



TiledPlanet Guide

By gimic970

Copyright & Version Information

TiledPlanet Designer Guide

© TiledPlanet.com. All rights reserved.

This document is provided for instructional purposes.

No part of this guide may be reproduced, distributed, or transmitted in any form without prior written permission, except where permitted by law.

Version: 1.0

Last Updated: 1/29/2026

TiledPlanet and TP Designer are trademarks of TiledPlanet.com.

Introduction

Interactive storytelling is not linear.

Unlike traditional narratives, interactive stories respond to player choice, adapt to past actions, and evolve based on state. Writing such stories requires not only imagination, but also structure — a way to track decisions, manage consequences, and maintain narrative clarity as paths branch and reconnect.

TiledPlanet Designer was created to make this process accessible.

This guide provides a complete, practical walkthrough of how to design, build, test, and publish interactive stories using TP Designer. It focuses on **how the system works**, why certain design patterns are effective, and how to think about interactive narrative as both storytelling and system design.

This is not a guide to literary craft.
It is a guide to mastering the tool.

Who This Guide Is For

This guide is written for:

- Writers exploring interactive fiction
- Game designers prototyping narrative systems
- Hobbyists and professionals building choice-driven stories
- Creators with varying levels of technical experience

No programming knowledge is required. Familiarity with basic storytelling concepts is helpful but not necessary.

How This Guide Is Structured

This guide is organized to move from **concept to execution**, then from **creation to publication**.

Early chapters focus on understanding how interactive stories work and how to plan them effectively. Middle chapters walk through building a complete story using TP Designer. Later chapters cover advanced techniques, publishing, and working within system constraints.

Each chapter builds on the previous one, but experienced users may choose to jump directly to specific sections.

How to Use This Guide

You may use this guide in several ways:

- **Read end-to-end** to build a complete mental model of TP Designer
- **Follow alongside the tutorial project** to learn by doing
- **Reference specific chapters** when designing or debugging a story

Sections that involve detailed mechanics are intentionally separated from conceptual explanations to keep the reading experience clear and approachable.

Screenshots and figures are referenced throughout the guide and collected in the appendix for easy lookup.

Table of Contents

Chapter 1

Understanding Interactive Stories in TiledPlanet

Chapter 2

Planning Your Story Before You Build

Chapter 3

Creating a Project and Setting Up Your Game

Chapter 4

Resources, Items, Flags, and Visual Identity

Chapter 5

Writing the Story: Nodes, Options, and Outcomes

Chapter 6

Designing Flow — Branches, Conditions, and Consequences

Chapter 7

Embedded Text Logic (ETL): Writing Reactive Story Text

Chapter 8

Limitations & Workarounds

Chapter 9

Publishing, Marketing, and Discovery

Chapter 10

Designing Around Constraints

Chapter 11

Go Write Your Story

Appendix A

Figures and Screenshots

Appendix B

Embedded Text Logic Reference

Appendix C

Common Design Patterns and Workarounds

Chapter 1

How TiledPlanet Stories Work

Before beginning work in the designer, it is important to understand the structural model behind interactive stories created on TiledPlanet.

A TiledPlanet story is not linear prose, nor is it a traditional branching tree. Instead, it is best understood as a **network of narrative moments** connected by player choice and governed by persistent story state.

Story Nodes as Narrative Units

All interactive stories on TiledPlanet are composed of **story nodes**.

A story node represents a discrete moment in the narrative. This moment may describe a situation, reveal information, resolve an action, or present a decision. When a node is reached, it may perform one or more of the following actions:

- Display narrative text
- Present imagery or play audio
- Modify the story's state
- Offer choices that determine what happens next

Story nodes should be thought of as **narrative beats**, not pages of text. Each node exists to move the story forward through action or consequence.

Options Represent Player Intent

Most story nodes present the player with **options**.

An option represents what the player attempts to do in that moment. Examples include searching an area, speaking to a character, using an item, or choosing to wait.

Each option leads to another story node, which represents the result of that action.

It is important to distinguish between the two:

Option text expresses player intent.

Node text expresses narrative outcome.

The option a player selects and the node it leads to do not need to share the same wording or meaning. This separation allows authors to write clear choices while maintaining descriptive, consequence-driven storytelling.

Nonlinear Narrative Structure

TiledPlanet stories are inherently nonlinear.

Story nodes may:

- Branch into multiple paths
- Rejoin at shared outcomes
- Loop back to earlier moments
- Transition between chapters
- Conclude the story in success or failure

This flexibility allows authors to design complex narrative structures without unnecessary duplication. Reusing nodes and reconverging paths helps keep stories manageable while preserving depth.

Conceptually, a TiledPlanet story functions as a **graph**, not a tree.

Persistent Story State

Stories remember what the player has done. This memory is represented through **flags** and **items**.

Flags

Flags represent events or conditions that have occurred. Examples include:

- A choice that was made
- An action that was completed
- An irreversible event

Flags answer questions such as:

“Has this already happened?”

Items

Items represent objects or capabilities the player possesses. Examples include keys, tools, or equipment.

Items answer questions such as:

“Does the player have what is required to perform this action?”

Flags and items do not affect the story on their own. They only influence the narrative when a story node explicitly references them to control outcomes, options, or text.

Consequences and Endings

Every choice in a TiledPlanet story should lead to a meaningful result.

That result may be:

- Narrative progress
- A temporary setback
- A branching path
- A successful ending
- A failure ending

Failure is not an error state. It is an intentional narrative outcome that reinforces player learning and story tension. Well-designed stories use consequences to guide players naturally, without overt instruction.

Iterative Story Development

Stories on TiledPlanet are designed to be built incrementally.

Authors may:

- Begin with placeholder text

- Revise narrative content at any time
- Replace or update media assets
- Restructure story flow without rewriting the entire story

No element of a story is permanently fixed. This flexibility encourages experimentation and refinement throughout the creative process.

What Comes Next

With an understanding of how TiledPlanet stories are structured, the next step is learning how to **plan a story before building it**.

The following chapter focuses on:

- Visualizing story flow
- Identifying key decision points
- Planning flags and items
- Avoiding structural problems before writing narrative text

Chapter 2

Planning Your Story Before You Build It

Before opening the designer and creating story nodes, it is strongly recommended that you plan your story at a high level. While TiledPlanet allows for easy iteration and revision, thoughtful planning reduces complexity, prevents dead ends, and results in a more cohesive narrative experience.

