



Co-op Narrative System: Master Reference Document



Overview

A scalable, fully modular, **split-screen co-op narrative engine** designed for two gamepad players. Players can navigate asynchronous storylines, interact with dialogue choices, and trigger world events — all driven by JSON narrative data.



Core Gameplay Design

- Split-screen (side-by-side) narrative experience.
 - Each player navigates their own text + choices.
 - Choices can dynamically alter state, trigger animations, or affect the other player.
 - Includes sync points requiring both players to advance together.
 - Uses JSON for narrative data — writer-friendly, designer-flexible.
-



SCENE STRUCTURES



Main.tscn

text

CopyEdit

Main (Node2D)

|— Panel

Optional background or layout

```

└─ PlayerUIPanel (Player 1)      # Instance of player_ui_panel.tscn
  └─ Cursor1 (Sprite2D)          # Player 1's custom cursor
└─ PlayerUIPanel2 (Player 2)    # Instance of player_ui_panel.tscn
  └─ Cursor2 (Sprite2D)          # Player 2's custom cursor
└─ SharedControlPanel            # Shared buttons (Map, Inventory,
etc.)
└─ DialogueSystem                # Dialogue manager (Node)
└─ TriggerDispatcher             # Trigger router (Node)

```

player_ui_panel.tscn

text

CopyEdit

```

PlayerUIPanel (Control)
└─ Panel                        # Background/styling
└─ VBoxContainer
  └─ ScrollContainer
    └─ RichTextLabel            # Dialogue history
    └─ CurrentTextLabel          # Current line of dialogue
    └─ ChoiceContainer           # Where ChoiceButtons get added
    └─ ContinueButton            # (Optional) Advance without choices
    └─ WaitingLabel              # UI shown during sync points

```

ChoiceButton.tscn

text

CopyEdit

```

ChoiceButton (Control)
└─ Background (NinePatchRect)  # Styled background
└─ Label (Label)                # Choice text
└─ AnimationPlayer              # Handles hover, chosen, denied

```

SharedControlPanel.tscn

text

CopyEdit

SharedControlPanel (Control)

```
|— HBoxContainer
|   |— MapButton (ChoiceButton)
|   |— InventoryButton (ChoiceButton)
|   |— SettingsButton (ChoiceButton)
```

SCRIPTS & PURPOSE

Cursor2D.gd

Attached to: `Cursor1`, `Cursor2`

Purpose: Controls per-player cursor logic.

- Reads left stick for movement, right stick for scroll.
 - Detects and interacts with `Interactable` nodes.
 - Enforces `owner_player_id` rules.
 - Calls `on_cursor_hover()` and `on_cursor_interact()`.
-

Interactable.gd

Base Class for: All interactable UI elements (e.g. `ChoiceButton`)

Purpose: Abstracts cursor interaction and player ownership.

- Exported `owner_player_id` (0, 1, or -1 for shared).
- Methods: `on_cursor_hover(cursor)`, `on_cursor_interact(cursor)`.
- Emits signals: `hovered`, `interacted`, `denied`.
- Utility: `_is_owned_by(player_id)`.

✓ **ChoiceButton.gd**

Attached to: `ChoiceButton.tscn`

Purpose: A custom, stylable choice button.

- Inherits from `Interactable`.
- Exports `choice_id` and uses `AnimationPlayer` for feedback.
- Emits `chosen(choice_id, player_id)` when selected.
- Uses animations: `hover`, `chosen`, `denied`.

✓ **DialogueSystem.gd**

Attached to: Node in `Main.tscn`

Purpose: Central controller for parsing and delivering JSON dialogue.

- Loads `sample_narrative.json`.
- Sends `text_p1/text_p2` to `PlayerUIPanels`.
- Spawns and connects `ChoiceButton` instances.
- Emits `node_loaded` and `trigger_fired`.
- Handles sync points and `triggers`.

✓ **TriggerDispatcher.gd**

Attached to: Node in `Main.tscn`

Purpose: Routes narrative `triggers` to gameplay effects or UI animations.

- Handles predefined triggers like "reveal_map", "give_item_sword".
 - Can show buttons, grant inventory, or change game state.
 - Easily expandable.
-

✓ GameState.gd

Autoload Singleton

Purpose: Stores player choices, inventory, and world state flags.

gdscript

CopyEdit

```
player_data = {
    0: { "choices": [], "inventory": [], "location": "start" },
    1: { "choices": [], "inventory": [], "location": "start" }
}
global_flags = {
    "map_opened": false,
    "sync_locked": false
}
```

- Methods: add_choice(), has_choice(), set_flag(), get_flag(), reset()
-

✖ JSON FORMAT FOR WRITERS

✓ Example Node

json

CopyEdit

```
"start": {
    "text_p1": "Where are we?",
    "text_p2": "It's dark here..."
}
```

```

"choices_p1": [
  { "text": "Look around", "next": "look" },
  { "text": "Stay quiet", "next": "wait" }
],
"choices_p2": [
  { "text": "Draw weapon", "next": "draw" }
],
"sync_point": false,
"triggers": ["reveal_map"]
}

```

Supported Keys:

- `text_p1` / `text_p2`: Per-player text.
- `choices_p1` / `choices_p2`: Optional choices.
- `next`: Destination node ID.
- `sync_point`: Wait for both players to continue.
- `triggers`: Trigger IDs for gameplay/UI actions.

Scalability Notes

Feature	Scalable?	How
Narrative growth	✓	Add more JSON nodes and branches
Visual polish	✓	All UI is animation-friendly
Game state reactions	✓	Use <code>TriggerDispatcher</code> or extend <code>GameState</code>
Multiplayer syncing	↔	Easy to layer on top of this with RPC or net sync

Writer autonomy	✓	Fully JSON-driven with a clean spec
Shared menus	✓	<code>SharedControlPanel</code> accepts shared buttons (owner_id = -1)

Writer Workflow Summary

1. Copy template node.
 2. Write text, choices, next node.
 3. Add triggers or sync points if needed.
 4. Validate JSON via jsonlint.com.
 5. Submit via Git with changelog notes.
-

Final Notes

This system is:

- Fully modular (each piece pluggable or swappable)
- Built for designer/writer collaboration
- Gamepad-first, co-op aware
- Extensible for future gameplay systems (inventory, combat, world exploration)