Visual Novel Game Engine

PHASE 3: APPLICATION LAYER & INTERFACE

Ethan Hom & Cristian Parker

System Description

- A "Content Management System" (CMS) for interactive storytelling.
  - Replaces hard-coded game scripts with relational database entries.
- Traditional game dialogue is often hard-coded, making changes difficult.
- Managing assets (images) and logic links manually is error-prone.
- Abstraction: An interactive GUI that "hides" all SQL operations.
- Dynamic Rendering: The game engine fetches dialogue, backgrounds, and sprites in real-time from the database.
- State Tracking: Persistently tracks player choices and game events.

Use Case Description (The Designer)

- User Role: The Designer (Backend)
- Interaction Type: Write / Modify (CRUD)
- Key Capabilities:
  - Asset Management: Uploading images for Locations and Sprites (stored as file paths).
  - Character Creation: Defining characters and text colors via INSERT statements.
  - Script Writing: Sequencing dialogue lines using a visual editor.
  - Logic Design: Creating branching paths (Choices) that link to different Scenes.

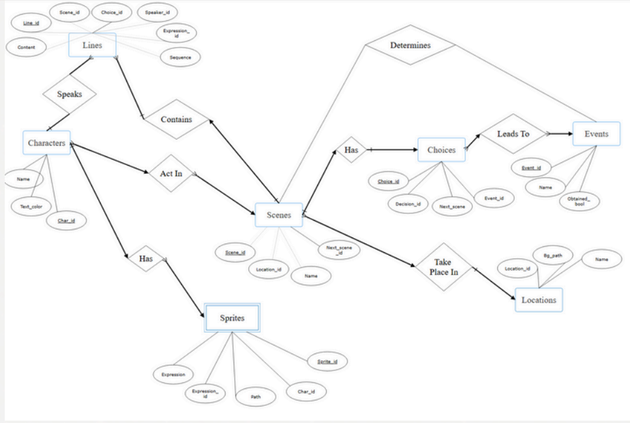
Use Case Description (The Player)

- User Role: The Player (Frontend User)
- Interaction Type: Read-Only & State Updates
- Key Capabilities:
  - Visual: The engine queries Scenes and Lines to render the correct background and sprite.
  - Progression: Clicking 'Next' triggers SELECT queries to fetch the next line in the sequence.
  - Decision Making: Clicking a Choice button triggers an UPDATE to the Events table.
  - Persistent State: The database remembers which events (flags) have been triggered.

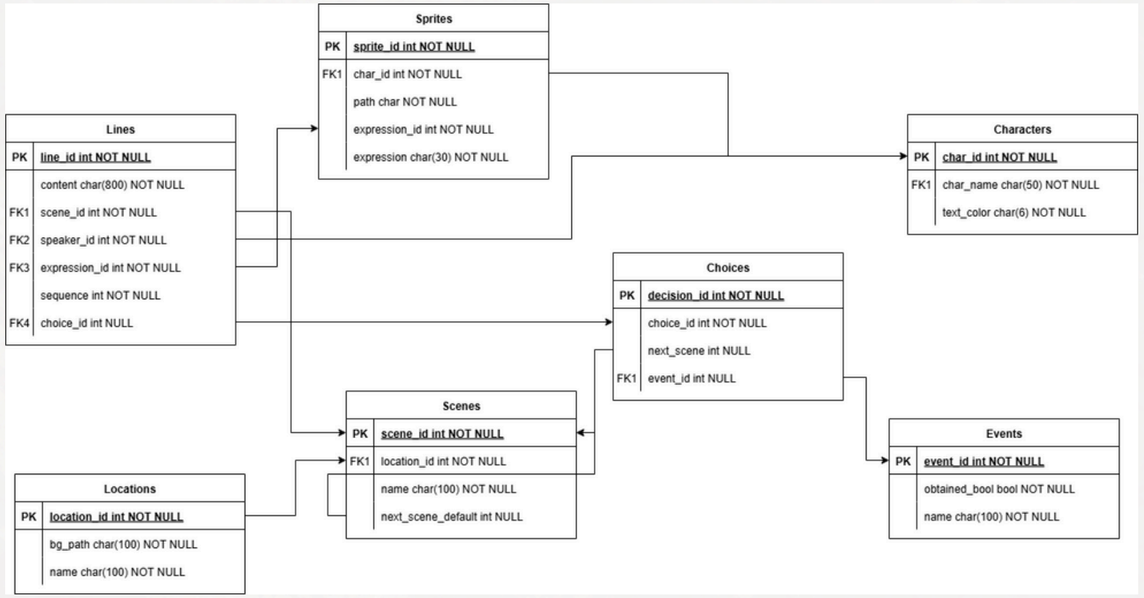
Implementation Details

- Tech Stack:
  - Database: SQLite (Relational integrity, Foreign Key support).
  - Backend: Python 3.11 (Logic layer, File I/O).
  - Frontend: NiceGUI (Web-based interface).
- Key Technical Features:
  - Queries: All user input have “???” placeholders to prevent SQL Injection.
  - Static Asset: Images are stored on disk; DB stores relative paths; Application serves absolute paths.
  - Dynamic Dropdowns: Frontend menus auto-populate by querying the DB (e.g., Speaker list updates automatically).

Database Design



- E/R Diagram (Outdated)
- Relational Schema (Outdated)
- Key Structure:
  - Central Entity: Scenes (Connects Locations, Lines, and Choices).
  - Normalization: 8 Tables to ensure data integrity.
  - Relationships:
    - Lines -> Characters (Foreign Key)
    - Lines -> Sprites (Foreign Key)
    - Choices -> Events (Foreign Key)



Database Design

Characters
- char_id (PK)
- char_name
- text_color

Sprites
- sprite_id (PK)
- char_id (FK → Characters)
- expression
- path

Locations
- location_id (PK)
- name
- bg_path

Scenes
- scene_id (PK)
- name
- location_id (FK → Locations)
- next_scene_default

Lines (Dialogue Script)
- line_id (PK)
- scene_id (FK → Scenes)
- speaker_id (FK → Characters)
- sequence (order in scene)
- content (dialogue text)
- sprite_id (FK → Sprites, optional)
- choice_id (optional)

Choices
- decision_id (PK)
- choice_id (group ID)
- decision_text
- next_scene (FK → Scenes)
- event_id (FK → Events, optional)

Events
- event_id (PK)
- name
- obtained_bool (true/false)

EventLogic (Conditional Branching Rules)
- logic_id (PK)
- start_scene (FK → Scenes)
- end_scene (FK → Scenes)
- event_id (FK → Events)