Saving System

v1

Diovan Marques
github.com/DiovanM

Visão geral

O projeto oferece um sistema de Save e Load de dados de fácil implementação e uso. Possui um sistema base de gerenciamento de arquivos e um sistema paralelo/complementar para o gerenciamento opcional de Slots de save, realizados pelos scripts SavingSystem.cs e SaveSlotSystem.cs, respectivamente. Possui uma cena de demonstração básica do sistema, exemplificando o Save e o Load, assim como funções auxiliares e callbacks que serão descritos posteriormente. Adiciona também um opção no menu do Unity para deleção dos arquivos (para propósito de desenvolvimento) e um script com utilitários.

Sistema

O sistema consiste de dois scripts, SavingSystem.cs e SaveSlotSystem.cs, responsáveis respectivamente pelo sistema base de salvamento e o gerenciamento de slots de save.

SavingSystem

É a base do sistema de salvamento. Nele são feitos todos os processos de criação de arquivos, carregamento, deleção e gerenciamento.

Informações gerais

- O diretório de salvamento será diferente dependendo da plataforma, sendo os arquivos salvos em uma subpasta “Saves” dentro de tal diretório.
  (No windows o diretório é
  “C:\Users\{user}\AppData\LocalLow\{company}\{application}\Saves\”)
- Os arquivos serão salvos no formato JSON, sem criptografia.

Save

O salvamento funciona recebendo um objeto de qualquer tipo contendo os dados e uma key, que será o nome do arquivo a ser salvo e deve ser utilizada para carregar o arquivo posteriormente. O objeto é então serializado e salvo no diretório em formato JSON.

```
SavingSystem.Save(data, "SaveDataExample")
```

Obs: Todas as classes a serem serializadas não devem herdar de MonoBehaviour e devem possuir a tag [System.Serializable], como na classe de dados utilizada na cena de exemplo.

- Save:
  Método básico para salvamento. Recebe como parâmetros um objeto, uma key, e um sub-diretório opcional. Salva os dados dentro do Slot selecionado. A função retorna uma bool indicando o sucesso ao salvar os dados.

- SaveToRoot:
  Método alternativo para salvamento. Semelhante ao método anterior, porém salva o arquivo a partir do diretório base, invés de utilizar o Slot selecionado.

Load

O carregamento funciona recebendo o tipo de dado a ser carregado e a key a ser procurada, retornando um objeto com os dados, caso exista.

```
var data = SavingSystem.Load<CubeExampleData>("SaveDataExample");
```

- Load:
  Método básico para carregamento dos dados. Recebe como parâmetros o tipo de dado a ser carregado, a key salva no diretório e um sub-diretório opcional. Irá procurar os dados dentro do Slot selecionado. Retorna o objeto com os dados, caso exista.
- LoadFromRoot:
  Método alternativo para carregamento. Semelhante ao método anterior, porém procura o arquivo a partir do diretório base, invés de utilizar o Slot selecionado.

Callbacks

```
public static Action<bool> OnSave;
public static Action<bool> OnLoad;
```

Existem dois callbacks no sistema de Save:

1. OnSave:
   Chamado ao final de uma tentativa de salvar dados. Retorna uma bool indicando o sucesso da operação.
2. OnLoad:
   Chamado ao final de uma tentativa de carregar dados. Retorna uma bool indicando o sucesso da operação.

```
SavingSystem.OnSave += OnDataSaved;
void OnDataSaved(bool success)
{
    if (success)
        Debug.Log("[SaveTest] SaveCallback: Data has been saved");
    else
        Debug.Log("[SaveTest] SaveCallback: Data has failed to be saved");
}
```

O objetivo dos callbacks é que o usuário possa efetuar ações de forma mais simples sempre que algum dado é salvo, caso ache necessário.

Utils

A classe SavingSystem possui alguns métodos utilitários de uso geral e de uso específico.