This chapter focuses on structuring your story *before* writing dialogue or configuring logic.

Think in Outcomes, Not Text

Effective planning begins by identifying **outcomes**, not scenes or dialogue.

Ask yourself:

- How can the story end?
- What constitutes success?
- What constitutes failure?
- Are there multiple paths to victory?
- Are there multiple forms of defeat?

By defining outcomes early, you establish narrative boundaries that guide every decision that follows. Story text can evolve later; structural intent should be defined first.

Identify Major Decision Points

Once outcomes are defined, identify the **decisions that meaningfully affect the story**.

Meaningful decisions:

- Change the direction of the narrative
- Alter what becomes possible later
- Have lasting consequences

Avoid over-planning minor choices. Not every option needs to branch the story. Some choices exist only to reinforce immersion or pacing.

At this stage, focus on decisions that:

- Gate access to future content
- Introduce risk or reward
- Permanently alter the story state

Sketch the Story Flow Visually

Planning is most effective when done visually.

Before writing any story text, sketch your story. See figure 1 for an example of what your sketch might look like.

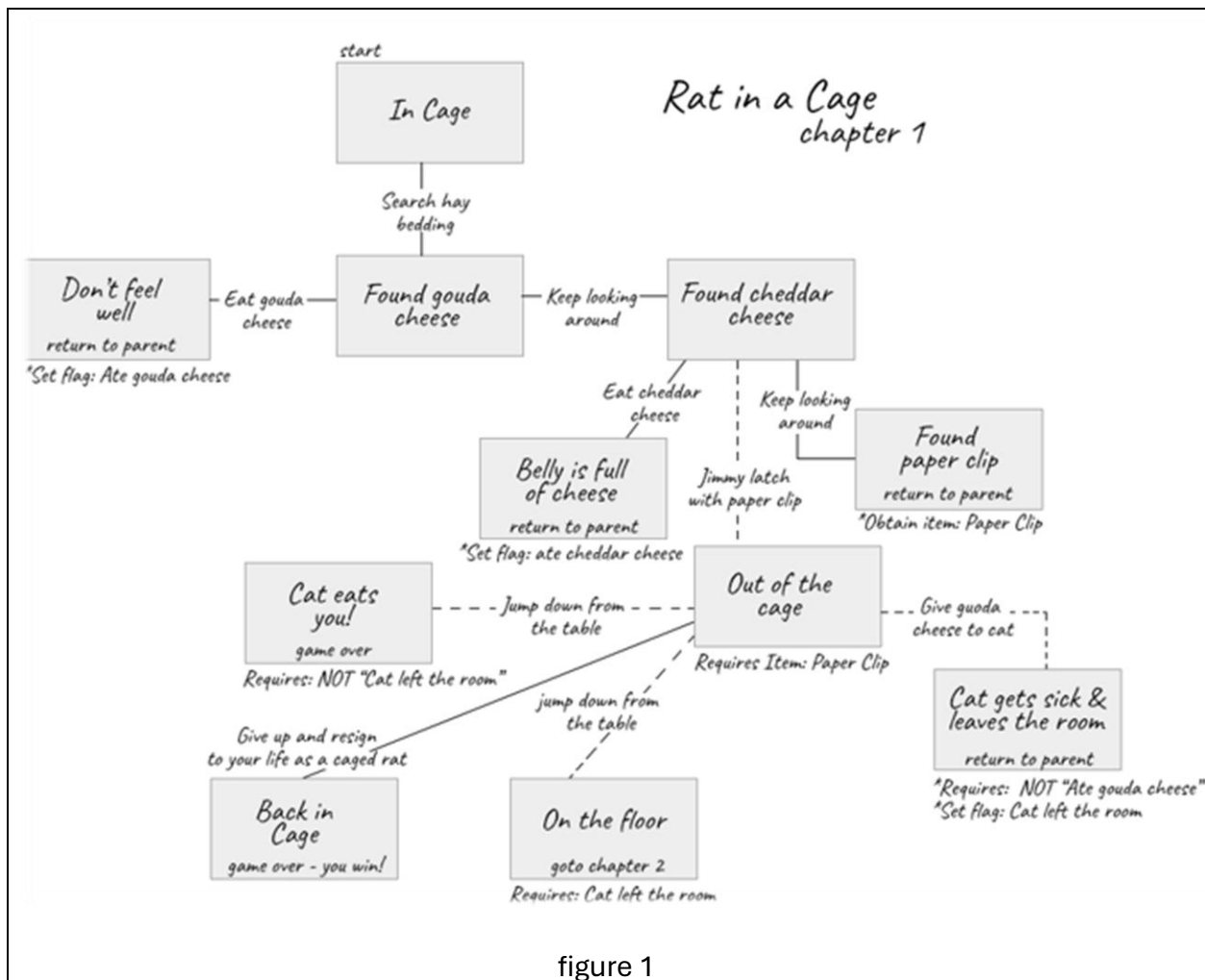


figure 1

Figure 1 contains boxes, lines, dotted lines, and various annotations:

- Boxes represent story nodes
- Lines represent options
- Dotted lines represent hidden options (which appear when a requirement has been met)
- Flags annotations (i.e. Ate cheddar cheese, etc.)
- Node behavior annotations:
 - Return to parent
 - Game over
 - Goto chapter

This visual approach makes it easy to:

- Identify dead ends
- Spot unnecessary complexity
- Ensure all paths lead somewhere meaningful

The diagram does not need to be detailed. A rough sketch is sufficient as long as it communicates structure.

Plan Flags as Events

Flags should be planned as **events**, not variables.

Each flag should represent something that *happened*, such as:

- A discovery
- A choice
- A consequence
- A point of no return

When planning flags, consider:

- Which events permanently change the story?
- Which choices should unlock or hide future options?
- Which outcomes depend on earlier actions?

Flags are most effective when they are:

- Clearly named
- Irreversible
- Used intentionally

Avoid creating flags for minor or temporary states unless they are essential to story logic.

Plan Items as Capabilities

Items represent what the player **has**, not what has occurred.

When planning items, ask:

- What physical objects matter to the story?
- Which actions should require possession of something?
- Which items should be consumed when used?

Items are particularly useful for:

- Gating access to options
- Creating trade-offs
- Reinforcing cause-and-effect relationships

As with flags, plan items early to avoid retrofitting logic later.

