Sections

Sections are the building blocks of your Yarnspin game. They define the structure of your story and allow you to organize your game into manageable parts. There are several types of sections, each serving a specific purpose.

Location Sections

Location Sections Location sections represent different places in your game. They serve as the backdrop for the story and give context to the player's actions. Locations can be described in text, and you can use images to further enhance the experience. Options within location sections provide players with choices to navigate through the game world and interact with their surroundings.

Dialog Sections

Dialog sections handle the conversations between characters in your game. They enable you to create dynamic and engaging dialogues, allowing the player to interact with and influence the story through their choices. Dialog options give players the ability to respond to other characters, shaping the direction and outcome of the conversation.

Character Sections

Character sections define the attributes of each character in the story. They provide a way to store and manage information about the characters, such as their names and images.

Globals

Globals are declarations that control the overall appearance and behavior of your game. They include settings like the game's title, author, starting section, fonts, colors, and more. Globals are essential for customizing your game and ensuring a consistent look and feel.

Flags

Flags are used to track the state of your game, such as which events have occurred, and can be set, cleared, or toggled. They allow you to create dynamic, interactive experiences by altering the game state based on player choices and actions.

Items

Items are objects that the player can collect, use, or drop throughout the game. They play a crucial role in creating a sense of immersion and encouraging exploration within your Yarnspin game.

Conditions

Conditions determine whether certain elements should be displayed or activated based on the current state of your game. By using conditions with flags and items, you can create branching paths and interactions that depend on what the player has done or what they possess.

With these core concepts in mind, you'll have a clearer understanding of how Yarnspin games are structured and how the different elements work together. In the next section, we'll guide you through creating your very first yarn, where you'll put these concepts into practice.

Your First Yarn

In this section, we'll guide you through creating a very simple Yarnspin game from scratch. We'll start by removing the existing scripts and then walk you through writing a small, but functional, script file. The goal is to create a single location with a single dialog to help you understand the basic structure of a Yarnspin game.

Step 1: Removing Existing Scripts

Before you start creating your own game, make sure to remove any existing scripts in the 'scripts' folder. This will ensure that your new script won't conflict with any pre-existing files.

Step 2: Creating a New Script File

Create a new text file in the 'scripts' folder and name it "my_first_yarn.txt". This file will serve as the script for your simple Yarnspin game.

Step 3: Defining Globals

At the top of the "my_first_yarn.txt" file, define the following globals:

```
title: My First Yarn
author: Your Name
start: my_location
```

This sets the title, author, and starting location for your game.

Step 4: Creating a Location Section

Now, let's create a simple location section. Add the following code to your script file:

```
=== my_location ===

img: office.jpg
txt: You are standing in a small room. There is a person here, it's Carol

opt: Talk to Carol.
act: my_dialog
```

This code defines a location called "my_location" with an image and description. There's also an option for the player to talk to Carol, which will lead to a dialog section.

Step 5: Creating a Character Section

Before defining a dialog with a character, we need to create a character section. Add the following code to your script file:

```
=== carol ===

name: Carol the Neighbor
short: Carol
face: carol.jpg
```

This code defines a character named Carol, with a short name and an image.

Step 6: Creating a Dialog Section

Next, let's create the corresponding dialog section. Add the following code to your script file:

```
=== my_dialog ===

carol: Hello! Welcome to my humble abode.
player: Hi, thank you for having me.

say: Say goodbye and leave.
act: end_conversation
```

This code defines a simple dialog section called "my_dialog" with a conversation between the player and Carol. The player has an option to say goodbye and leave the conversation.

Step 7: Ending the Conversation

Finally, let's create an action to end the conversation and return to the location. Add the following code to your script file:

```
=== end_conversation ===

carol: Goodbye! Have a great day!
act: my_location
```

This code defines an action called "end_conversation" that ends the conversation and returns the player to "my_location".

Step 8: Running Your First Yarn

Now that you've created your simple Yarnspin game script, save the "my_first_yarn.txt" file and run the Yarnspin compiler to generate the "yarnspin.dat" file. To play your game, simply launch the Yarnspin player.

Congratulations! You've just created your very first Yarnspin game. As you become more comfortable with the Yarnspin scripting language, you can start to add more locations, dialogs, characters, and interactions to

3. Unwinding the sample Yarn

Overview

start.txt

locations.txt

carol.txt

john.txt

alice.txt

4. Spinning a Yarn

Planning

Story

Scripting

Art

Sound

Package and release

5. Scripting Reference

When you run `yarnspin.exe` it will compile all the scripts and assets into a single compressed `yarnspin.dat` file. You can then distribute `yarnspin.exe` and `yarnspin.dat`, and that is the complete game ready for distribution. If there is no `scripts` folder in the same location as `yarnspin.exe`, it won't attempt compilation.