- SaveExists:
  Verifica a existência de um arquivo de save. Recebe como parâmetros a key e um sub-diretório opcional.
- SaveExistsOnRoot:
  Semelhante ao método anterior, porém procura pelo arquivo a partir do diretório base, invés do Slot atual.
- DeleteSaveFile:
  Deleta um arquivo dentro do Slot selecionado atual. Recebe como parâmetros a key e um sub-diretório opcional.
- DeleteSaveFileOnRoot:
  Semelhante ao método anterior, porém procura pelo arquivo a ser deletado a partir do diretório base, invés do Slot atual.
- DeleteDirectory:
  Deleta um diretório dentro do Slot de save atual. Recebe como parâmetros o sub-diretório a ser deletado.
- DeleteAllSaveFiles:
  CUIDADO
  Deleta todos os arquivos dentro do diretório base. Recebe como parâmetro um sub-diretório opcional.

SaveSlotSystem

De uso opcional, gerencia o sistema de slots de save.

Informações gerais

- Um slot é definido por uma pasta dentro do diretório principal. O nome da pasta se dá pelo prefixo “Save” seguido do id do Slot, por exemplo “Save1”.
- Cada pasta de slot possui um arquivo de configuração chamado “SlotData” e sua existência define a pasta como uma pasta de Slot válida.
- Todos os dados salvos ficarão dentro da pasta do Slot atual selecionado.
- O uso do sistema de Slots é opcional. Caso um Slot não seja definido, os dados serão salvos na pasta “Save1”.

SlotData

```
public class SaveSlotBaseData
{
    public int id;
    public string name;
    public float completionPercentage;
    public float playTime; //Play time in seconds

    public SaveSlotBaseData()
    {
        id = 1;
        name = "Default";
        completionPercentage = 0f;
        playTime = 0f;
    }

    public SaveSlotBaseData(int id, string name, float completionPercentage, float playTime)
    {
        this.id = id;
        this.name = name;
        this.completionPercentage = completionPercentage;
        this.playTime = playTime;
    }
}
```

O SlotData possui a estrutura mostrada acima. As propriedades da classe SlotData podem ser alteradas, mas deve sempre possuir no mínimo a propriedade *id*.

O objetivo do SlotData, além de servir como guia para o sistema de Save, é guardar dados gerais do Save atual, como um nome, porcentagem de compleição do jogo e tempo de jogo.

Utilização do sistema

Para que o sistema de Slots seja utilizado corretamente, um novo SlotData deve ser criado antes de qualquer salvamento ou carregamento durante o jogo. Após criado, deve ser definido qual o id do Slot a ser utilizado. Feitas essas configurações, todos os dados salvos e carregados iram se basear no Slot utilizado.

- SaveSlotData:
  Cria um novo Slot. Recebe como parâmetro um objeto do tipo SaveSlotBaseData.
  CUIDADO
  Um Slot novo salvo irá sobrepor um Slot antigo, caso exista.
- SetCurrentSlot:
  Define o Slot a ser utilizado. Recebe como parâmetro um id. Retorna uma bool indicando o sucesso da operação.
- GetSaveSlotsData:
  Retorna uma lista do tipo SaveSlotBaseData com os Slots encontrados na pasta de Save. Importante ser utilizado para um melhor gerenciamento dos Slots.
- DeleteAllSlotData:
  Deleta o diretório do respectivo Slot. Recebe como parâmetro um id referente ao Slot a ser deletado. Retorna uma bool indicando o sucesso da operação.

