

FRAMING BY DESIGN PROJECT

DOCUMENTATION:

Framing by Design is primarily designed to teach players problem-solving skills and how to collaboratively work with others to tackle various issues.

The project currently consists of 4 major systems, the dialogue system, an overworld system, a notebook UI, and a data system. This document will explain how to play the game, as well as how each of these systems work in-detail.

If you would like further, more detailed insight, see the code for developer comments. Additionally, this project was made using the Unity Engine, which has its own documentation that may be helpful.

How to run the project and play the game (For playtesters):

This project was built and ran using a Windows 11/10 machine.

To run the game, run the included.exe file. Alternatively, open the project via Unity Hub using the latest editor version and play the game in the editor. In this build of the game, the goal is to speak with all characters currently implemented. The game should score you properly based on your performance in the interviews and allow you to collect insights regarding the problem.

The game is played with the following controls:

- W - Move Up
- A - Move Left
- S - Move Down
- D - Move Right
- Spacebar - Progress through dialogue
- I - Interact with characters
- E - Interact with overworld objects
- L - Log and export your current playthrough history and stats (If player consented to such)

--How to retrieve player logs--

When a log is exported, it can be viewed through the following filepath:

C:\Users\(\Put your username here)\AppData\LocalLow\DefaultCompany\Framing By Design

the log file will be titled something like:

"dialogue_log_793b9299-218d-4ee1-a7e0-eece5ddde42b.csv"

The Dialogue System:

The dialogue system acts as the primary method through which the player can interact with other characters. It allows for branching narratives, the tracking of player choices, and a visually satisfying gameplay experience. It also allows the player to unlock badges based on their unique problem-solving playstyle. Data from the dialogue system is exported to a log, which can be used to evaluate the player's performance in developing problem-solving skills.

Its goals:

1. Load structured dialogue content from designer-authored CSV files.
2. Drive the UI presentation (NPC lines, player dialogue, choices, feedback).
3. Coordinate branching logic (NextNode navigation, choice outcomes, failure conditions).
4. Track user decisions for:
 - Game scoring
 - Badge progression and badge unlocks
 - Logging for data analytics
5. Control dialogue session lifecycle (start, line-by-line navigation, close).

This system is designed to be declarative; all content and branching behavior come from CSV files, so building dialogue flows can occur outside of code.

Class: DialogueDatabase

Responsibility: Parse CSV files into a structured list of DialogueLine objects.

Key Features

- Accepts multiple CSV files (TextAsset[] csvFiles).
- Processes each CSV row into a strongly structured DialogueLine.
- Supports fields such as NPC lines, player lines, choices, scoring, feedback, badges, and branching via NextNode.

- Provides API methods:
 - GetLinesByNode(nodeId)
 - GetLinesByNPC(npcId)

CSV Parsing Rules

- CSVs are manually parsed to support quoted text with commas.
- Each row is expected to contain 21 fields; missing fields are padded to avoid crashes.
- All strings are trimmed to avoid whitespace mismatches when referencing nodes.

Class: DialogueLine

Represents a single dialogue row from a CSV file.

Each line may represent:

- An NPC line (NPC speaks)
- A choice line (player chooses)
- A player intro (short one-off player line before NPC speaks)
- Metadata for scoring, badges, IRI flags, feedback types, etc.

Table of important Line properties:

Field	Purpose
NodeID	Identifier grouping multiple dialogue lines under a single state of dialogue.
NPCLine	What the NPC says at this node (optional).

ChoiceID	If non-empty, this row represents a player choice option.
PlayerText	The text shown when the player picks this choice.
Score	Parsed integer score (e.g. "+2", "-1").
NextNode	The subsequent node after this line or choice.
FeedbackText	Text shown after a choice is made, before moving on.
BadgePath	Path indicating badge progression category.
InsightUnlockID	Marks points where insight unlocks occur.
NPC_Emootional State_After	Emotional state after player's choice (for logging).

Multiple DialogueLine rows can exist under a single node

- One row with NPCLine (optional)
- Zero or one PlayerIntro
- Several rows representing player choices

Class: DialogueManager

Orchestrates the actual execution of dialogue using content provided by DialogueDatabase and displaying it through DialogueUI.

