Save Custom Game

Documentation

Creator: Lucas Gomes Cecchini

Online Name: AGAMENOM

Overview

This document will help you use **Assets Save Custom Game** for **Unity**.

With it you can save your game in several Slots.

It will take a **screenshot** of the game and there are three ways to save in **GameData(.json)**, **LocalLow(.json)** and **PlayerPrefs**.

It also has Auto Save features based on **Time** or **Events**.

Instructions

You can get more information from the Playlist on YouTube:

https://youtube.com/playlist?list=PL5hnfx09yM4JDFFdX1vfpNCLLdLq4ixZA&si=3w1IN0WrpV8lFMrS

Script Explanations

Class

SaveDataUtility

This script, **`SaveDataUtility`**, contains a set of static methods that help in retrieving, setting, and managing various types of data within a **`SaveCustomObject`**. Here's a breakdown:

Variables:
- **saveObject:** Represents an instance of **`SaveCustomObject`**.
- **itemTag:** A string identifier for items within the **`SaveCustomObject`**.
- **floatTag**, **intTag**, **stringTag**, **boolTag:** String identifiers for specific values within items.
- **newValue:** Represents the new value to be set.
- **targetCamera:** A `Camera` object used for capturing screenshots.
- **screenshot:** A byte array used to store encoded screenshot data.

Methods:
1. **GetFloat**, **GetInt**, **GetString**, **GetBool:**
   - **Usage:** Retrieve specific types of values (float, int, string, bool) from **`SaveCustomObject`** based on item and tag identifiers.
   - **Parameters: `saveObject`** is the **`SaveCustomObject`** instance, and tags identify specific values within items.
   - **Returns:** The value of the specified type based on the tags provided.

2. **SetFloat**, **SetInt**, **SetString**, **SetBool:**
   - **Usage:** Set specific types of values (float, int, string, bool) within **`SaveCustomObject`** based on item and tag identifiers.
   - **Parameters: `saveObject`** is the **`SaveCustomObject`** instance, and tags identify specific values within items**. `newValue`** is the value to be set.
   - **Returns:** No return; these methods set values within the **`SaveCustomObject`**.

3. **CaptureScreenshot:**
   - **Usage:** Capture a screenshot using a specified camera and assign it to **`SaveCustomObject`**.
   - **Parameters: `saveObject`** is the **`SaveCustomObject`** instance, and **`targetCamera`** is the camera used for capturing the screenshot.
   - **Returns:** No return; it assigns the captured screenshot to the **`SaveCustomObject`**.

4. **RenderScreenshot:**
   - **Usage:** Convert a byte array representation of a screenshot into a **`Texture2D`**.
   - **Parameters: `screenshot`** is the byte array containing the screenshot data.
   - **Returns:** A **`Texture2D`** object converted from the byte array.

5. **TextureToSprite:**
   - **Usage:** Convert a **`Texture2D`** object into a **`Sprite`**.
   - **Parameters: `texture`** is the **`Texture2D`** object to be converted.
   - **Returns:** A **`Sprite`** created from the provided **`Texture2D`**.

6. **GetComponentSaveCustomInScene:**
   - **Usage:** Find and assign the **`SaveCustomInScene`** component if found in the scene.
   - **Parameters:** A reference to the **`SaveCustomInScene`** component.
   - **Returns:** No return; it assigns the found component to the provided reference.

7. **EnableAutoSave, DisableAutoSave:**
   - **Usage:** Enable or disable auto-saving by toggling the **`autosaveEnabled`** flag in **`SaveCustomObject`**.
   - **Returns:** No return; it modifies the **`autosaveEnabled`** flag in the **`SaveCustomObject`**.

8. **SaveEvent:**

- **Usage:** Trigger an autosave event by calling **`SaveAutoGame()`** on the **`AutoSaveCustom`** component.
- **Returns:** No return; it triggers the autosave event if the necessary component is found.

Exception Handling:
- These methods handle exceptions using **`KeyNotFoundException`** to indicate when requested tags or items are not found within the **`SaveCustomObject`**.

Overall:
The script provides a set of utilities to manipulate and manage data within a custom save object, handling exceptions when tags or items are not found and enabling auto-saving functionality. It also includes functionalities for capturing and rendering screenshots within the game environment.