Anticipate Loops and Returns

Nonlinear stories often benefit from **loops**, where players return to a previous decision point after taking an action.

When planning loops:

- Decide whether options should reappear

- Determine whether choices should be repeatable
- Plan flags to hide options that should not return

Looping intentionally allows you to:

- Present more options than a single node allows
 - Encourage exploration without overwhelming the player
 - Reuse narrative space efficiently
-

Separate Structure from Writing

At this stage, avoid writing detailed narrative text.

Use placeholders such as:

- “Finds an item”
- “Encounters danger”
- “Returns to safety”

Separating structure from prose ensures that:

- Logic is sound before writing begins
- Narrative tone remains consistent
- Rewrites do not require structural changes

Once the structure works, writing becomes significantly easier.

Planning Prevents Rework

While TiledPlanet supports easy revision, unplanned stories often require:

- Rewriting large sections
- Retrofitting flags
- Untangling unnecessary branches

Even a simple planning session can eliminate most structural issues before they arise.

Time spent planning is time saved later.

What Comes Next

With a clear story plan in place, the next step is creating a project and defining its presentation.

The following chapter covers:

- Creating a new project
- Choosing story type and plan
- Setting colors, fonts, and identity
- Preparing a story to be playable from the start

Chapter 3

Creating a Project and Establishing Identity

With a clear understanding of how TiledPlanet stories function and a plan for your narrative structure, the next step is creating a project. A project serves as the container for your story, its resources, and its public identity.

This chapter explains what a project represents, which decisions matter at creation time, and which choices can safely be revisited later.

What a Project Represents

A project is the foundational unit of a TiledPlanet game. It contains:

- All story nodes and chapters
- Items and flags used to track story state
- Media resources such as images, sound, and music
- Presentation settings, including fonts and colors
- Marketing and publishing information

When a project is created, TiledPlanet automatically generates a starting story node and loads default resources. This means the project is immediately playable, even before any story content is written.

Naming Your Game

Your game's name is a core part of its identity.

The project name:

- Must be unique across the platform
- Becomes part of the game's public URL
- Is visible to players in listings and search results

Because the name appears in URLs, it is best to choose something concise and readable. Subtitles and descriptive taglines can be added later and do not affect the game's address.

Selecting a strong name early helps establish consistency across marketing, sharing, and player discovery.

Choosing a Story Type

TiledPlanet supports multiple game formats. This guide focuses on **Interactive Stories**, which are node-based narratives driven by player choice and story state.

Other formats may exist or be introduced in the future, but the core concepts covered in this guide apply specifically to interactive storytelling.

Once selected, the story type defines the overall structure of the project but does not limit creative expression within that format.

Understanding Project Plans

Project plans primarily control **scale**, not storytelling capability.

Plans may differ in:

- Maximum number of story nodes
- Availability of certain visual features
- Resource limits

All plans support the same fundamental narrative mechanics: nodes, options, flags, items, and conditional logic.

When choosing a plan, consider the expected size of your story rather than its complexity. Many complete stories can be built comfortably within modest limits.

Visual Identity and Presentation

Each project has a visual identity that frames the player's experience. This includes:

- Background and interface colors
- Fonts used for story text and titles
- Introductory text and title presentation

These elements affect tone and atmosphere but do not influence story logic. They are intentionally flexible and can be changed at any time.

Authors are encouraged to avoid over-polishing visual details early. Establishing a consistent identity is useful, but presentation can always be refined once the story structure is complete.

Projects Are Safe to Experiment With

One of the core design goals of TiledPlanet is to encourage experimentation.

At any point, you may:

- Rename the project
- Change visual settings
- Replace media assets
- Restructure story nodes

These changes do not break the project or invalidate earlier work. Nothing is permanently locked, and iteration is expected.

This flexibility allows creators to focus on storytelling rather than configuration.

“Ready-to-Run” by Design

Because every new project includes a starting node and default resources, it is always in a runnable state.

This design serves two purposes:

- It allows authors to test the game engine immediately
- It reinforces the idea that a story can be built incrementally

Even an unfinished project can be played, tested, and refined as it grows.

What Comes Next

With a project created and its identity established, the next step is preparing the building blocks your story will rely on.

The following chapter explores:

- Items and flags as story state
- Media resources and how they are reused
- Separating narrative logic from presentation

Chapter 4

Resources, State, and Visual Identity

Before writing story text, it is useful to understand the systems that support and shape your narrative. TiledPlanet separates **story logic**, **story state**, and **presentation** into distinct layers. This separation allows stories to remain flexible, maintainable, and easy to revise.

This chapter introduces the concepts of items, flags, and media resources, and explains how they work together without becoming tightly coupled to the story itself.

Story State and Narrative Memory

Interactive stories rely on memory. The story must be able to remember what the player has done and what they currently possess.

TiledPlanet models this memory through **flags** and **items**.

These elements exist independently of story nodes and are referenced when needed.

Flags as Narrative Events

Flags represent events or conditions that have occurred during the story.

A flag typically indicates that:

- A decision has been made
- An action has been completed
- An irreversible change has occurred

Flags are best understood as statements of fact about the story's past.

Examples include:

- A character was helped
- A location was visited
- A promise was broken

Once applied, a flag remains part of the story's state unless explicitly designed otherwise.

Flags are commonly used to:

- Unlock or hide options
 - Trigger alternate outcomes
 - Modify descriptive text
-

Items as Player Capabilities

Items represent objects or tools the player carries.

Unlike flags, items may:

- Be gained
- Be required to perform actions
- Be consumed or removed

Items are well suited for situations where:

- Physical possession matters
- Trade-offs are involved
- The story benefits from tangible resources

Examples include keys, tools, weapons, or symbolic objects.

As with flags, items do nothing on their own. They influence the story only when a node explicitly checks for or modifies them.

Media Resources as Reusable Assets

Media resources include images, illustrations, sound effects, and music.

These resources are managed globally and referenced by story nodes as needed. A node does not contain media; it points to media stored elsewhere.

This design allows:

- A single image or sound to be reused in multiple places
- Assets to be replaced without modifying story logic
- Visual and audio consistency across the story

Replacing a resource updates it everywhere it is used, making iteration safe and efficient.

Presentation Is Separate from Logic

Visual and audio presentation enhances storytelling but should never restrict it.