When compiling a yarn, it will load all files, regardless of extension, in the `scripts` folder and try to compile them. It doesn't matter what you put in different files, all files will be loaded and processed in one go. A script file can contain many `sections`, where a section is declared by putting three equal signs before and after its name - and names must be unique across all files. Like this: `=== my_section ===` Everything that comes before the first section in a file is read as a `global`, see below.

Sections comes in three flavours: location, dialog and character, but they are all declared that same way.

Note: as you read this documentation, make sure you have played through the tutorial game, which illustrates the concepts described here. It probably also helps to revisit it and look at its scripts as you read through the docs.

The tutorial game can be played here:

https://mattiasgustavsson.com/wasm/yarnspin

Location sections

A location section can contain one or more image and text declarations, as well as options. Each declaration can optionally have a condition before it, and the declaration will only be included if the condition evaluates to true. Conditions can only test flags, and are written with the flag name before a question mark like this: `my_flag ? txt: This text will only display if my_flag has been set` You can check if a flag is not set by placing the word `not` before it `not my_flag ? txt: This text will only display if my_flag has NOT been set` You can check if any out of a list of flags are set, by listing multiple flags `my_flag other_flag third_flag ? txt: This text will only display if ANY of the three flags have been set` You can check if several flags are set by writing them as multiple condition statements `my_flag ? other_flag ? third_flag ? txt: This text will only display if all three flags have been set` Please note that when using `not` for multiple flags, the `not` is only apply for the single flag following it, not to the whole list of flags.

A section can use `img` to display an image `img: picture_of_a_room.jpg` Note that all images must be in the `images` folder. If multiple images are specified, only one will actually be displayed - use condition statements to control which one.

A section can use `txt` to display text, and it can have multiple txt statements to display multiple texts `txt: This text will be displayed. txt: And so will this.` Note that you can use conditions to control which texts are displayed.

A section can use `act` to perform action, like setting, clearing or toggling a flag `act: set my_flag act: clear other_flag act: toggle third_flag` or go to another section, after the player clicks the mouse or press a key to dismiss the current one `act: my_other_section`

The `act` statement can also be used to get or drop items `act: get Some item` or `act: drop Some item`

After the img/txt/act declarations, a location section can have `use`, `chr` and `opt` declarations

`chr` declarations adds a character to the character list, and if the player clicks on it, we go to the section specified in its corresponding `act` statement. A `chr` declaration is always followed by an `act` statement with a section name. `chr: some_character act: talk_to_character` Note that `some_character` needs to be defined somewhere as a character section (see below) and that `talk_to_character` needs to be defined as either a location section or a dialog section

`use` works similarly, but for items `use: Some item act: section_describing_what_happens` If the player does not currently have the item in his inventory, the use declaration is ignored.

`opt` adds an option at the bottom of the screen, and also has a corresponding `act` declaration specifying a section to go to `opt: I want to go to the other section act: my_other_section`

Note that `use`, `chr` and `opt` declarations can also have conditions specified before them, and the condition will control whether the following use/chr/opt declaration is active or not.

A section is determined to be a location section if it contains any of these declarations listed, and no other types of declarations.

Dialog sections

A dialog section can not use the `img`, `txt` or `chr` declarations, but it can use the `act` and `use` declarations as described for the location sections, including conditions.

The main part of the dialog are phrase declarations, which consists of a character name followed by some text: `some_character: Hello! some_character: Good to see you again` Note that `some_character` must have been declared somewhere as a character section (see below). The character section defines the displayed name of the character, and what image is used as its portrait.

You can also use the pre-defined character `player` for lines that are spoken by the player `player: Hey, maybe you can help me? some_character: I certainly hope so! player: You are most kind.` Lines spoken by `player` will be displayed at the bottom of the screen and can be in a different font/color.

There can be as many phrase declarations as you like, and they will be run through in sequence. After all have been displayed, you can have `use` declarations, working just like for location sections, and `say` declarations, which allows the player to choose what to say next. They work just like the `opt` declarations for location sections, but for dialogs. “` some_character: How can I help you?

say: I don’t think you can… act: some_section

say: I need a million dollars, right now! act: other_section ” `Just as for location sections, these can have conditions, and the` act` statement is a section (dialog or location) to jump to when the option is selected.