ExceptionUtility

This **`ExceptionUtility`** class provides a method, **`GetCallingMethodInfo()`**, which retrieves information about the method call in terms of its file name, line number, and relative file path within the project's assets. Let's break it down:

Variables:
- **stackTrace:** A stack trace object used to capture method call information.
- **frames:** Stack frames obtained from the stack trace.
- **foundSaveDataUtility:** A flag to track encountering methods from `SaveDataUtility`.

Methods:
1. **GetCallingMethodInfo:**
   - **Usage:** Retrieves file name, line number, and relative file path information for the calling method.
   - **Returns:** A formatted string containing the file path and line number of the method call.

Explanation:
1. **Initialization:**
   - It creates a **`StackTrace`** object to capture method call information and retrieves the stack frames.

2. **Stack Frame Iteration:**
   - It iterates through each frame in the stack trace.

3. **Frame Analysis:**
   - For each frame, it obtains method information and its declaring type.
   - Checks if the method is neither from **`SaveDataUtility`** nor **`ExceptionUtility`**. If so:
     - Retrieves the file name and line number information from the stack frame.
     - Modifies the file path to make it relative to the project's Assets folder.
     - Formats and returns the method call's file path and line number.
   - Tracks encountering methods from `SaveDataUtility`.
   - Stops processing when encountering methods from **`ExceptionUtility`** after encountering methods from **`SaveDataUtility`**.

Overall:
This utility provides a method to fetch information about the calling method's file name, line number, and relative file path within the project's Assets folder. It excludes calls originating from **`SaveDataUtility`** and **`ExceptionUtility`** to focus on the actual calling context within the project's codebase. This information is useful for debugging purposes and pinpointing the exact location of method calls within the project's files.

Components

SaveCustomInScene

This script manages the saving and loading of game data in Unity, including screenshots, game time, and custom items. It also supports resetting the save data.

Variables:
- **`saveCustomObject`**: Reference to the **`SaveCustomObject`** instance used for managing game data.
- **`gameTime`**: Current in-game time formatted as a string.
- **`fileName`**: Default name for the save file.
- **`savePath`**: Path where the save file will be stored.
- **`sceneName`**: Name of the current scene.
- **`sceneCamera`**: Reference to the camera used for capturing the scene.
- **`elapsedTime`**: Time elapsed since the start of the game.

Methods:
- **`ResetSave()`**: Resets the save data by clearing the elapsed time, screenshot, game time, and scene name in **`saveCustomObject`**.

- **`FixedUpdate()`**: Incrementally updates **`elapsedTime`**, **`gameTime`**, and **`sceneName`** at fixed intervals.

- **`UpdateSaveCustomObject()`**: Updates **`saveCustomObject`** with the current game time and scene name.

- **`UpdateGameTime()`**: Converts **`elapsedTime`** into a formatted in-game time string (**`hours:minutes:seconds`**).

- **`GetCamera()`**: Finds and assigns the appropriate camera for scene capture, defaulting to the main camera if none are found.

- **`SaveData()`**: Captures a screenshot, creates a **`SaveCustomFile`** object with game data, and saves it as a JSON file. It supports saving to a file or **`PlayerPrefs`** based on **`saveCustomObject`** settings.

- **`LoadData()`**: Loads game data from a file or **`PlayerPrefs`** based on **`saveCustomObject`** settings and assigns it to **`saveCustomObject`** or the scene.

- **`LoadDataString(SaveCustomFile data)`**: Loads game data from a **`SaveCustomFile`** object into **`saveCustomObject`** and the scene. It updates **`saveCustomObject`** properties and loads the specified scene.

`SaveCustomFile` Class:
- **Usage**: Represents the serialized data for saving and loading game states.
- **Variables**:
  - **`screenshot`**: Byte array for storing screenshot data.
  - **`gameTime`**: String for storing in-game time.
  - **`sceneName`**: Name of the scene.
  - **`saveCustomItems`**: List of custom items for saving.
  - **`elapsedTime`**: Floating-point value for storing elapsed time.

