

https://github.com/JGrandtnerova/BlockBreaker_XML

Ukladanie dát v Unity do súboru XML

Úvod

V tomto dokumente je popísané ukladanie dát do Unity do XML súboru, a všeobecne čo všetko nám treba v Unity vytvoriť.

1. Ukladanie dát pomocou XML

```
using UnityEngine;
using System.Xml.Serialization; //obsahuje triedy, ktoré sa používajú na serializáciu objektov do dokumentu. V .xml formáte

14 references
public class ActorData
{
    [XmlAttribute("Name")]
    public string name;

    [XmlElement("Level")]
    public int level;

    [XmlElement("Score")]
    public int score;
}
```

Vytvorila som si script ActorData, v ktorom som vytvorila premenné name, level a score – nebudem ho v Unity ničomu priradovať, takže môžem zmazať MonoBehaviour.

← Bolo potrebné pridať knižnicu:

`using System.Xml.Serialization;` //obsahuje triedy, ktoré sa používajú na serializáciu objektov do dokumentu. V .xml formáte

Je to kvôli tomu, že tieto premenné sa ukladajú do XML dokumentu.

← XmlAttribute je atribút Actora (alebo inak povedané Playera), čo predstavuje naše meno. Dala som ho ako **atribút** kvôli neskoršiemu vyhľadávaniu podľa mena (aby

sa nemuselo vyhľadávať do hlbších úrovní), ktoré je spravené pre to, aby sa Actori neopakovali (každé 1 meno bude mať len 1 záznam v .xml súbore a ten sa bude prepisovať, aby sa to dalo dobre loadovať podľa mena).

Ostatné premenné (level, score) som dala ako elementy (XmlElement). Ukážeme si to keď budeme pozerať výsledný .xml súbor.

Ďalej je potrebné vyriešiť ako sa nám budú tieto dáta do .xml súboru ukladať. Na to slúži ActorContainer.cs:

Používa knižnice System.Xml.Serialization(vysvetlené vyššie), a System.Collections.Generic (pre vytvorenie Listu objektov)

Je potrebné pridať aj názov schémy rootu .xml dokumentu:

`[XmlAttribute("ActorCollection")]`, ktorú sme si nazvali ActorCollection(zobrazuje sa v .xml dokumente). V Classe ActorContainer vytvoríme XML pole všetkých Aktorov (Actors), v ktorom Itemy, alebo položky, budú jednotliví aktori (Actor). Následne vytvoríme nový list, do

```
1 using System.Collections;
2 using UnityEngine;
3 using System.Xml.Serialization;
4 using System.Collections.Generic;
5
6 [XmlAttribute("ActorCollection")] //nazov schemy rootu/korena XML dokumentu
7 public class ActorContainer
8 {
9     [XmlElement("Actors")]
10    [XmlElement("Actor")]
11    public List<ActorData> actors = new List<ActorData>();
12
13
14 }
```

ktorého vkladáme už dáta Actora z ActorData -> pod Item Actor.

Ďalej potrebujeme, aby sa dal načítať a uložiť náš Actor, teda aby sme mali nejaké dáta ktoré môžeme uložiť. Funkcionalita je vyrúšená v SaveData.cs, obsahujúca nasledovné metódy →

Je potrebné taktiež použiť knižnice System.IO (slúžiaci na prácu s prúdmi) a System.Xml.Serialization (už vysvetlené vyššie).

```
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using System.IO;
5 using System.Xml.Serialization;
6 using UnityEngine;
7
8 public class SaveData
9 {
10     public static ActorData Load(string path, string name) ...
11
12     public static void SaveActor(ActorData data) ...
13
14     public static ActorContainer LoadActors(string path) ...
15
16     public static void SaveActors(string path, ActorContainer actors) ...
17 }
```

```
public static void SaveActor(ActorData data)
{
    bool saved = false;

    ActorContainer newactors = new ActorContainer();
    ActorContainer actors = SaveData.LoadActors(Path.Combine(Application.dataPath, "Resources/actors.xml"));
    foreach (ActorData datas in actors.actors)
    {
        if (datas.name == data.name)
        {
            newactors.actors.Add(data);
            saved = true;
        }
        else { newactors.actors.Add(datas); }
    }
    Console.WriteLine(data);
    if (!saved)
    {
        newactors.actors.Add(data);
    }
    SaveData.SaveActors(Path.Combine(Application.dataPath, "Resources/actors.xml"), newactors);
}
```

V metóde SaveActor najskôr zisťujeme, či daný actor v našom .xml súbore existuje. Ak existuje, pridáme mu nové dáta, ak neexistuje, tak sa vytvorí nanovo. Všetky ostatné dáta sa prepíšu. Potom sa dáta uložia pomocou SaveActors metódy na určenú pathu.

