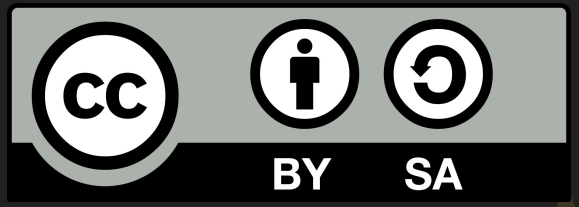
# Serialização e Save

Slides por:
Ítalo Tobler Silva (italo.tobler.silva@usp.br)
Rafael Miranda Lopes




Este material é uma criação do
Time de Ensino de Desenvolvimento de Jogos Eletrônicos (TEDJE)
Filiado ao grupo de cultura e extensão Fellowship of the Game (FoG), vinculado ao ICMC - USP

# Objetivos

- ➔ Explicar o que é serialização de dados
- ➔ Justificar a necessidade de serialização
- ➔ Mostrar como funciona serialização na Unity
- ➔ Alguns exemplos de como salvar informações persistentes na Unity
- ➔ Detalhar como a Unity serializa as informações (Editor e Runtime)

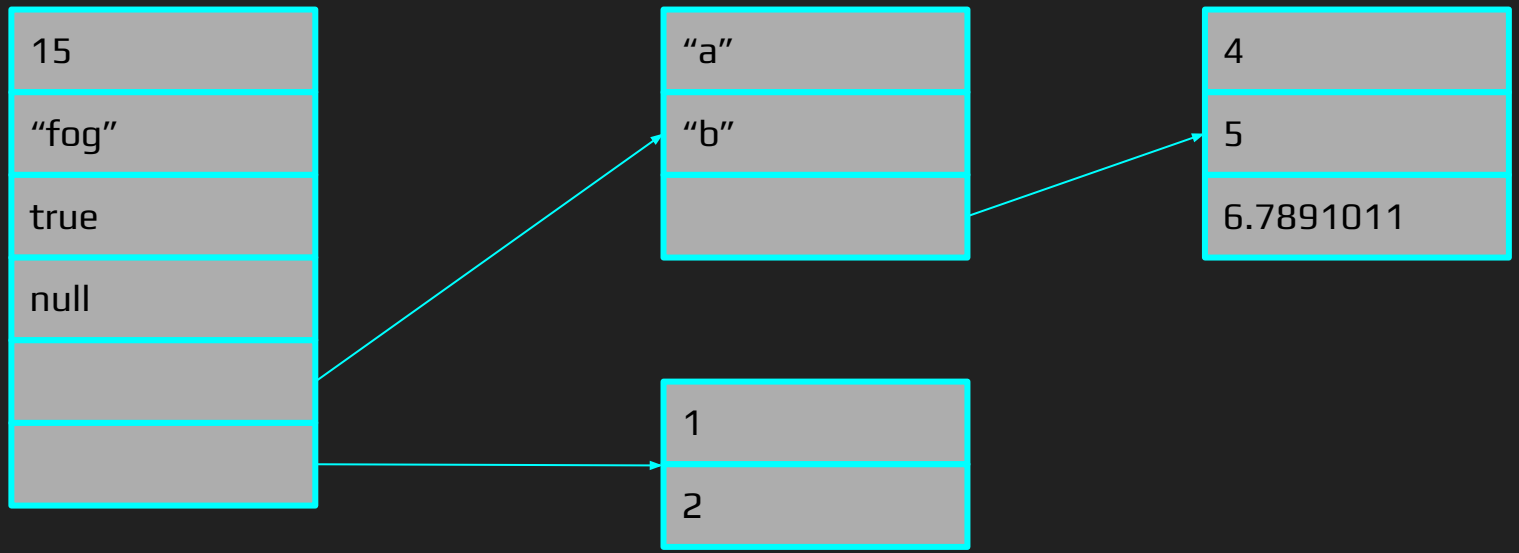
# Serialização

# Serialização

- ➔ Definição: Transformar dados dispersos em dados serializados, com estrutura definida e independente de arquitetura
- ➔ Desserializar: processo inverso da serialização

# Dados durante execução

HEAP



# Limitações

- ➔ Reutilizar os dados gerados?
- ➔ Transmitir informações?
- ➔ Interação com outros programas?