Usage:
- **Saving Data**: Use the **`SaveData()`** method to capture a screenshot and save the game data as JSON in a specified path or **`PlayerPrefs`**.
- **Loading Data**: Use the **`LoadData()`** method to load saved data from a file or **`PlayerPrefs`** and apply it to **`saveCustomObject`** or the scene.
- **Resetting Save**: Use the **`ResetSave()`** method to clear all save data and reset the state.

Summary:

The `**SaveCustomInScene**` script continuously updates and manages game-related data, including saving and loading states, screenshots, and elapsed time, enabling seamless game state persistence and resumption.

AutoSaveCustom

This script manages automatic saving within the game based on specified intervals and conditions.

Variables:
- `**saveCustomInScene**`: Reference to the `**SaveCustomInScene**` component for saving game data.
- `**currentAutoSaveSlot**`: Keeps track of the current slot for autosaving.
- `**timeSinceLastSave**`: Tracks the time elapsed since the last save.
- `**saveInterval**`: Specifies the time interval between autosaves.

Methods:
- `**Start()**`: Performs initial setup, checks the `**saveCustomInScene**` reference, sets autosave settings, and retrieves the current autosave slot from `**PlayerPrefs**`.
- `**FixedUpdate()**`: Tracks time since the last save and triggers autosave if the set interval is reached and autosave by event is disabled in settings.
- `**SaveAutoGame()**`: Triggers the autosave functionality based on certain conditions:
  - Checks if autosave is enabled.
  - Sets the filename for the autosave based on the current autosave slot.
  - Calls the `**SaveData()**` method from `**SaveCustomInScene**` to perform the autosave.
  - Updates the autosave slot for the next save.
  - Saves the updated autosave slot in `**PlayerPrefs**`.

Usage:
- **Start():** Initializes autosave settings, retrieves the current autosave slot.
- **FixedUpdate():** Constantly tracks time and triggers autosave when the set interval is reached and autosave by event is disabled.
- **SaveAutoGame():** Initiates the autosave functionality based on conditions, using the `**SaveData()**` method from `**SaveCustomInScene**`.

Summary:
This script orchestrates the automatic saving mechanism in the game. It constantly checks time intervals and conditions to perform autosaves, ensuring that the player's progress is regularly saved within the game. The `**SaveAutoGame()**` method is crucial in handling the autosave functionality, ensuring that the game state is preserved periodically based on the specified settings.

TMP_SaveMenuManager and SaveMenuManager

This script, **`SaveMenuManager`**, handles the functionality related to saving and loading game data from different slots, displaying save information, and managing the UI elements for the save menu.

Variables:
- **Save Settings:** References related to saving game data.
- **Confirmation Panel Settings:** UI elements related to the confirmation panel for saving.
- **Title Systems:** Text elements for titles displayed in the save menu.
- **Button Systems:** Buttons and UI elements associated with each save slot.
- **Page Systems:** Buttons for navigating between different pages of save slots.
- **currentSaveNumber:** Tracks the currently selected save slot number.
- **firstTime:** Flags the first-time setup of UI elements.

Methods:
- **OnEnable():** Sets up initial settings, UI elements, and retrieves the last selected save slot number from PlayerPrefs.
- **SetTitle():** Displays different titles based on the current save slot number.
- **SetupButtonsAndInputField():** Sets up navigation buttons, input field, and save buttons for the save menu.
- **SetupInitialSaveName():** Sets up the initial save name based on the selected save slot.
- **UpdateSaveNames():** Updates the displayed save names based on the current save number.
- **IncreaseNumber():** Increments the current save number and updates the displayed save names.
- **DecreaseNumber():** Decrements the current save number and updates the displayed save names.
- **ValidateInput():** Validates input in the input field, allowing only digits.
- **OnEndEditInputField():** Handles the end edit event of the input field.
- **SaveGame():** Saves game data to the specified slot or displays a confirmation panel if the slot is occupied.
- **IsSaveSlotOccupied():** Checks if a given save slot is occupied by checking the associated UI image texture.
- **ClearRawImagesAndTextures():** Clears all RawImage textures.
- **ConfirmSave():** Confirms the save operation after displaying the confirmation panel.
- **CancelSave():** Cancels the save operation by hiding the confirmation panel.
- **LoadData():** Loads data from save slots and updates the UI accordingly.
- **DetermineSavePath():** Determines the save path based on various conditions.
- **RenderImageSlot():** Renders the UI image slot based on the slot number and availability of data.
- **LoadDataString():** Loads data into UI elements based on the slot number and associated data.