Primary Capabilities:

- Load a node and determine what should be displayed.
- Advance through dialogue via Space key or through UI callbacks.
- Show NPC text (supports multi-page chunking for long lines).
- Display player choices.
- Handle scoring and badge progression.
- Send events to the analytics/logging subsystem.
- Manage failure nodes and trigger retry conditions.

Class: DialogueUI

The DialogueUI script has responsibilities mainly tied to the visual representation of the dialogue system:

- Show NPC lines with typewriter effect
- Show player lines
- Show feedback
- Show choices list
- Manage visibility of UI containers
- Provide callback when typing is finished
- Indicate typing state - IsTyping()

Class: BadgeManager & BadgeData

The BadgeManager class tracks badge progression paths per NPC. Badge unlocks can also be retrieved for analytics logging.

Badges are defined in the BadgeData script. The badge's unique ID can be retrieved for tracking purposes. Important badge properties include:

- The NPC it is tied to
- The name of the badge
- The required score to unlock the badge
- The player's current progress toward earning the badge
- Whether or not the badge has been unlocked
- The badge's icon
- The badge's description

The Overworld System:

The overworld system is how the game works outside the main logic of the game, how the player can begin to interact with NPCs/objects, being able to move, teleport between rooms, sprites used for the game, etc. This is the basic feel and how the player is able to actually play the game.

Goals:

1. Allow for player movement throughout the game
2. Allow for player to start the interaction with NPCs or objects
3. Allow the player to move between different rooms and sections of the map as he wishes
4. Showcase player score
5. Manage Intro screen to ask user to log choices
6. Allowing camera to follow player around the game

Class: Movement

As the name states this class allows the player to move around the map using regular A,W,S,D keys or arrow keys. It also checks to see if there is an object in front of the player by casting an invisible box in front of the player, if it collides the player is not able to move past that area.

Class: StartConversation

This class allows the player to interact with NPC by clicking the letter I when the player is in range. It communicates with the Dialogue Manager to enable it to showcase the Dialogue UI with the correct dialogue, NPC name and starting node for each NPC.

Class: Portal

In the game the player has to move through many different rooms to be able to communicate with all the different NPCs, and in order to travel between each room layout. To do so there is the Portal class which uses GameObjects as portals and entrances. GameObjects categorized as Portals are the door the player takes, while entrances are where the player will appear on the other side of said door.

Class: ScoreManager

Using the score given to each Node of the CSV the player can get a +2, +1, 0, or -1 depending on the choice they receive. The ScoreManager class just displays the overall player score from all the conversations, even when not talking with an NPC, this allows the player to view if their choice is a good one or not

Class: IntroManager

The IntroManager class manages the title screen, and the permission screen for the game. This allows the player to get ready instead of just being dropped into the game without any warnings. Also in order to log all the choices the player made during their game we need confirmation from the player that they allow this, so a permission screen appears with a Yes or No button after the title screen. The IntroManager manages all of this and sends a confirmation to the DialogueLogManager to store the values.

Class: CameraFollow

This class is implemented to the main camera and it follows the player around, so that the camera isn't stuck in just one place. The class takes in a GameObject called target, and it will follow that "target" while the game is being played. In this case it is the player.

The Notebook UI:

Upon clicking the notebook, the player should see their collected insights.

Class: InsightManager & InsightDefinitions

InsightDefinitions acts a database for the insights the player can collect. InsightManager defines the Insights struct and the InsightManager which handles adding and returning collected insights to the notebook

Class: NotebookUI

This script defines the appearance of the notebook and lists collected insights.

The Data System:

Upon pressing L, the player's data will be logged and exported to a csv if they offer their consent. In order to do this, the game has two scripts: DialogueLogTypes and DialogueLogManager.

Class: DialogueLogTypes

Defines the columns the game exports when logging player data.

Class: DialogueLogManager

Primarily handles csv exporting and checking if the player consented to data logging. This script also generates a player ID and session ID in order to differentiate different players and their different sessions.