# Utilidade de serialização

- ➔ Persistência dos dados
- ➔ Independente de linguagem e arquitetura
- ➔ Envio pela rede

# Formatos de Serialização

- Bytestream
  - Conjunto de bytes, não necessariamente correspondendo a caracteres
  - Fazer a escrita e leitura das informações na mão

# Formatos de Serialização

- XML
  - Linguagem de markdown
  - Pode não ser muito legível
  - Antiga, fácil de encontrar ferramentas compatíveis

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="ProjectRootManager" version="2" languageLevel="JDK_1_7" project-jdk-name="1.8" project-jdk-type="JavaSDK">
    <output url="file://$PROJECT_DIR$/build/classes" />
  </component>
  <component name="ProjectType">
    <option name="id" value="Android" />
  </component>
</project>
```

# Formatos de Serialização

- YAML
  - Muitas funcionalidades
  - Sintaxe mais complexa
  - Usada pela Unity ->

```
%YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!29 &1
OcclusionCullingSettings:
  m_ObjectHideFlags: 0
  serializedVersion: 2
  m_OcclusionBakeSettings:
    smallestOccluder: 5
    smallestHole: 0.25
    backfaceThreshold: 100
  m_SceneGUID: 00000000000000000000000000000000
  m_OcclusionCullingData: {fileID: 0}
--- !u!104 &2
RenderSettings:
  m_ObjectHideFlags: 0
  serializedVersion: 9
  m_Fog: 0
  m_FogColor: {r: 0.5, g: 0.5, b: 0.5, a: 1}
  m_FogMode: 3
  m_FogDensity: 0.01
```

# Formatos de Serialização

- JSON
  - Mais moderno
  - Muito utilizada, especialmente em web
  - Fácil de usar

```json
{
    "name": "Dr Charles",
    "lives": 3,
    "health": 0.8,
    "level": 1,
    "timeElapsed": 47.5,
    "playerName": "Dr Charles Francis"
}
```

# Serialização na Unity

# XML, YAML

- XML
  - C# possui uma biblioteca oficial para manipulação de XML
- YAML
  - A unity usa uma versão personalizada de YAML para os arquivos de cena e prefabs
  - Existem bibliotecas para manipulação de arquivos YAML em C#

# Bytestream

- ➔ Array de bytes (byte[]) em C#
- ➔ Serialização feita manualmente, por exemplo com o uso de interfaces

```
public interface ISerializable{
    //Serialization
    byte[] Serialized();
    void Deserialize(byte[] data);
}
```

Dá trabalho:

```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
public interface ISerializable{
    //Serialization
    byte[] Serialized();
    void Deserialize(byte[] data);
}
public class Example : ISerializable{
    private int value;
    private string name;
    [System.Serializable]
    private struct SerialExample{
        public int _value;
        public string _name;
        public SerialExample(int value, string name){
            _value = value;
            _name = name;
        }
    }
    public byte[] Serialized(){
        SerialExample data = new SerialExample(value, name);
        MemoryStream stream = new MemoryStream();
        BinaryFormatter formatter = new BinaryFormatter();
        formatter.Serialize(stream, data);
        return stream.GetBuffer();
    }
    public void Deserialize(byte[] data){
        MemoryStream stream = new MemoryStream(data);
        BinaryFormatter formatter = new BinaryFormatter();
        SerialExample desserialized = (SerialExample)formatter.Deserialize(stream);
        value = desserialized._value;
        name = desserialized._name;
    }
}
```

# JSON

- A Unity já possui uma classe que facilita muito o uso chamada JsonUtility

```
[System.Serializable]
public class PlayerInfo{
    public string name;
    public int lives;
    public float health;
}

PlayerInfo player =  JsonUtility.FromJson<PlayerInfo>(jsonString);
string jsonString = JsonUtility.ToJson(player);
JsonUtility.FromJsonOverwrite(jsonString, player);
```

- Descrição no próprio manual da Unity

# Como salvar as informações?

- Escrita em arquivo
  - System.IO em C#
- Onde salvar o arquivo?
  - Application.persistentDataPath
  - Definido pela unity, diferente para cada plataforma

```
using System.IO;

public void Save(PlayerInfo player){
    string jsonString = JsonUtility.ToJson(player);
    string path = Application.persistentDataPath;
    Directory.CreateDirectory(path);
    string fileName = "playerSave";
    string filePath = System.IO.Combine();
    using (StreamWriter streamWriter = File.CreateText (localFilePath)){
        streamWriter.Write (jsonString);
        streamWriter.Close();
    }
}
```

```
public PlayerInfo Load(){
    string path = Application.persistentDataPath;
    string fileName = "playerSave";
    string filePath = System.IO.Combine();
    PlayerInfo player = null;
    if(File.Exists(filePath)){
        try{
            using (StreamReader streamReader = File.OpenText(localFilePath)){
                string jsonString = streamReader.ReadToEnd();
                if(jsonString.Length>0){
                    player = JsonUtility.FromJson<T>(jsonString);
                }
                streamReader.Close();
            }
        }catch{
            // Arquivo corrompido
        }
    }
    return player;
}
```

# persitentDataPath e itch.io

- ➔ Jogos no itch tem o persistentDataPath alterados quando uma nova versão é salva
- ➔ Workaround:

```
string path;
#if UNITY_WEBGL
path = System.IO.Path.Combine("/idbfs", Application.productName);
#else
path = Application.persistentDataPath;
#endif
Directory.CreateDirectory(path);
```

# PlayerPrefs

- Sistema da Unity para salvar informações sem serializar as coisas ou escrever em arquivos manualmente
- Funciona como um conjunto de Dictionaries para alguns tipos de variável pré definidos
- Recomendado para preferências, como volume do jogo
- Bem simples e bem documentado

# Networking

- ➔ Envio das informações serializadas ao invés de escrever em um arquivo
- ➔ Antigo sistema de networking da unity (unet) foi descontinuado e ainda não há uma alternativa oficial
- ➔ Por enquanto recomenda-se o uso de pacotes como o Mirror

# Por baixo dos panos

# Serialização automática de objetos

- ➔ Tipos de dado já existentes (não referência) são serializáveis por padrão
- ➔ Classes e structs explicitamente marcadas como serializáveis
- ➔ Campos públicos e privados com [SerializeField]

```
[System.Serializable]
public class PlayerInfo{
    public string name;
    public int lives;
    public float health;
}

PlayerInfo player =  JsonUtility.FromJson<PlayerInfo>(jsonString);
string jsonString = JsonUtility.ToJson(player);
JsonUtility.FromJsonOverwrite(jsonString, player);
```

# E para tipos não serializáveis por padrão?

- Interface ISerializationCallbackReceiver
  - Callbacks antes de serializar e após desserializar
  - Crie campos que a Unity consegue serializar e use-os para armazenar as informações dos campos não serializáveis

# Unity Editor

- O editor da Unity funciona quase como uma aplicação feita usando Unity
- Cenas, prefabs e outros assets tem suas informações guardadas em um arquivo serializado e um .meta
- Inspetor e scripts de Editor
- Principal diferença entre o Editor e a aplicação gerada: Autoridade para alterar as configurações de assets

# Unity Editor (Modo de edição)

- Abrir uma cena/prefab na unity desserializa o arquivo e nos dá uma interface para editar o conteúdo (salvar a cena serializa as alterações)
- O inspetor de um asset desserializa as informações do arquivo e do .meta correspondente e nos dá uma interface para alterar as opções, serializando alterações
  - Ferramentas de editor nos permitem alterar o comportamento do inspetor e outras funções

# Unity Editor (Runtime)

- Após o jogo ser buildado, os assets não são mais alterados, a não ser que alguma alteração seja feita manualmente através do código
  - Application.dataPath mostra onde os assets ficam
- Informações de assets são desserializadas e carregadas em memória (funciona como qualquer programa)
- Não há edição, mas ainda pode haver serialização

# Unity Editor (Modo de jogo)

- Um híbrido entre o modo de edição e runtime
- O jogo é carregado usando a cena (ou cenas) atualmente aberta como ponto inicial
  - Usa o estado em memória, não o asset
  - Permite alteração de objetos carregados em memória na cena
  - Permite alteração de assets durante a execução, exceto salvar cenas

# Dúvidas?

# Referências


[1] https://docs.unity3d.com/Manual/script-Serialization.html
[2] https://devopedia.org/data-serialization
[3] https://www.w3.org/TR/xml11//
[4] https://docs.microsoft.com/en-us/dotnet/api/system.xml.xmldocument
[5] https://yaml.org/
[6] https://github.com/aaubry/YamlDotNet
[7] https://docs.unity3d.com/Manual/UnityYAML.html
[8] https://www.json.org
[9] https://docs.unity3d.com/Manual/JSONSerialization.html
[10] https://docs.microsoft.com/en-us/dotnet/api/system.io?view=netcore-3.1
[11] https://docs.unity3d.com/ScriptReference/Application-persistentDataPath.html
[12] https://docs.unity3d.com/ScriptReference/Application-dataPath.html
[13] https://itch.io/t/140214/persistent-data-in-updatable-webgl-games
[14] https://docs.unity3d.com/ScriptReference/PlayerPrefs.html
[15] https://mirror-networking.com/
[16] https://docs.unity3d.com/Manual/script-Serialization-Custom.html
[17] https://docs.unity3d.com/ScriptReference/ISerializationCallbackReceiver.html
[18] https://www.youtube.com/watch?v=zObWVOv1GlE


Todos os links acessados em 15/11/2020