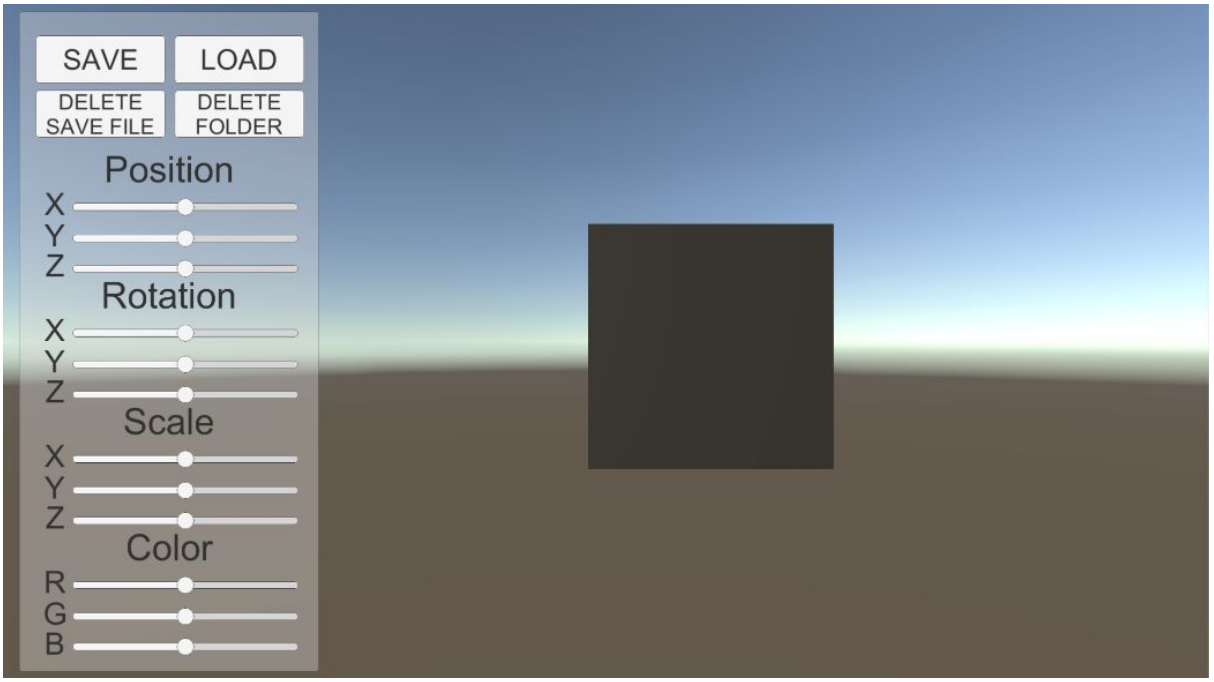
- UpdateCurrentSlotData:
  Atualiza no diretório os dados do Slot atual. Recebe como parâmetros o nome do Slot, tempo de jogo e porcentagem de compleição.
- currentSaveSlotData:
  Variável que contém os dados do Slot atual.

Utils

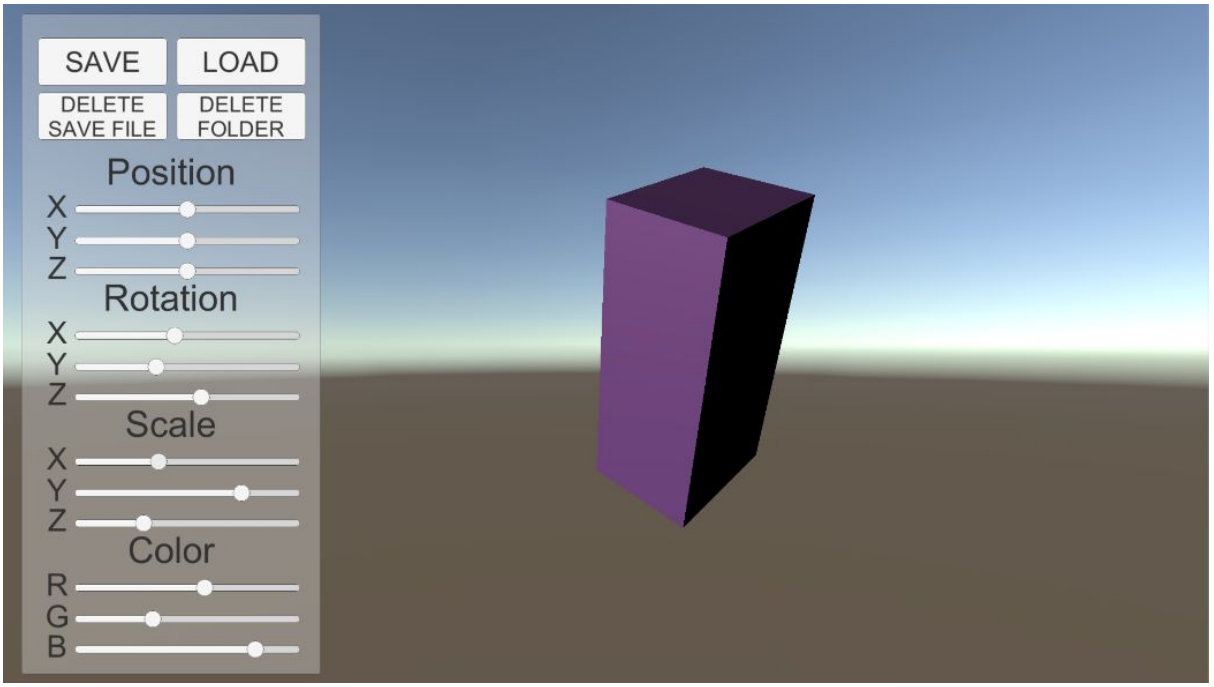
Existem dois métodos utilitários no gerenciamento de Slots, com o objetivo de uso ocasional.

