Level Editor Plugin Documentation

Overview

The Level Editor plugin is a Unity tool designed to streamline level creation, allowing developers to customize levels by configuring environments, particle effects, sound effects, and gameplay logic. The plugin includes an editor window, a level setup system, and various gameplay management scripts.

Setup Instructions

1. Installation

- Place the `LevelEditorPlugin` folder inside the `Assets/Plugins/` directory of your Unity project.
- Ensure that all scripts and assets are under `Assets/Plugins/LevelEditorPlugin/`, which includes `Editor` and `Runtime` folders.

2. Initial Project Configuration

1. **Add Required Tags and Layers**:
    - Ensure your project includes tags such as `Background` (for the background object).
    - Layers should be set for environment objects and obstacles for proper collision detection.
2. **Prepare UI Elements**:
    - The plugin expects certain UI elements to be in place, such as:
        - A `TextMeshProUGUI` object named `Question` for displaying the level question.
        - A `Transform` object named `Wordset` for holding word buttons.
    - You can use `TMP` components, which can be added by installing the TextMesh Pro package from the Unity Package Manager.
3. **Configure Audio Sources**:
    - Ensure that a `GameObject` with an `AudioSource` component is available for playing sound effects in the game.

Using the Level Editor

Opening the Level Editor Window

1. Go to `Tools > Level Editor Plugin > Level Editor` in the Unity menu to open the main Level Editor window.
2. The window will display a series of sections (Level Settings, Environment Management, Particle System Prefabs, Sound Effects) to allow for full level configuration.

Creating and Configuring Levels

1. **Creating a New Level**:
    - In the Level Editor window, click on the **Create New Level** button.
    - Specify a name and save location for the `LevelData` asset in the save panel.
    - The new `LevelData` asset will appear in the editor for further configuration.
2. **Editing an Existing Level**:
    - Use the **Level Data** field to select an existing `LevelData` asset.
    - Once selected, the editor will populate with settings specific to the level.
3. **Configuring Level Settings**:
    - **Level Name**: Enter a name for the level.
    - **Background Image**: Assign a background sprite to appear in the level.
    - **Question Text**: Provide text that will display as the question or prompt for the level.
    - **Animated Scene Prefab**: Assign a prefab to display an animation upon level completion.
4. **Adding and Configuring Words**:
    - **Words**: Click **Add New Word** to add a word button that players can select. Each word can be edited directly in the list.
    - **Correct Words**: Click **Add Correct Word** to designate specific words as correct answers.
5. **Configuring Particle Effects and Sound Effects**:
    - **Particle System Prefabs**: Define particle effects to display upon correct or incorrect selections.
    - **Sound Effects**: Define sound clips for various actions. Add new sound entries with names and corresponding `AudioClip`s.
6. **Saving the Level Configuration**:
    - Once configured, click **Save Level Settings** to save changes to the selected `LevelData` asset.
7. **Previewing Level Changes**:
    - To see a preview of the configured level, click **Preview/Update Level**. This will display the level with the specified settings in real-time.

Managing Environment Objects

1. **Environment Management Section**:

    - The Level Editor allows you to add, replace, or delete environment objects such as ground, obstacles, or decorations.
    - **Add**: Click the **Add [Environment Name]** button to instantiate an environment object.
    - **Replace**: Select an object in the scene and click **Replace with [Environment Name]** to replace it.
    - **Delete Selected**: Deletes the currently selected environment object in the scene.
    - **Delete All Environments**: Removes all environment objects from the scene.

In-Depth Script Explanation

1. LevelEditorWindow.cs

The `LevelEditorWindow.cs` script creates the main editor window for managing level configurations. It includes:

- **UI Elements**: Buttons and fields for creating new levels, selecting `LevelData` assets, and configuring various sections (Level Settings, Environment Management, etc.).
- **Sections**: Each foldable section corresponds to a category in level editing (e.g., particle systems, sound effects).
- **Save and Preview Functionality**: The `DrawSaveButton` and `DrawPreviewButton` methods enable users to save or preview their level setup.

2. UIManager.cs

This script manages the user interface components for level configurations within the editor:

- **DrawLevelSettings**: Displays level properties like level name, question text, and background image.
- **DrawWordManagement**: Provides controls for adding and managing word buttons in the level.
- **DrawCorrectWords**: Allows users to specify correct words for the level.
- **DrawParticlePrefabsList** and **DrawSoundEffectsList**: Handle the addition and management of particle effects and sound effects.

3. EnvironmentManager.cs

Handles the management of environment objects within the level:

- **InstantiateEnvironment**: Adds an environment object to the scene based on a specified prefab.
- **ReplaceEnvironment**: Replaces a selected environment object with a new prefab.
- **DeleteEnvironment**: Deletes the selected environment object.
- **DeleteAllEnvironments**: Clears all environment objects in the level.

4. LevelPreview.cs

The `LevelPreview.cs` script provides real-time preview capabilities for level configurations:

- **ApplyQuestionText**: Displays the question text on a UI element named `Question`.
- **InstantiateWords**: Instantiates word buttons in the scene, based on the `LevelData` configuration.
- **InstantiateAnimatedScene**: Creates an animated scene prefab in the editor for preview purposes.

5. LevelData.cs

The `LevelData` scriptable object stores all data required for each level, including:

- **Words and Correct Words**: Lists that define words for selection and correct answers.
- **Particle and Sound Effects**: Lists of particle and sound effect prefabs to enhance gameplay feedback.
- **Placed Environments**: Stores position and rotation data for environment elements placed in the level.

6. EnvironmentData.cs

Defines data for each environment element, such as ground or obstacles:

- **Prefab**: A reference to the environment prefab to instantiate.
- **EnvironmentType**: An enum that classifies each environment type, like Ground or Obstacle.

7. EventManager.cs

Manages game events and broadcasts when correct or incorrect words are selected:

- **OnCorrectWordSelected** and **OnIncorrectWordSelected**: Triggered when a word is selected.
- **OnAllCorrectWordsSelected**: Signifies the completion of correct selections, triggering level completion events.

8. AudioManager.cs

Handles sound effects in the game using a singleton pattern:

- **PlaySFXByName**: Plays a specific sound clip by name from `LevelData`.

9. CharacterController.cs

Controls character movement, starting movement when all correct words are selected:

- **StartMoving**: Triggers movement in response to the `OnAllCorrectWordsSelected` event.

10. CharacterMovement.cs

Handles the core movement logic:

- **MoveForward**: Moves the character forward.
- **DetectObstacleAndJump**: Detects obstacles and initiates a jump.
- **CalculateJumpForce**: Determines the necessary jump force to clear obstacles.

11. GameController.cs

Manages the main gameplay logic for each level:

- **SetupLevel**: Configures level settings based on `LevelData`.
- **OnWordSelected**: Checks selected words, updating UI and triggering events.
- **NextLevel**: Loads the next level scene.

Best Practices for Using the Plugin

1. **Organize Levels with Scriptable Objects**:
   - Store each level's `LevelData` as a separate asset for easy management and versioning.
2. **Use Preview Frequently**:
   - Use the Preview functionality to quickly verify level configurations and make adjustments as needed.
3. **Check for Missing Resources**:
   - Ensure that all referenced assets, such as particle effects, sound effects, and environment prefabs, are available in the project to avoid missing references.
4. **Test Levels in Editor**:
   - Run levels in the Unity Editor to confirm that all gameplay events, such as word selection and level completion, function as expected.
5. **Customize Based on Requirements**:
   - Extend or adjust scripts like `GameController` or `CharacterController` if additional gameplay behaviors are required.