Presentation elements include:

- Title screens and menus
- Interface frames or backgrounds
- Fonts and color palettes
- Introductory text

These settings establish tone and atmosphere but do not affect how the story behaves.

Authors are encouraged to treat presentation as a layer that can be refined later, once the story's structure and logic are stable.

Preparing Resources Before Writing

Some authors prefer to prepare images and audio before writing, while others work in the opposite direction. TiledPlanet supports both approaches.

Preparing resources early can:

- Clarify tone and mood
- Inspire narrative decisions
- Reduce interruptions later

However, no resource decision is permanent. Assets can be added, removed, or replaced at any time without affecting the underlying story structure.

Visibility and Feedback During Creation

As stories grow, it becomes increasingly important to understand where flags, items, and resources are being used.

TiledPlanet provides visual indicators within the story diagram to show:

- Where flags are applied

- Where items are acquired
- Which nodes represent endings or transitions

These indicators exist to assist authors during development and are not visible to players.

What Comes Next

With story state and resources defined, you are ready to begin constructing the narrative itself.

The next chapter introduces **story nodes** in detail and explains how individual moments combine to create interactive flow.

Chapter 5

Story Nodes — Moments That Change the World

At the core of every TiledPlanet story is the **story node**. Story nodes are the fundamental building blocks from which all narrative flow, logic, and consequence emerge.

This chapter explains what a story node represents, how nodes behave, and how complex stories are constructed from simple, repeatable rules.

What a Story Node Represents

A story node represents a **single narrative moment**.

That moment may:

- Describe a situation
- Resolve an action
- Reveal information
- Present a decision
- Apply consequences

A node is not a “page” of text, nor is it a container for multiple unrelated events. It is a focused unit of narrative purpose. Refer to figure 1 and see how this relates to the actual user interface screenshot in figure 2.

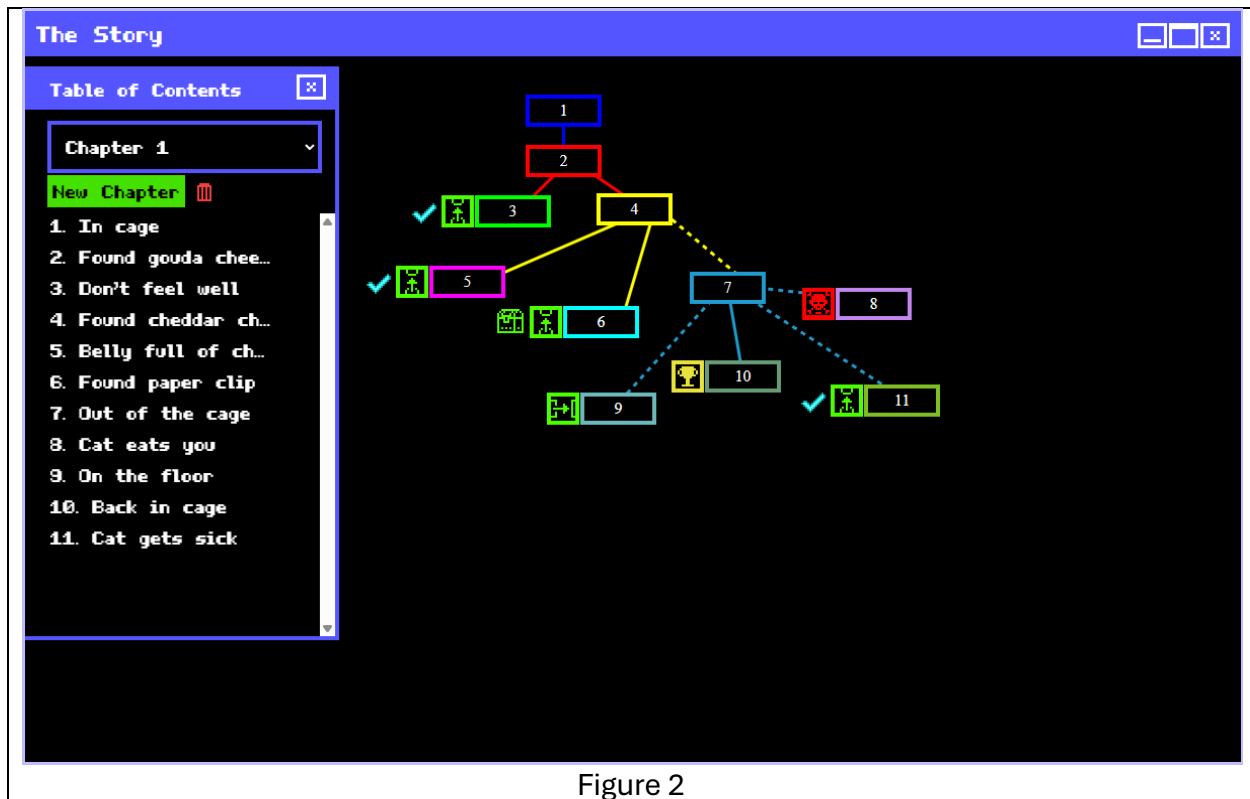


Figure 2

When a node is reached, it executes fully before presenting any choices to the player. Optionally, a node can be accompanied by a icon(s) that symbolizes how the node behaves or flows:

	Go back to parent node
	Go to a chapter
	Game ends in defeat
	Game ends in victory
	A flag has been set
	An item has been obtained

What Happens When a Node Runs

When a story node is entered, it may perform several actions in sequence:

- Display narrative text
- Present an illustration or play audio
- Apply one or more flags
- Grant or consume items
- Determine how the story may proceed

Only after the node completes these actions are options presented to the player.

This predictable execution order allows authors to reason clearly about cause and effect.

Nodes Define Outcomes, Not Just Transitions

It is tempting to think of nodes merely as connectors between choices. In practice, nodes are where **meaning** happens.

A well-designed node answers at least one of the following questions:

- What changed as a result of the last choice?
- What does the player now understand?
- What is at stake going forward?

Even nodes that return the player to a previous decision point serve an important role by acknowledging actions and reinforcing consequences.

Options Belong to Nodes

Options are always defined **within** a node.

Each option represents a possible action the player may attempt. Selecting an option transitions the story to another node, which represents the result of that action.