- SetDirectoryAddress:
  O nome base padrão para a pasta de um Slot é “Save”, portanto os Slots salvos se chamarão “Save1”, “Save2”, etc. Esse método modifica o nome base para a pasta de Slots, podendo se chamar, por exemplo, “MeuSave1”, “MeuSave2”, etc. O método recebe como parâmetro o novo nome para a pasta de Slot.
- ResetDirectoryAddress:
  Reseta o nome base para a pasta de Slots para o nome padrão “Save”.

Exemplo



A cena de exemplo consiste de um cubo cujas características podem ser alteradas interagindo com os elementos na tela. Nela, é possível salvar as características atuais do cubo para carregá-las posteriormente. Também exemplifica a deleção do arquivo referente aos dados do cubo, assim como a deleção da pasta de save por inteira. Todas as ações realizadas são impressas no console para facilitar sua visualização.



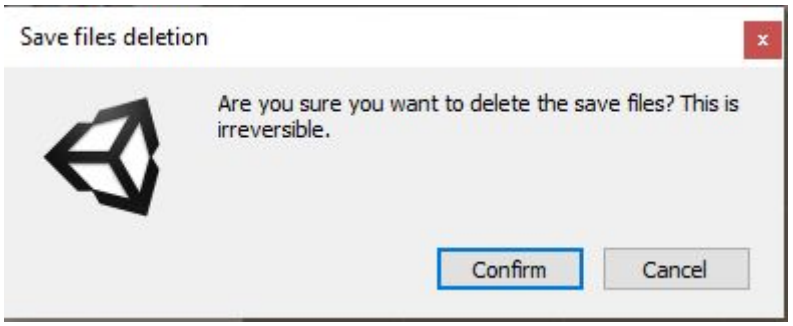
É recomendado um estudo básico do script de exemplo a fim de verificar o uso das features de Save, Load, deleção de arquivos, deleção de pasta, uso dos callbacks de Save, Load e a estruturação e uso dos dados tanto ao salvar quanto ao carregar.

Utils

O sistema contém dois scripts utilitários chamados SaveSystemEditorUtils.cs e UtilsClasses.cs.

SaveSystemEditorUtils

Adiciona uma opção de desenvolvimento no menu superior do Unity para a deleção de todos os arquivos dentro do diretório de saves.



UtilsClasses

```
[Serializable]
public class ListData<T>
{
    public List<T> list;

    public ListData() { }

    public ListData(List<T> list)
    {
        this.list = list;
    }
}
```

Concentra classes utilitárias de uso genérico que podem vir a ser úteis durante o uso do sistema. Atualmente possui a classe ListData, criada para facilitar o salvamento do tipo List<T> sem a necessidade de criar uma classe específica, uma vez que List não é serializável para JSON no sistema.

```
var data = new ListData<SoldierData>(soldiersData);
SavingSystem.Save(data, "soldiers_save");
```