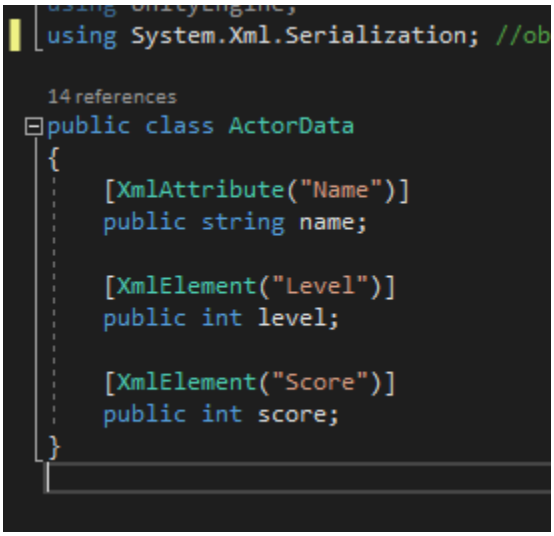
https://github.com/JGrandtnerova/BlockBreaker_XML

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



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

← Bolo potrebné pridať knižnicu:

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

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

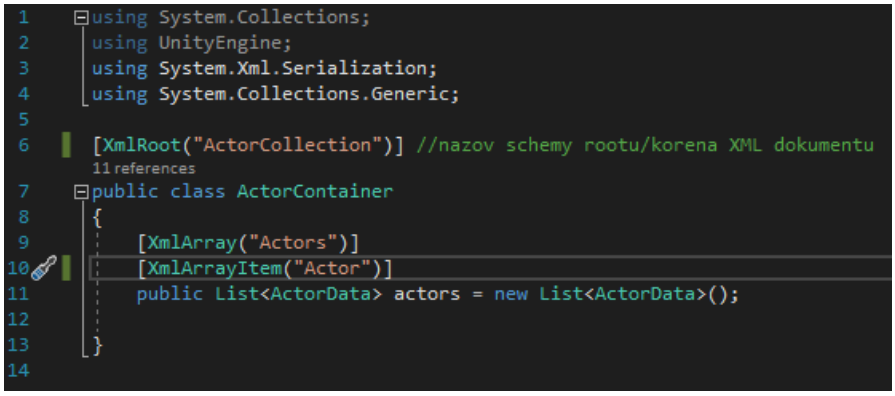
← XmlAttribute je atribút Actora (alebo inak povedané Playera), čo predstavuje naše meno. Dala som ho ako **atribút** kvôli neskoršiemu vyhľadávaniu podľa mena (aby sa nemuselo vyhľadávať do hlbších úrovní),ktoré je spravené pre to, aby sa Actori neopakovali (každé 1 meno bude mať len 1 záznam v .xml 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 schému rootu .xml dokumentu: `[XmlRoot("ActorCollection")]` , ktorú sme si nazvali ActorCollection(zobrazuje sa v .xml dokumente). V Classe ActorContainer vytvorime XML pole všetkých Aktorov (Actors), v ktorom Itemy, alebo položky, budú jednotliví aktori (Actor). Následne vytvoríme nový list, do



ktorého vkladáme už dáta Actora z ActorData.

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

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

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Xml.Serialization;
using UnityEngine;

5 references
public class SaveData
{
    2 references
    public static ActorData Load(string path,string name)...

    1 reference
    public static void SaveActor(ActorData  data)...
    2 references
    public static ActorContainer LoadActors(string path)...
    1 reference
    public static void SaveActors(string path,ActorContainer actors)...
}
```

```
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/cestu.

```
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();
}
```

Metóda SaveActors sa stará o to, že vytvorí nový XML serializér typu ActionContainer, 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.