It is important to separate:

- **Option text**, which expresses player intent

- **Node content**, which expresses narrative outcome

This separation allows authors to write clear, intuitive choices while retaining full control over narrative presentation.

Nodes Can Be Reused

A single story node may be reached from multiple paths.

This enables:

- Converging story branches
- Shared failure states
- Common narrative outcomes

Reusing nodes reduces duplication and helps maintain consistency across the story. It also makes large stories easier to reason about and revise.

Temporary Actions and Returning to Decisions

Not all nodes move the story forward permanently.

Some nodes exist to represent short actions such as:

- Examining an object
- Eating or using an item
- Reading a note

These nodes perform their narrative function and then return the player to the previous decision point. When combined with story state, they allow authors to acknowledge actions without forcing permanent progression.

This pattern is especially useful for exploration-heavy stories.

Ending Nodes

Some nodes represent definitive conclusions.

A node may explicitly end the story in:

- Victory
- Defeat

Ending nodes provide closure and signal that no further choices remain. They are an intentional part of story design and should be planned with the same care as branching paths.

Failure endings are not errors; they are narrative outcomes that reinforce tension and learning.

Chapters and Structural Organization

As stories grow, nodes may be organized into **chapters**.

Chapters exist to:

- Reduce visual complexity
- Group related sections of the story
- Improve authoring clarity

Chapters do not isolate logic or reset state. Transitioning to a chapter is simply a way of moving the story to another section of the narrative graph.

Designing Nodes with Purpose

Effective nodes are deliberate.

Before creating a node, consider:

- What this moment accomplishes
- How it affects the story state
- Whether it introduces meaningful choice

Nodes that exist only to move the player forward without consequence often weaken a story's impact.

Every node should justify its presence.

What Comes Next

With an understanding of how story nodes function individually, the next step is learning how nodes interact to form complex narrative flow.

The following chapter explores:

- Branching paths
- Conditional options
- Requirements and gating
- Re-converging story lines

Chapter 6

Designing Flow — Branches, Conditions, and Consequences

With an understanding of individual story nodes, the next challenge is designing how those nodes interact. Narrative flow is not defined by any single moment, but by the structure created when moments branch, reconnect, and react to player choices.

This chapter explains how to design story flow that is flexible, readable, and resilient as complexity grows.

Branching with Intent

Branching is the most visible feature of interactive storytelling, but not all branches are equal.

Effective branching:

- Represents meaningful differences in outcome
- Reflects player agency
- Reinforces cause and effect

Ineffective branching:

- Exists only to offer superficial variety
- Quickly collapses without consequence
- Overwhelms the player without adding value

When designing branches, prioritize **impact over quantity**. A small number of meaningful branches is more effective than many shallow ones.

Requirements and Conditional Options

Not every option should always be available.

TiledPlanet allows options to appear or disappear based on story state. These conditions are expressed using **requirements**, which typically check for:

- Completion or non-completion of flags
- Possession or absence of items

Requirements affect **visibility**, not navigation. If a requirement is not met, the option is not shown to the player at all.

This distinction allows authors to:

- Hide unavailable actions naturally
 - Present identical options with different outcomes
 - Avoid confusing the player with impossible choices
-

Identical Options, Different Outcomes

In some cases, two options may appear identical to the player while leading to different results.

This technique is useful when:

- The player lacks information
- Timing matters
- Prior actions influence success or failure

By controlling option visibility with requirements, authors can present a consistent interface while allowing story logic to remain nuanced and responsive.

Re-converging Story Paths

Uncontrolled branching can quickly become unmanageable. To prevent exponential growth, stories should be designed to **re-converge** whenever possible.

Re-convergence allows:

- Multiple paths to share outcomes
- Common failure or success states
- Reduced duplication of content

A single node may serve as the destination for several different options. This is not a compromise — it is a structural strength.

Looping and Exploration

Not all stories move strictly forward.

Loops allow players to:

- Explore environments
- Perform optional actions
- Gather information or resources

When designing loops, authors must decide:

- Whether options can be repeated
- Which actions should be removed after selection
- How state should change over time

Loops often rely on flags to hide options that have already been taken, ensuring progress without trapping the player in repetition.

Consequences as Design Tools

Consequences are the primary way a story teaches the player.

Consequences may include:

- New information
- Restricted options
- Altered outcomes
- Story endings

Failure is one of the most effective consequences. A well-placed failure teaches the player what does *not* work, while encouraging them to try again with new understanding.

Stories that avoid failure entirely often feel flat and inconsequential.

Managing Complexity

As stories grow, complexity must be actively managed.

Helpful strategies include:

- Reusing nodes where appropriate
- Grouping related nodes into chapters
- Planning reconvergence points early
- Using flags to simplify conditional logic

Clarity in structure allows creativity in content.

Designing for the Player's Experience

When designing flow, consider the player's perspective:

- Are choices understandable in the moment?
- Do outcomes feel fair, even when negative?
- Is progress visible and meaningful?

Players should feel that the story responds to their actions, even when it does not reward them.

What Comes Next

With narrative flow established, you can begin refining how the story *responds* to player history within individual moments.

The next chapter introduces **Embedded Text Logic (ETL)**, which allows story text itself to react dynamically to past events and inventory.

Chapter 7

Embedded Text Logic (ETL): Writing Reactive Story Text

Interactive stories gain much of their power from responsiveness. Beyond branching paths and conditional outcomes, a story can acknowledge *how* the player arrived at a moment, not just *where* they are.

Embedded Text Logic (ETL) enables story text itself to change based on the player's history, without altering story flow or structure.

What Embedded Text Logic Is

Embedded Text Logic allows authors to conditionally display portions of story text based on **flags** or **items** the player has acquired (or has not acquired).

In practical terms, ETL allows a single story node to describe the same moment differently depending on what has already happened in the story.

For example:

- A character may appear confident or injured
- An environment may feel familiar or hostile
- A situation may feel hopeful or dire

All without creating new nodes or branches.

What Embedded Text Logic Is Not

ETL does **not**:

- Create new story paths
- Add or remove options
- Control navigation
- Replace branching logic

ETL affects **presentation only**, not progression.

It is best understood as a narrative enhancement layer rather than a structural tool.

Appropriate Uses of ETL

ETL is particularly effective for:

- Reinforcing consequences of earlier choices
- Providing narrative callbacks
- Adjusting tone or emotional context
- Reflecting player condition or preparedness

Used sparingly, ETL deepens immersion without increasing structural complexity.