Metóda SaveActors sa stará o to, že vytvorí nový XML serializér typu ActorContainer, otvorí filestream otvorí filestream, a otvorí

súbor, **ktorý už existuje** - FileMode.Truncate, pričom truncate mu poskytne možnosť zapisovať do súboru - a ktorý je potom potrebné zatvoriť!! serializuje stream, zatvorí stream a návratovou hodnotou práve actors. Tým sa zabezpečí načítanie z našej pathy.

```
public static void SaveActors(string path, ActorContainer actors)
{
    XmlSerializer serializer = new XmlSerializer(typeof(ActorContainer));
    FileStream stream = new FileStream(path, FileMode.Truncate);
    serializer.Serialize(stream, actors);
    stream.Close();
}
```

```

public static ActorData Load(string path, string name)
{
    ActorContainer actorContainer = new ActorContainer();
    actorContainer = LoadActors(path);

    foreach(ActorData data in actorContainer.actors)
    {
        if (name == data.name)
        {
            return data;
        }
    }
    return null;
}

```

V metóde Load, ktorá vracia ActorData si vytvoríme nový ActorContainer, do ktorého načítame dáta, a ak sa meno zhoduje s menom v .xml súbore, dáta sa načítajú. Ak nie tak sa nenačíta nič.

V Loade sme používali metódu LoadActors pre pridelenie aktorov do kontajnera. Táto metóda sa stará o to, že vytvorí nový XML serializér typu ActorContainer, otvorí filestream a otvorí súbor, **ktorý už existuje** - FileMode.Open - a ktorý je potom potrebné zatvoriť!! deserializuje stream do kontajnera actors, zatvorí stream a návratovou hodnotou práve actors. Tým sa zabezpečí načítanie z našej pathy.

```

public static ActorContainer LoadActors(string path)
{
    XmlSerializer serializer = new XmlSerializer(typeof(ActorContainer));
    FileStream stream = new FileStream(path, FileMode.Open);
    ActorContainer actors = serializer.Deserialize(stream) as ActorContainer;

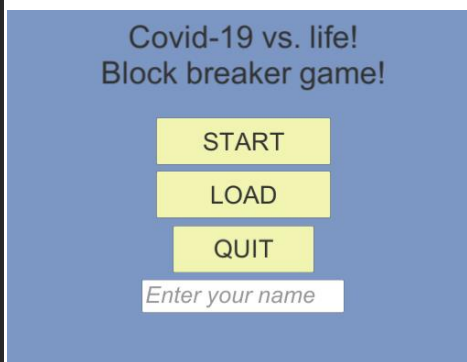
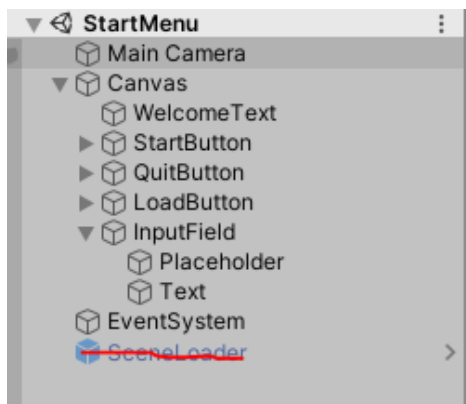
    stream.Close();

    return actors;
}

```

2. Pridanie funkcionality do Unity

Start Menu Scéna

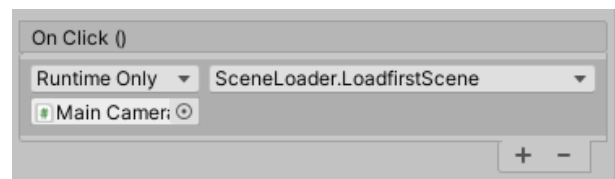


V StartMenu scéne som pridala tlačidlo Load a InputField, do ktorého používateľ zadá svoje meno.

InputField -> do Placeholdera vkladám „Enter your name“ text, ktorý sa následne

po kliknutí prepíše tým, čo užívateľ vložil do poľa (to sa potom vloží do Text).