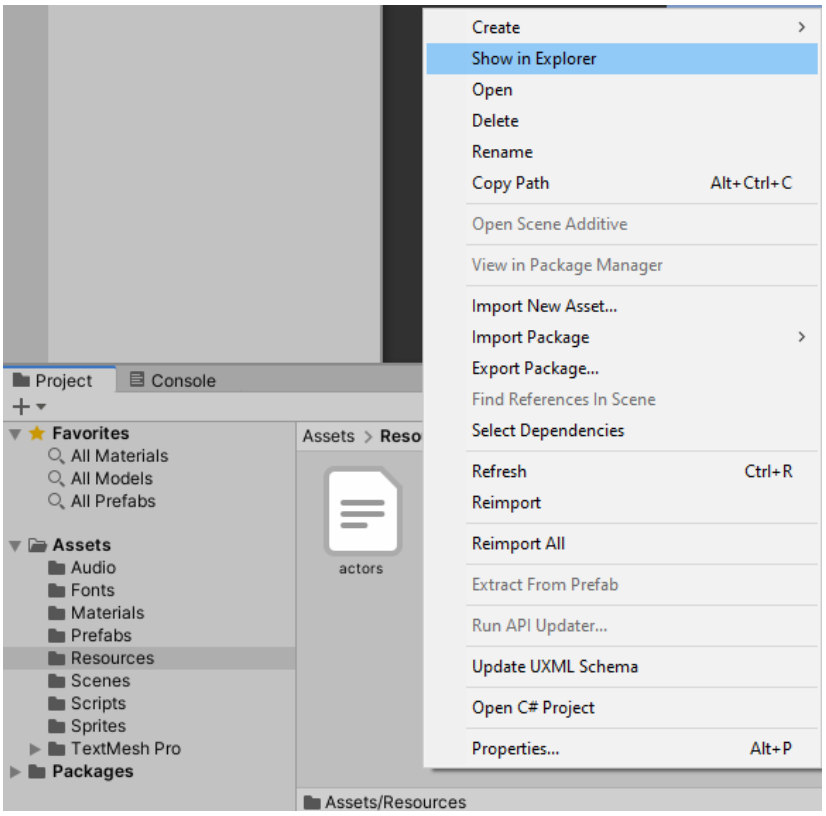
```
public static ActorData Load(string path,string name)
{
    ActorContainer actorContainer = new ActorContainer();
    actorContainer = LoadActors(path);

    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 nečítajú. Ak nie tak sa nenačíta nič.

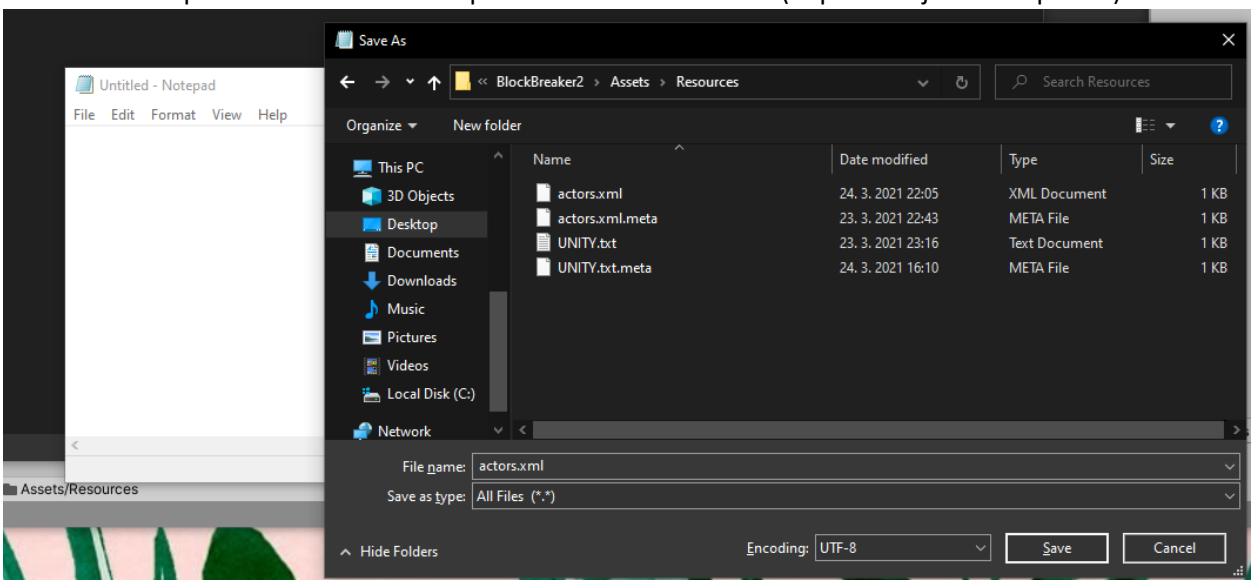
```
public static ActorContainer LoadActors(string path)
{
    XmlSerializer serializer = new XmlSerializer(typeof(ActorContainer));
    FileStream stream = new FileStream(path, FileMode.Open);
    ActorContainer actors = serializer.Deserialize(stream) as ActorContainer;