Optional Else Text

Each ETL condition may include optional alternative text.

When an ELSE condition is provided:

- One of two text blocks will be displayed

When no ELSE condition is provided:

- Text appears only if the condition is met

This makes ETL suitable for both subtle narrative additions and clear contrasts in description.

Multiple ETLs in a Single Node

A single story node may contain multiple ETL statements.

This allows authors to layer narrative detail based on different aspects of story state. For example:

- One ETL might reference an injury
- Another might reference an item
- Another might reference a past encounter

Each ETL operates independently, allowing text to accumulate naturally based on player history.

ETL and Story Design Discipline

Because ETL responds directly to story state, it assumes that the story's logic is internally consistent.

When multiple ETL conditions could evaluate as true at the same time, all matching text will appear. Authors must design story progression carefully to avoid unintended combinations.

For this reason, ETL is most effective when:

- Story state is planned intentionally
- Mutually exclusive conditions are enforced through design
- Flags and items are acquired in controlled ways

ETL rewards thoughtful story architecture.

Editing and Maintenance Considerations

ETL statements can be modified after they are created, but care must be taken to preserve their structure.

Authors should:

- Avoid altering reserved keywords
- Modify only the narrative text portions
- Test changes after editing

For advanced users, ETL statements may be written manually without using the editor interface.

Detailed syntax rules and advanced patterns are documented separately in the reference appendix.

When to Use ETL — and When Not To

ETL should be used to enhance moments that already matter.

It should not be used:

- To compensate for unclear story flow
- As a replacement for meaningful branching
- To add unnecessary complexity

When in doubt, favor clarity over cleverness.

UNDERSTANDING ETL DECLARATION

Let's examine the following ETL declaration...

[has(f63) then "Boy, the cheddar cheese in your belly is making you feel a little sluggish. A nap is in order!" else "You're feeling a little hungry. You wish you had something to eat. Oh well."]

"has" encloses Id's. Id's are preceded with an "f" or "i" (flag or item, respectively). Id's can be viewed in the Flags or Items dialogs. "f63" declares the flag with an id of 63 to be required for the ETL text to be shown. Multiple Id's are separated with a comma (a comma denotes "AND"). For example, has(f63,f64) is saying f63 AND f64 must exist. For OR, pipe is used to separate the Id's. For example, has(f63|f64) is saying the f63 OR f64 must exist. Negated Id's are preceded with an exclamation. For example, has(f63!f64) is saying f63 OR NOT f64. Tip: clicking on an Id within the ETL statement will bring up a popup showing you what the item/flag is.

You can make changes to the ETL, as needed, once it's inserted into your story text, but be careful not to ruin the statement by removing expected characters, like the brackets or reserved words like "then" or "else", etc. You may change the text between the quotations, only. You can write an ETL statement into your story text, manually, without using the ETL UI (for those who want to be savvy!).

Note: To add a quote inside a quotation, escape the quote with a backslash.

Note: You can create and insert as many ETLs into a body of text as you need.

Note: You cannot mix-and-match flags and items in the has declaration – so you cannot do: has(f34,f35,i38).

Note: You cannot mix-and-match logical operators – so you cannot do: has(f35,f36|f44).

What Comes Next

With story structure, flow, and narrative responsiveness in place, the next step is preparing your story for release.

The following chapter covers:

- Understanding limitations and workarounds

Chapter 8

Limitations & Workarounds

There are limitations to what you can do with TiledPlanet interactive text adventures. One of those limitations is, as you might have already noticed, you must tell your story with a maximum of 6 options (which include hidden options), per node. If you need 7 options (or more), you'll need to be creative and use multiple nodes to present the other options. Remember that you can always loop back to a previous node. You'll need to use flags to hide nodes that have already been selected because the "Options can be selected only once" rule will not work if you loop back – the "Options can be selected only once" logic is hardwired to examine the current node and its children (not grandchildren, great grandchildren, etc.). So again, you'll need to be creative and use flags to hide previously selected options.

Examine figure 3 below.