Usage:
- **OnEnable():** Initializes and sets up UI elements when the script is enabled.
- **SaveGame():** Manages the saving functionality, either directly or through a confirmation panel.
- **LoadData():** Loads data from save slots and updates the UI to display the information.

Summary:
This script orchestrates the entire save menu system, managing the display of save slots, handling user interactions for saving and loading, and updating the UI elements accordingly. It interacts with the **`SaveCustomInScene`** component to perform actual data saving and loading operations. The script is responsible for managing the UI elements, data validation, and rendering the save information based on the selected save slot.

TMP_LoadMenuManager and LoadMenuManager

This script is a **`LoadMenuManager`** in a Unity game that manages loading saved data into the game. Let's break down its components:

Variables:
- **`saveCustomInScene`**: Reference to **`SaveCustomInScene`** for loading.
- **`confirmationPanel`**: Panel for confirmation.
- **`confirmButton`** and **`cancelButton`**: Buttons to confirm or cancel loading.
- **`titleLoad`**: Text for the load menu title.
- **`text`** and **`textAutomatic`**: Default load text and text for autosave.
- **`loadPath1`** to **`loadPath6`**: Paths for different load slots.
- **`buttonLoad1`** to **`buttonLoad6`**: Buttons for different load slots.
- **`rawImageLoad1`** to **`rawImageLoad6`**: Images for different load slots.
- **`textLoad1`** to **`textLoad6`**: Texts for different load slots.
- **`right`** and **`left`**: Buttons for navigating to the next or previous page.
- **`inputField`**: Field for specifying the load slot.

Methods:
- **`OnEnable()`**: Sets up buttons, input fields, load names, and title when the object is enabled.
- **`SetTitle()`**: Sets the title based on the current load number.
- **`SetupButtonsAndInputField()`**: Sets up listeners for buttons and input fields.
- **`SetupInitialLoadName()`**: Sets the initial load name based on the current load number.
- **`UpdateLoadNames()`**: Updates the displayed names for each load slot.
- **`IncreaseNumber()`** and **`DecreaseNumber()`**: Increments or decrements the load number.
- **`ValidateInput()`**: Validates input characters for the load number field.
- **`OnEndEditInputField()`**: Triggered when the input field editing ends.
- **`LoadGame()`**: Loads the game data for a specific button (load slot).
- **`IsLoadSlotOccupied()`**: Checks if a specified slot for loading is occupied.
- **`ClearRawImagesAndTextures()`**: Clears the images and textures on load slots.
- **`ConfirmLoad()`**: Confirms the loading for a specific slot.
- **`CancelLoad()`**: Hides the confirmation panel when canceling the load action.
- **`LoadData()`**: Loads data from save slots into UI elements.
- **`DetermineLoadPath()`**: Determines the path to locate the saved file.
- **`InteractableSlot()`**: Enables or disables interaction for a specific slot.
- **`LoadDataString()`**: Loads data into UI elements for a specific slot.

Usage:
This script is attached to a game object in Unity. It manages the loading of saved game data into the game environment based on the specified load slots. It interacts with UI elements, sets up buttons for navigation, and handles the confirmation of loading saved data into the game.

The script dynamically updates UI elements like buttons, images, and texts based on the saved data available in the specified slots, allowing users to load their saved game progress.

SaveCustomObject

This script contains classes to manage and create custom object data within Unity Editor, mainly focused on creating and handling `SaveCustomObject` assets.

`CustomObjectDataCreator` Class:
- **Usage:** This class is specific to the Unity Editor and is responsible for creating a custom asset.
- **Methods:**
  - **`CreateCustomObjectData()`:**
    - Creates a custom asset **`SaveCustomObject`** within the Unity Editor.
    - Defines a menu item under **`Assets/Create/Save Custom Game/Save Custom Object Data`** to invoke this method.
    - Checks if the asset exists and prompts the user to replace it if necessary.
    - Creates an instance of **`SaveCustomObject`**, saves it as an asset, and refreshes the Asset Database.

