Save Custom Game

Documentation

_____

Creator: Lucas Gomes Cecchini

Pseudonym: 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

The `**SaveDataUtility**` script is designed to facilitate the management of save data in a Unity project. It provides methods for accessing and modifying various types of data, capturing screenshots, converting textures to sprites, and managing auto-save functionality. Here's a breakdown of how each part of the script works:

Methods and Their Functions

1. `**GetSaveCustomObject**`:
   - **Purpose**: Loads and returns an instance of `**SaveCustomObject**` from the Unity Resources.
   - **Process**: Looks for an asset named "Save Custom Object Data". If not found, it logs an error and returns `**null**`.

2. `**GetFloat**`, `**GetInt**`, `**GetString**`, `**GetBool**`:
   - **Purpose**: Retrieve values of different types (float, int, string, bool) from `**SaveCustomObject**` using specific tags.
   - **Process**:
     - Retrieves `**SaveCustomObject**`.
     - Searches through a list of items (`**saveCustomItems**`) within the `**SaveCustomObject**`.
     - Finds the item using `itemTag`, then searches for the value using `**floatTag**`, `**intTag**`, `**stringTag**`, or `**boolTag**`.
     - Throws an error if the item or tag is not found.

3. `**SetFloat**`, `**SetInt**`, `**SetString**`, `**SetBool**`:
   - **Purpose**: Set values of different types (float, int, string, bool) in `**SaveCustomObject**`.
   - **Process**:
     - Retrieves `**SaveCustomObject**`.
     - Searches for the item using `**itemTag**`.
     - Updates the value if the tag is found. If the tag is not found, it creates a new entry.
     - If the item is not found, it creates a new item with the specified value and adds it to the list.

4. `**CaptureScreenshot**`:
   - **Purpose**: Captures a screenshot from a specified camera and stores it in `**SaveCustomObject**`.
   - **Process**:
     - Validates the camera and `**SaveCustomObject**`.
     - Creates a render texture with dimensions that fit within a specified pixel limit.
     - Renders the camera view to the render texture and reads the pixels into a `**Texture2D**`.
     - Encodes the `**Texture2D**` to PNG format and assigns it to `**SaveCustomObject**`.

5. `**RenderScreenshot**`:
   - **Purpose**: Converts a byte array of a screenshot into a `**Texture2D**`.
   - **Process**:
     - Checks if the byte array is valid.
     - Creates a `**Texture2D**` and loads the image data from the byte array.

6. `**TextureToSprite**`:
   - **Purpose**: Converts a `**Texture2D**` into a `**Sprite**`.
   - **Process**:
     - Validates the texture.
     - Defines a rectangle and pivot for the sprite.
     - Creates and returns a `**Sprite**` using the texture and defined parameters.

7. `**GetComponentSaveCustomInScene**`:

- **Purpose**: Finds and returns the **`SaveCustomInScene`** component from a GameObject in the scene.
- **Process**:
  - Searches for a GameObject named "[Save Custom Object]".
  - Tries to get the **`SaveCustomInScene`** component from it and logs the result.

8. **`EnableAutoSave`** and **`DisableAutoSave`**:
- **Purpose**: Enable or disable auto-saving in **`SaveCustomObject`**.
- **Process**:
  - Retrieves **`SaveCustomObject`**.
  - Sets the **`autosaveEnabled`** flag to true or false.

9. **`SaveEvent`**:
- **Purpose**: Triggers an auto-save event.
- **Process**:
  - Finds the GameObject named "[Save Custom Object]".
  - Gets the **`AutoSaveCustom`** component and calls its **`SaveAutoGame`** method to perform an auto-save.

Variables and Their Roles

- **`itemTag`**, **`floatTag`**, **`intTag`**, **`stringTag`**, **`boolTag`**: Tags used to identify specific data within **`SaveCustomObject`**.
- **`newValue`**: The value to be set or updated in the **`SaveCustomObject`**.
- **`targetCamera`**: The camera used to capture a screenshot.
- **`screenshot`**: Byte array representing the screenshot image.
- **`texture`**: Texture used to create a **`Sprite`**.

Usage

- Use this utility to manage game save data, such as player settings or game state.
- The screenshot functions can be used to capture in-game images for saving or sharing.
- Convert textures to sprites for use in UI or game objects.
- Manage auto-save functionality to ensure progress is regularly saved.

This script provides a comprehensive set of tools for saving, loading, and manipulating data within a Unity project.

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.

DefaultSelectedButton

This script is designed to manage the selection of a specific UI button when the UI element it is attached to is activated. Here's a breakdown of how it works:

Variables

- **button**: This is a reference to the button that you want to be selected by default. It's set through the Unity Inspector.

Methods

- **OnEnable()**: This method is automatically called when the UI element to which this script is attached becomes active. The steps it performs are:
  1. **Clears any previously selected UI object**: This ensures that no other UI element is currently selected when the UI element with this script becomes active.
  2. **Selects the desired button**: This sets the specified button as the currently selected UI element, so it's highlighted and ready for interaction by default.

Usage

- Attach this script to a UI element (such as a panel or window) in your Unity project.
- Assign the button you want to be selected by default to the **`button`** variable in the Inspector.
- When the UI element with this script becomes active, it will automatically clear any existing selection and set the specified button as the selected element.

This functionality is particularly useful for ensuring that the correct button is focused when a new UI screen appears, enhancing the user experience by streamlining navigation.

CheckButtonSelected

This script is designed to ensure that a default button is selected in the UI when no other UI elements are currently selected or when the currently selected element is inactive. Here's an explanation of how it works:

Variables

- **defaultButton**: This variable holds a reference to the button that should be selected by default. This button is set through the Unity Inspector.

Methods

- **Update()**: This method runs every frame and performs the following checks:
  1. **Checks if no object is currently selected**: It looks at the EventSystem to see if there is a selected UI element. If there is no selected object (**`currentSelectedGameObject`** is **`null`**), it means that no element is currently focused.
  2. **Checks if the currently selected object is inactive**: It checks whether the currently selected UI element (if any) is active in the hierarchy. If the selected object is inactive, it cannot be interacted with.
  3. **Selects the default button if needed**: If either of the above conditions is true, it sets the **`defaultButton`** as the selected element, ensuring that the user is directed to the default button for interaction.

Usage

- Attach this script to a UI element in your Unity project (like a panel or screen).
- Assign the button you want to be selected by default to the **`defaultButton`** variable in the Inspector.
- The script will automatically check each frame if the current selection is valid. If it finds that no UI element is selected or the selected element is inactive, it will set the default button as the selected element.

This ensures that there is always a consistent and accessible UI element for the user to interact with, enhancing the user interface experience and navigation.