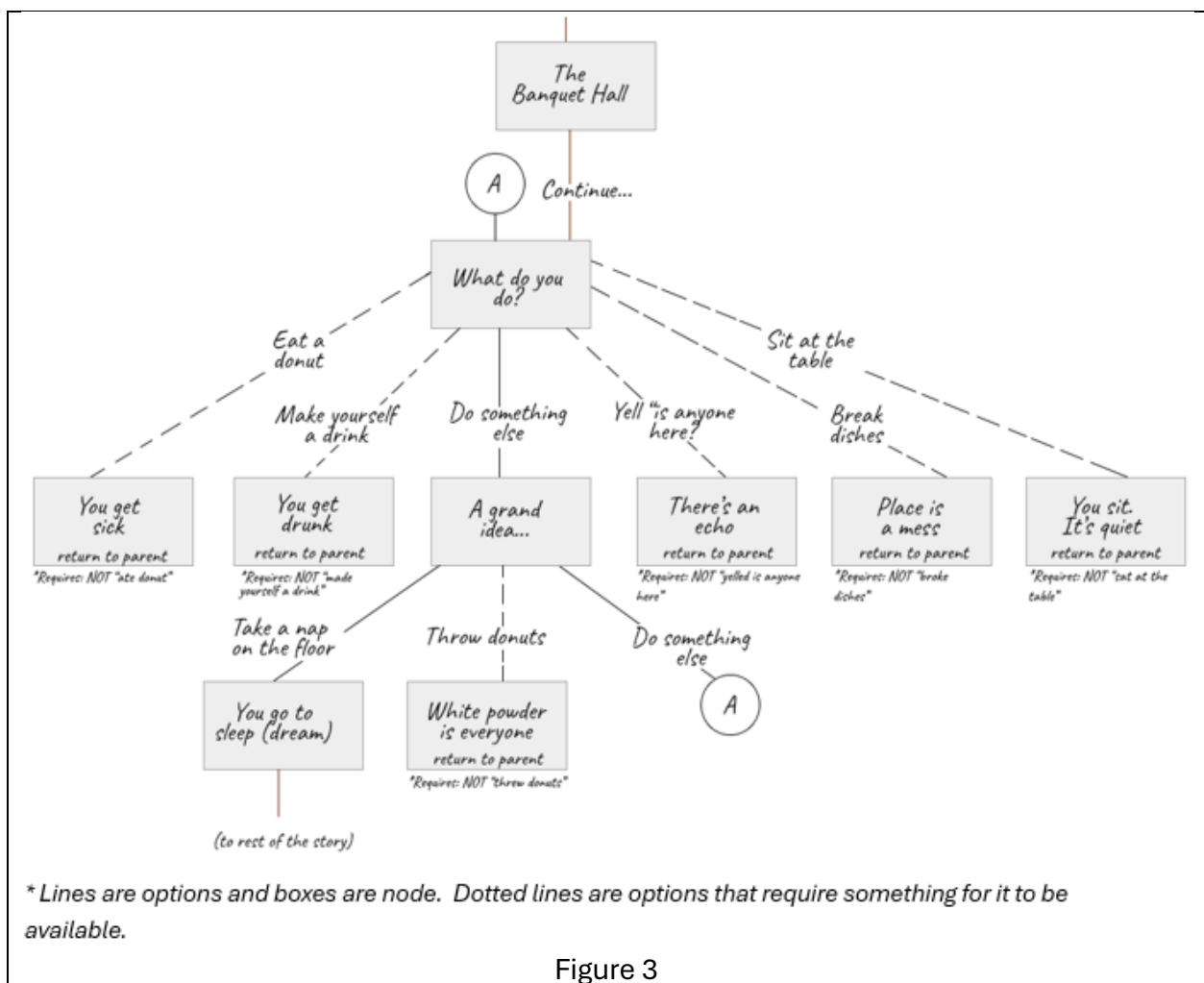


Figure 3

See what we did here? “The Banquet Hall” node has a single option “continue” – you can label it however you like – it could be something like “Contemplate your options...” or whatever, but it will be **one** option. When the player selects this option (again, it’s the only option), the next node simply displays text that reads “What do you do?” – 6 options are displayed in that node: (1) Eat a donut, (2) Make yourself a drink, (3) Do something else, (4) Yell “Is anyone here?”, (5) Break dishes, and (6) Sit at the table. 5 of these options have requirements. For example, the “You get sick” node requires that the player has NOT completed “Ate donut” (flag). When the donut is eaten, this option will no longer be available. Notice how all 5 of these nodes *return to parent*. The “What do you do?” node does not institute the “Options can be selected only once” rule – instead we are using flags to remove the options from the list.

Simple.

So, how do we extend the options beyond those 5? We create a 6th option that will display another node with more options! Perhaps, the node for that option might read “You have a grand idea to try something else...”. Then you’d show the next 3 options: (1) Take a nap on the floor, (2) Throw donuts, and (3) Give something else a try. “Take a nap on the floor” flows to the rest of the story. “Throw donuts” is displayed when the player has NOT completed “Threw donuts”. Notice how “Give something else a try” goes to the “What do you do?” node. It loops back up! Then it displays the options “Eat a donut”, etc. But remember, since we are using flags, if the donut was eaten already, that option is removed from the list!

Using this method, you can have as many options as you like – you just need to be creative in your story telling to make it happen.

Chapter Limitation

Another limitation is that nodes can NOT jump to any node in another chapter. Jumping to a chapter always puts the player at the initial node in that chapter. However, you can jump to ANY chapter from any node. For example, a node in chapter 1 can go directly to chapter 5. “Chapter” is a misnomer, really – and chapters do NOT have to go in order. Think of a chapter as part of the story you’d like to separate – perhaps to make the story more manageable and the diagram less hairy.

When a node causes a jump to a chapter, a single option appears: “Continue”. Clicking “Continue” takes the player to the designated chapter. Currently, this label is hardcoded. We plan on changing this at some point, allowing you to change this label to whatever you like. So, instead of “Continue”, it might read “Go down the dark hall”, or whatever. Look for this feature in a later release!

ETL Limitations

As you know, ETL's can examine flags and items, but not both (within the same statement). Also, the logical operator AND and OR are mutually exclusive, as well – you can pick one, not both. To get around this, you can stack your ETL's while excluding ELSE text – doing this is sort of like a bunch of if-statements. Examine the following story text...

You enter the hallway. It's dark. It's cold.

```
[has(i268) then "The grip your long bow firmly and remove an arrow from your quiver." else ""]  
[has(i288) then "You grip your staff in both hands, ready for any surprises." else ""]  
[has(i263,!i264) then "The sword feels heavy in your hand. You raise it up, in-the-ready." else ""]  
[has(i263,i264) then "The sword feels heavy in your hand. You raise it up, in-the-ready. Your  
buckler is held low as you peer over its edge." else ""]  
[has(i273|i290) then "You unsheathe your knife. You raise it up, read to stab." else ""]
```

You continue to walk down the hallway. Every step is slow and deliberate.

For these ETL statements to work correctly, the player can have the long bow, staff, sword WITHOUT a buckler, a sword WITH a buckler, or a knife. If the player were to have a staff and a knife, the text would render (in gameplay) like this...

You enter the hallway. It's dark. It's cold.

You grip your staff in both hands, ready for any surprises. You unsheathe your knife. You raise it up, ready to stab.

You continue to walk down the hallway. Every step is slow and deliberate.

This is not what we want! The goal here is to have a single ETL criteria met – which means when you construct your story, you must ensure that only one of those ETL statements will produce text. That means you must create your story in a way that leads to acquiring ONLY one of these items. However, in our example, the shield can also be acquired along with any of the other items. See how one ETL checks for a sword and a buckler (i263 = sword, i264 = shield) while the other one checks for a sword and NOT a buckler (!i264 = no shield)? If the player happens to have the staff and the buckler, it will just render the text about the “staff”. The buckler is only mentioned if it is accompanied by a sword.

Note: Remember that commas denote “AND” while pipes denote “OR”.

Lastly, the knife comes in two flavors (perhaps f273 = dagger and f290 = dirk). If the player has either, the text is displayed. Notice the pipe separating these Ids – a pipe denotes OR.

What Comes Next

The following chapter covers:

- Publishing your game
- Adding marketing assets
- Making your story discoverable
- Sharing your work with players

Chapter 9

Publishing, Marketing, and Discovery

Completing a story is an important milestone, but an unpublished story has no audience. TiledPlanet is designed to make publishing straightforward while still encouraging thoughtful presentation and meaningful player engagement.