GameObject/prefab SceneLoader nebudeme v StartMenu potrebovať. Vložila som skript SceneLoader do Main Camery, a v tlačidlách Start, Load a Quit ťahám príslušné funkcie cez MainCamera -->



Následne som upravila SceneLoader nasledovne:

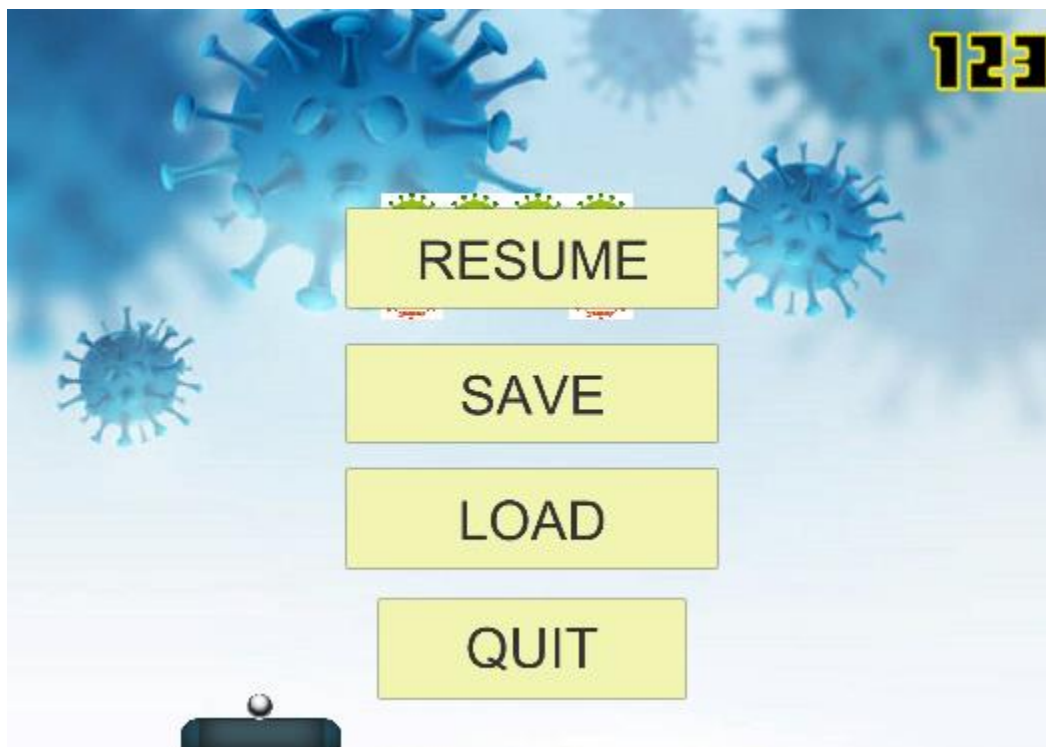
Funkcia QuitGame() zostala nezmenená.

LoadNextScene() už slúži len na prechod medzi levelmi, zo StartMenu cez tlačidlo START sa dostanem pomocou funkcie LoadfirstScene(), kvôli pridávaniu mena hráča z InputFieldu (čo je spravené ďalej vo funkcionalite PauseMenu), čo teda nechcem, aby sa volalo pri prechode z levelu do levelu:

```
public void LoadfirstScene()  
{  
    // zo start buttonu do lvl1  
  
    PauseMenu.playername = playername.text;  
  
    int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;  
    SceneManager.LoadScene(currentSceneIndex + 1);  
    GameStatus.currentScore = 0;  
}
```

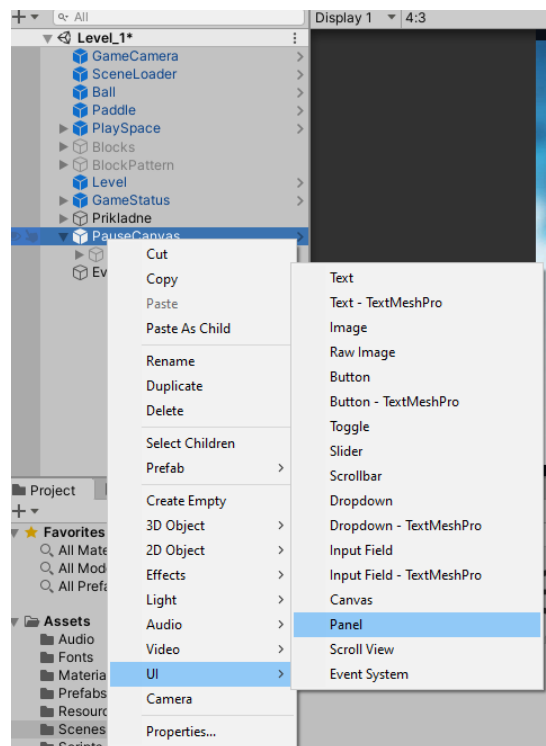
Pre tlačidlo LOAD je vytvorená funkcia LoadSavedScene(), ktorá taktiež pomocou PauseMenu dostane z InputFieldu player name, vyhľadá v actors.xml (cez ActorData) to meno a data, a cez SceneManager.LoadScene