    stream.Close();

    return actors;
}
```

V Loade sme používali metódu LoadActors pre pridelenie aktorov do kontajnera. Táto metóda sa stará o to, že vytvorí nový XML serializér typu ActionContainer, 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.

Všimneme si, že je stále spomenuté, že .xml súbor už existuje. Je teda potrebné ho vytvoriť Vytvoríme priečinok s názvom Resources, a otvoríme ho vo File Exploreri



Následne si do priečinku uložíme .xml spbor s názvom actors.xml (napríklad aj cez notepad ☺)



Výsledný dokument po testovaní hry vyzerá nejako takto (otvorený vo visual studio):

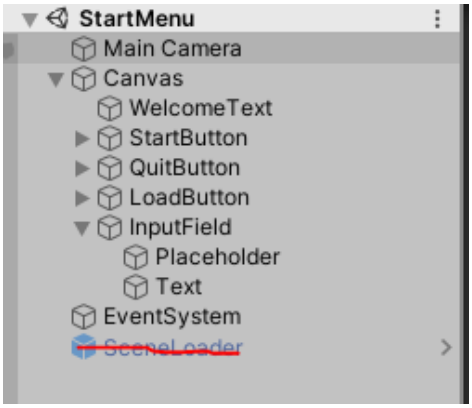
actors.xml

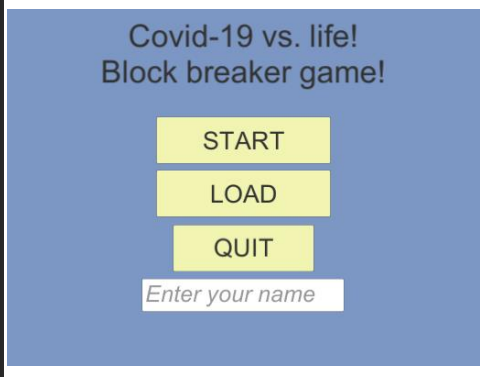
```
<?xml version="1.0"?>
<ActorCollection xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Actors>
    <Actors Name="sss">
      <Level>2</Level>
      <Score>4</Score>
    </Actors>
    <Actors Name="Player1">
      <Level>2</Level>
      <Score>52</Score>
    </Actors>
    <Actors Name="Plasads">
      <Level>2</Level>
      <Score>4</Score>
    </Actors>
    <Actors>
      <Level>2</Level>
      <Score>4</Score>
    </Actors>
    <Actors Name="ssss">
      <Level>1</Level>
      <Score>2</Score>
    </Actors>
    <Actors Name="aaa">
      <Level>2</Level>
      <Score>4</Score>
    </Actors>
  </Actors>
</ActorCollection>
```

Môžeme vidieť, že meno je samostatný atribút, tak to vidíme pekne navrchu, a ostatné elementy Level a Score máme pod týmto atribútom. Každé meno je jedinečné, nemôže existovať 2x hráč s rovnakým menom -> údaje sa prepíšu ☺.

2. Pridanie funkcionality do Unity

Start Menu Scéna

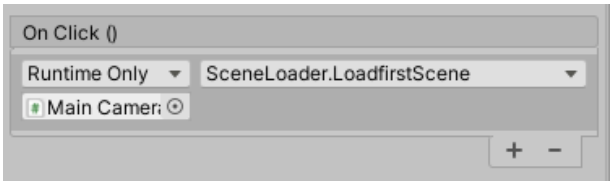




V StartMenu scéne som pridala tlačidlo Load a InputField, do ktorého používateľ zadá svoje meno.

InputField -> do Placeholdera vlkadám „Enter your name“ text, ktorý sa následne po kliknutí prepíše tým, čo užívateľ vloží do poľa (to sa potom vloží do Text).

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



Následne som upravila SceneLoader následovne:

Funkcia QuitGame() zostala nezmenená.

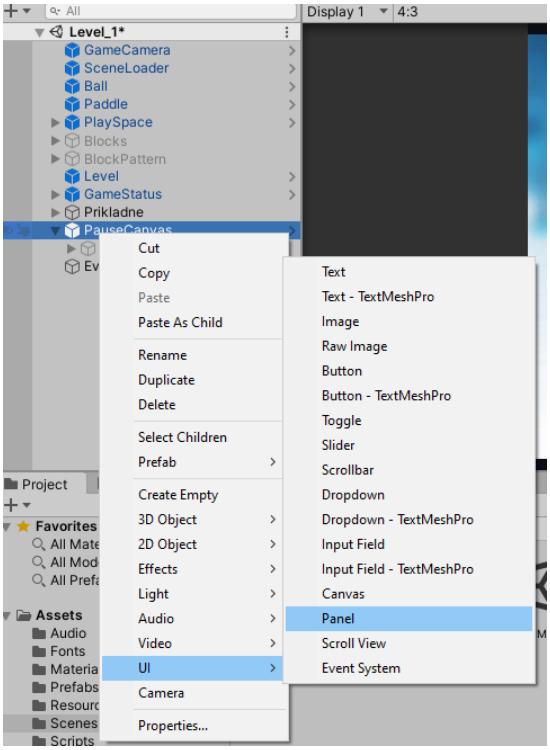
LoadNextScene() už slúži len na prechod medzi levelmi, zo StartMenu cez tlacidlo START sa dostanem pomocou funkcie LoadfirstScene(), kvôli pridávaniu mena hráča z InputFieldu (čo je spravené ďalej vo funkcionalite PauseMenu), čo teda nechcem, aby sa volalo pri prechode z levelu do levelu:

```
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

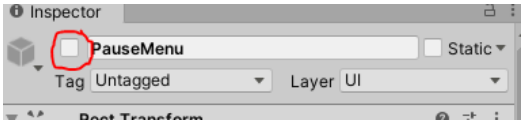
Scény Levelov

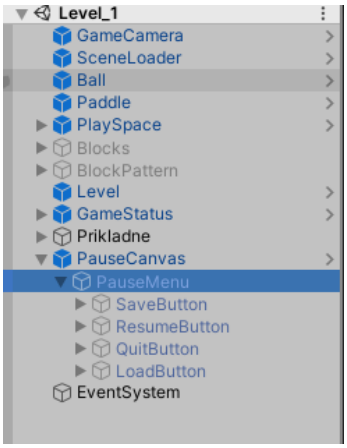
Vytvoríme si PauseCanvas ako Game Object, ktorý si po skončení práce šupneme do prefabov a využijeme aj v druhom leveli. Následne si do PauseCanvasu vložíme Panel →



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).

Následne náš panel PauseMenu vypneme, pretože ho voláme pomocou Escape na klávesnici, a nechceme, aby tam bol zobrazený stále.





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 :

```using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script | 2 references
public class PauseMenu : MonoBehaviour
{
    public static bool gamePaused = false;

    public GameObject pauseMenuUI;


    Unity Message | 0 references
    void Update()...
    1 reference
    public void Resume()...
    1 reference
    void Pause()...

}
```

Do Pause Menu sa dostaneme v Leveloch stlačením escape (a von z neho buď cez Resume a lebo ešte raz escape). Preto som vytvorila metóde Update následovné: 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 →

```
Unity Message | 0 references
void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape)) {
        if (gamePaused)
        {
            Resume();
        }
        else
        {
            Pause();
        }
    }
}
```

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

```
public void Resume()
{
    pauseMenuUI.SetActive(false);
    Time.timeScale = 1f;
    gamePaused = false;
}
1 reference
void Pause()
{
    pauseMenuUI.SetActive(true);
    Time.timeScale = 0f;
    gamePaused = true;
}
```

← TimeScale nastavim 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 potrebujeme aj definovať cestu, kde sa to uloží. 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;
}
```