`SaveCustomObject` Class:
- **Usage:** Inherits from **`ScriptableObject`** and represents the custom object data to be saved.
- **Variables:**
  - **`screenshot`**: Stores a screenshot as a byte array.
  - **`gameTime`**: Tracks the game's time.
  - **`sceneName`**: Stores the name of the scene.
  - **`autosaveEnabled`**: Controls whether autosaving is enabled.
  - **`saveGameByEvent`**, **`saveGameByTime`**: Conditions for triggering saves.
  - **`saveInterval`**: Time interval for autosaving.
  - **`gameData`**, **`localLow`**, **`playerPrefs`**: Different saving modes.
  - **`saveCustomItems`**: List of **`SaveCustomItem`** representing various data types.

`SaveCustomItem`, **`SaveCustomFloat`**, **`SaveCustomInt`**, **`SaveCustomString`**, **`SaveCustomBool`** **Classes:**
- **Usage:** These serializable classes are used to structure and store specific data types within **`SaveCustomObject`**.
- **Variables:**
  - **`itemTag`**, **`floatTag`**, **`intTag`**, **`stringTag`**, **`boolTag`**: Identification tags for the respective data types.
  - **`floatValue`**, **`intValue`**, **`stringValue`**, **`boolValue`**: Actual values of the respective data types to be saved.

Overall:
- **Editor Specific Class: `CustomObjectDataCreator`** handles asset creation within the Unity Editor.
- **Data Management Class: `SaveCustomObject`** manages the custom object's settings and various data types for saving within the game.
- **Serializable Classes: `SaveCustomItem`**, **`SaveCustomFloat`**, **`SaveCustomInt`**, **`SaveCustomString`**, **`SaveCustomBool`** structure the data to be saved within the **`SaveCustomObject`** asset.

Usage:
- **Editor:** Creates a custom asset through the Unity Editor's menu option (**`Assets/Create/Save Custom Game/Save Custom Object Data`**).
- **Runtime: `SaveCustomObject`** is used during the game to hold and manage custom object data/settings for saving/loading within the game.

SaveCustomInitialization

This **`SaveCustomInitialization`** script serves as a setup mechanism during runtime initialization to handle the loading and initialization of a **`SaveCustomObject`**.

Variables:
- **saveCustomObject:** Static reference to the **`SaveCustomObject`** that will be initialized and used throughout the game.

Methods:
1. **RunGameInitialization:**
   - **Usage:** Executed upon runtime initialization due to the **`[RuntimeInitializeOnLoadMethod]`** attribute.
   - **Steps:**
     - Logs an initialization message.
     - Calls **`LoadSettingsData()`** to load the **`SaveCustomObject`** from Resources.
     - Checks if the **`saveCustomObject`** is loaded.
     - If not found, logs an error.
     - Creates a new GameObject named **"[Save Custom Object]"**.
     - Adds **`SaveCustomInScene`** and **`AutoSaveCustom`** components to the created GameObject.
     - Assigns references between components and objects.
     - Ensures the GameObject persists across scene changes using **`Object.DontDestroyOnLoad`**.

2. **LoadSettingsData:**
   - **Usage:** Loads the **`SaveCustomObject`** data from Resources.
   - **Steps:**
     - Loads the **`SaveCustomObject`** using **`Resources.Load`**.
     - Logs an error if the **`Save Custom Object Data`** fails to load.

Overall:
1. **Initialization:**
   - Logs an initialization message.
   - Loads the **`SaveCustomObject`** from Resources and checks if it's loaded.
2. **GameObject Setup:**
   - Creates a GameObject to hold save-related components.
   - Adds necessary components (**`SaveCustomInScene`** and **`AutoSaveCustom`**) to this GameObject.
   - Establishes references between these components and the **`SaveCustomObject`**.
3. **Persistence:**
   - Ensures the created GameObject persists across scene changes using **`DontDestroyOnLoad`**.

Usage:
- Automatically runs **`RunGameInitialization()`** upon the game's runtime initialization due to the **`[RuntimeInitializeOnLoadMethod]`** attribute.
- Ensures the **`SaveCustomObject`** is properly loaded from Resources.
- Creates a GameObject responsible for managing save-related functionalities, ensuring it persists throughout the game.
- Establishes necessary component references and setup for saving functionality.