A section is determined to be a dialog section if it contains any of the declarations listed for dialog sections, and no other types of declarations.

Character sections

Character sections are much simpler, and you can not jump to a character section with an `act` statement. A character section defines the name and appearance for a character only.

```
 === some_character ===
name: The Abominable Snowman
short: Frosty
face: cool_portrait.png
```

The section name `some_character` is used in location sections, in the `chr` statements, and in dialog sections in phrase declarations. The section name is not displayed anywhere, it is just for referring to the character.

When a character is added to a location using the `chr` statement, the `short` name is displayed in the list to the left on the screen.

When a dialog is playing, the longer `name` is displayed above the portrait picture defined as `face`. All portrait images must be in the `faces` folder. There are 1000 auto generated portrait images included, but you can of course make your own as well.

Globals

Any declarations that appear before the first section definition in a file is a global declaration. These control the overall appearance and behaviour of the game. Each global can be declared only once throughout all files, but it doesn't matter which globals are declared in what file.

The list of globals are:

```
title
```

The name of this yarn - will be displayed in the title bar of the window of native builds.

```
author
```

Your name, as the author of the yarn

```
start
```

Specifies which section the game starts in. Must be a defined dialog or location section.

```
items
```

Items, as used with `get` / `drop` / `use` declarations, doesn't have to be declared ahead of referring to them. But sometimes you might want to, as to avoid spelling mistakes and hard to find bugs. If you specify the `items` global, it must contain a comma separated list of ALL items referred to in any `get` / `drop` / `use` statements in your scripts, or you will get a compile error. Specifying `items` is optional, but if you do specify it, all items must be listed.

```
flags
```

Just as for items, you might want to explicitly pre-define all flags before using them in `set` / `clear` / `toggle` statements or conditions. The `flags` global is optional, but if present it must list all flags.

```
palette
```

This points to an image file in the `palettes` folder, which will be processed and used as the palette for the game. An image used as a palette must have at most 256 distinct colors, but may have less. To process images for the game, a look up table has to be generated the first time a palette is used, and this can be a slow operation, especially for palettes with many colors. But it only needs to be done

once per palette, and only while you are writing your yarn - the final distribution contains just pre-processed, run-time ready data.

```
display_filters
```

Yarnspin has two different CRT emulation filters, emulating the look of either an old TV or an old PC monitor. This global allows you to specify which one to use, like `display_filters: tv` or `display_filters: pc`. You can turn the filter off as well, for crisp pixels: `display_filters: none`. It is also possible to specify a list of filters, in which case the player will be able to cycle through them, in the order specified, by pressing F9 in the game. Declaring multiple filters looks like this: `display_filters: tv, pc, none`.

```
logo
```

Specifies one or more image files, as a comma separated list, to display at the start of the game, before jumpin to the first section. Images needs to be in the `images` folder, and will be displayed in order, waiting for player input to dismiss each one.

```
alone_text
```

On the left of the screen is a list of characters present at the current location. If there are no characters in a location, the default is to display a text there instead that says `You are alone.`. Using this global, you can change what the text says, or disable it altogether by simply specifying `alone_text:`

```
font_description
font_options
font_characters
font_items
font_name
```

These globals specify font files to use for the various text areas in the game. Font files must all be stored in the `fonts` folder, and must be .ttf files containing pixel fonts. A selection of fonts have been included. If these globals are not specified, the default fonts will be used.

```
background_location
```

Specifies an image file (present in the `images` folder) to use as a background when the game is displaying a location section.

```
background_dialog
```

Specifies an image file (present in the `images` folder) to use as a background when the game is displaying a dialog section.

```
color_background
color_disabled
color_txt
color_opt
color_chr
color_use
color_name
color_facebg
```

These globals controls the text display color for the various text areas in the game. They specify the index in the palette (0 to 255) of the color to use for each text. If not specified, defaults will be calculated and used.

Adjusting images

Yarnspin has a built-in editor to make basic adjustment to your images and portraits, and preview how they will look after processing, as well as try them out with different palettes. You can run the image editor by launching yarnspin with the commandline parameters `-i` or `--images`, like this:

```
  yarnspin --images
```

6. Additional Resources

Story structure

Art tutorials

Poser

AI

Sound

Music

Yarnspin source code

Other game development tools