Scény Levelov



Vytvoríme si PauseCanvas, ktorý si po skončení práce šupneme do prefabov a využijeme aj v druhom leveli. Následne si do PauseCanvasu vložíme Panel →

Panel si pomenujeme Pause menu, a vložíme do neho 3 buttony RESUME, SAVE, LOAD, QUIT (skopírujeme si ich zo StartMenu a len prepíšeme texty, názvy, a vymažeme OnClick).



Ďalším krokom je vytvorenie PauseMenu.cs, pre pauzovanie našej obrazovky v leveli, a spravenie funkcionality buttoniv RESUME, SAVE, LOAD, QUIT (plus využité aj v START buttone :

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5  using UnityEngine.SceneManagement;
6
7  public class PauseMenu : MonoBehaviour
8  {
9      public static bool gamePaused = false;
10
11      public GameObject pauseMenuUI;
12
13      void Update()...
14
15      public void Resume()...
16
17      void Pause()...
18
19  }
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

```

Do Pause Menu sa dostaneme v Leveloch stlačením escape (a von z neho buď cez Resume a lebo ešte raz escape). Preto som vytvorila metódu Update nasledovné: Zadeklarovala som **premennú gamePaused** a nastavila som ju na false (lebo teda nechceme aby bola pauznutá hneď od začiatku), a game object pauseMenuUI:

```

public static bool gamePaused =
false;
public GameObject pauseMenuUI;

```

Spravím v metóde Update() nasledovné →

V nej sú metódy Resume() a Pause(), ktoré som vytvorila nasledovne:

```

public void Resume()
{
    pauseMenuUI.SetActive(false);
    Time.timeScale = 1f;
    gamePaused = false;
}

void Pause()
{
    pauseMenuUI.SetActive(true);
    Time.timeScale = 0f;
    gamePaused = true;
}

```

```

void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape)) {
        if (gamePaused)
        {
            Resume();
        }
        else
        {
            Pause();
        }
    }
}

```

← TimeScale nastavím na 0f kvôli tomu, aby mi na pozadí hra nebežala ale stopla sa. Keď stlačím Resume button, time scale potrebujem nastaviť na 1f – aby bežala.

Ďalej je teda potrebné vytvoriť ukladanie a načítanie dát pre buttony v pause menu v leveli (SaveButton a LoadButton). To nám v skripte PauseMenu.cs zabezpečia metódy SaveDatas() a LoadData().

Vytvoríme si teda premennú `public static string` `playername`;

V `SaveDatas()` si teda vytvoríme nové `ActorData` a vložíme do nich `playername`, `level` a `score`. Potom zavoláme `SaveData.SaveActor` nech sa nám to uloží. To isté, ale opačne spravíme do `LoadData`, až na to, že mi to nefungovalo bez toho, aby som tam definovala pathu. Tak si ju tam teda definujeme. Taktiež si nastavíme `gamePaused` na `false`, aby nám to hneď spustilo a nemuseli sme stláčať `escape/resume` button znovu ako pri `Save` buttone, kde to podľa mňa má význam. Tu je trošku bug, lebo aj tak to nefunguje ako by som chcela.

```
0 references
public void SaveDatas()
{
    ActorData data = new ActorData();
    data.name = playername;
    data.level = SceneManager.GetActiveScene().buildIndex;
    data.score = GameStatus.currentScore;

    SaveData.SaveActor(data);
}

0 references
public void LoadData()
{
    ActorData data = SaveData.Load(System.IO.Path.Combine(Application.dataPath, "Resources/actors.xml"), playername);
    GameStatus.currentScore = data.score;
    SceneManager.LoadScene(data.level);
    Time.timeScale = 1f;
    gamePaused = false;
}
```