This chapter explains how to prepare your story for release, how it becomes discoverable, and how players encounter it once published.

Preparing a Story for Publication

Before publishing, it is recommended that your story be:

- Fully playable from start to finish
- Tested across all major paths
- Free of dead ends or unreachable outcomes

Publishing is not a declaration of perfection. It is an invitation for players to engage with your work. Iteration is expected, and changes can be made after release.

Marketing Metadata and Presentation

Each published story includes marketing information that influences how it appears across the platform and in search results.

This information typically includes:

- A marketing image
- One or more screenshots
- A short descriptive summary

The description serves as metadata for search engines and listings. It should clearly communicate:

- The premise of the story
- Its tone or genre
- What makes it distinctive

Strong marketing assets improve discoverability and set appropriate expectations for players.

Activating a Game

Publishing a story on TiledPlanet is intentionally simple.

Once a project is marked as active, it becomes publicly visible and accessible to players. This action can be reversed at any time, allowing authors to deactivate a story for revision or removal.

The ease of activation encourages creators to publish early and improve through real player interaction.

The Proving Grounds

Newly published stories appear in the **Proving Grounds**.

The Proving Grounds serve as an evaluation space where new content gains visibility based on player engagement rather than prominence alone. Metrics such as:

- Player activity
- Story completion
- Repeated play

help determine how a story is surfaced within the platform.

This system rewards stories that are not only interesting, but also playable and complete.

Landing Pages and Player Interaction

Each story has a dedicated landing page.

The landing page allows players to:

- Read a description of the story
- View screenshots
- See ratings and reviews

- Launch the game

Landing pages are shareable and serve as the primary hub for player interaction with a story.

Players must be registered to leave reviews, ensuring feedback comes from engaged participants.

Direct Links and Sharing

In addition to discovery within the platform, each story has a direct web address that can be shared externally.

Direct links are useful for:

- Sharing with friends or collaborators
- Posting on social media
- Including in portfolios or project pages

These links provide a streamlined way for players to access your story without navigating listings manually.

Iteration After Release

Publishing is not the end of development.

After release, authors may:

- Update story content
- Adjust balance or logic
- Replace media assets
- Refine presentation

Changes take effect without requiring republishing. This allows stories to evolve naturally over time.

Responsible Publishing

When creating tutorial projects or experimental content, authors should take care to avoid unnecessary clutter in public listings.

Stories that are not intended for general play can be deactivated when no longer needed.

Thoughtful publishing contributes to a healthier creative ecosystem for all creators and players.

What Comes Next

With your story published and visible, the final step is understanding how to design effectively within the system's constraints.

The next chapter discusses:

- Structural limitations
- Creative workarounds
- Advanced design patterns

Chapter 10

Designing Around Constraints

Every creative system has limits. Rather than restricting creativity, constraints often provide the structure necessary for thoughtful, elegant design. TiledPlanet is no exception.

This chapter explains the known limitations of the platform and, more importantly, the design strategies that allow you to work within them effectively.

The Six-Option Limit

Each story node may present a maximum of six options, including hidden options.

While this may initially feel restrictive, it encourages clarity and intentional design. Most meaningful decisions rarely require more than a handful of choices at once.

When additional options are needed, authors can extend choice space by:

- Using intermediate nodes
- Looping back to prior nodes
- Revealing options conditionally through flags

The result is a controlled expansion of possibility without overwhelming the player.

Extending Choice Through Structure

One common technique is to separate *presentation* from *selection*.

For example:

- A node may present a single option such as “Consider your options”
- The following node presents up to six actionable choices
- Selected actions loop back, with flags hiding previously chosen options

This approach allows:

- Large sets of actions
- Persistent environments
- Exploration-style gameplay

without exceeding the option limit.

Looping with Purpose

Loops are most effective when paired with state tracking.

Because the “options can only be selected once” rule applies only to direct child nodes, looping designs often rely on flags to remove options that have already been taken.

This gives authors full control over:

- Which actions remain available
- Which actions disappear
- How progress is communicated

Loops should always move the story *forward*, even when the player revisits familiar spaces.

Chapter Boundaries and Navigation

Chapters are organizational tools, not linear mandates.

A chapter:

- Groups related nodes
- Simplifies large diagrams
- Improves author clarity

Nodes may jump to any chapter, but always arrive at that chapter’s initial node. This ensures predictable entry points and prevents fragile cross-chapter dependencies.

Chapters do not need to follow a numerical or chronological order. They are best thought of as *story regions* rather than acts.

Embedded Text Logic Constraints

Embedded Text Logic (ETL) is intentionally constrained to maintain clarity and performance.

ETL statements:

- May reference flags *or* items, but not both
- Must use either AND or OR logic, not a combination
- Evaluate independently

To handle complex conditions, authors can stack multiple ETL statements without ELSE text, allowing text to appear only when specific conditions are met.

This approach rewards disciplined story design, where mutually exclusive states are enforced through progression rather than logic alone.

Designing for Mutual Exclusivity

When multiple ETL statements exist in a single block of text, all matching statements will render.

To avoid unintended combinations:

- Ensure that story paths lead to exclusive states
- Use flags to lock out conflicting outcomes
- Design item acquisition carefully

Well-designed stories make impossible states unreachable, rather than filtering them after the fact.

Constraints as Creative Signals

Limitations often reveal where structure is needed.

If a story feels constrained by:

- Option limits
- Chapter boundaries
- ETL rules

it may indicate:

- Too many choices presented at once
- Insufficient reconvergence

- Unclear story state design

In these cases, constraints act as guides rather than obstacles.

Thinking Like a Systems Designer

Interactive storytelling blends narrative craft with systems thinking.

Strong stories:

- Anticipate player behavior
- Respond consistently to choice
- Balance freedom with direction

The most effective TiledPlanet stories are those where structure quietly supports immersion, allowing the player to focus on experience rather than mechanics.

Final Thoughts

You now understand:

- How stories are structured
- How choices shape flow
- How state influences outcomes
- How to publish and share your work
- How to design within constraints

What remains is practice.

Chapter 11

Go Write Your Story

The tools are in your hands.

Start small. Experiment freely. Learn from player behavior. Iterate often.

Interactive stories are not written once — they are *grown*.

When you are ready, resources are available:

- Community forums
- Direct support
- Collaborative spaces

Most importantly, write something you would want to play.