# **Dokumentácia k ukladaniu a načítaniu dát v hre Arkanoid**

Táto implementácia ukladania a načítania dát bude rozdelená do dvoch častí:

* **Dáta hráča** – meno hráča, skóre z levelov, hviezdičky.
* **Dáta levelov** – informácie o dizajne levelov vytvorených hráčom v level editore (pozadie levelu, pozície a typy blokov)

# **Ukladanie a načítanie dát hráča**

## **Trieda SaveData**

Ako prvé sme vytvorili triedu, ktorá bude reprezentovať štruktúru ukladaných dát a bude použitá na manipuláciu s týmito dátami pred uložením a po načítaní. Trieda sa nazýva SaveData.

using System.Collections.Generic;

[System.Serializable]

public class LevelData

{

    public int levelID;

    public int score;

    public int stars;

}

[System.Serializable]

public class SaveData

{

    public string playerName;

    public List<LevelData> levelSaves = new List<LevelData>(new LevelData[10]);

}

Táto trieda obsahuje premennú **playerName** typu string pre uchovávanie mena hráča a list objektov typu **LevelData** o veľkosti 10, keďže v hre sa bude nachádzať 10 levelov. Trieda **LevelData** obsahuje 3 premenné typu integer, **levelID** pre id konkrétneho levelu, **score** pre maximálne score dosiahnuté v danom leveli a **stars** pre maximálny počet hviezdičiek dosiahnutých v danom leveli.

Všetky premenné musia byť verejné, aby k nim bolo možne pristupovať a triedy musia mať atribút **Serializable** aby ich bolo možné serializovať.

# **Funkcie Save() a Load()**

Funkcie **Save()** a **Load()** slúžia na ukladanie a načítanie z a do súboru **save.json**. Na serializovanie a deserializovanie dát do formátu JSON používame Unity triedu **JsonUtility** a pre zapísanie a prečítanie dát z a do súboru používame triedu **File** z knižnice System.IO

## **Funkcia Save()**

Funkcie **Save()** slúžia na serializvanie dát v triede **SaveData** do formátu JSON a uloženie týchto dát do súboru **save.json**.

Táto funkcia má ako argument objekt triedy **SaveData** obsahujúci dáta pre uloženie.

Dáta sa serializujú do premennej **json** typu string formátované pre jednoduchú čitateľnosť človekom (druhy argument vo funkcii ToJson).

Následne sa text v tejto premennej zapíše do súboru save.json na lokácii **Application.dataPath** ak tento súbor ešte neexistuje bude automaticky vytvorený.

**Application.dataPath** reprezentuje cestu k súboru v ktorom sa hra momentálne nachádza (tradičnejšie sa ako lokácia používa .persistentDataPath, ktorá na každom operačnom systéme predstavuje inú cestu vhodnú pre ukladanie dát, avšak keďže táto hra nebude exportovaná na iné operačné systémy a mne sa viacej páči keď to mám v súbore s hru tak som sa rozhodol pre túto cestu)

    public void Save(SaveData saveData)

    {

        string json = JsonUtility.ToJson(saveData, true);

        File.WriteAllText(Application.dataPath + "/save.json", json);

    }

Funkcia **Save()** je vždy volaná pri vypnutí aplikácie.

    void OnApplicationQuit()

    {

        Save(saveData);

    }

## **Funkcia Load()**

Funkcie **Load()** slúžia na načítanie dát zo súboru **save.json** a deserializvanie dát z formátu JSON do triedy **SaveData**.

Text zo súboru **save.json** je načítaný do premennej **json**.

Text v tejto premennej je následne deserializovaný a údaje sú uložené do triedy **SaveData**, ktorú táto funkcia následne vráti.

    public SaveData Load()

    {

        string json = File.ReadAllText(Application.dataPath + "/save.json");

        return JsonUtility.FromJson<SaveData>(json);

    }

# **Práca s dátami pred a po uložení**

V tejto implementácii s dátami pracujeme v skripte **GameManager**.

Na konci funkcie **Awake()** (na konci z dôvodu aby mohol byť objekt zničení aby nedošlo zbytočnému čítaniu zo súboru, ak je objekt duplicitný, keďže sa jedná o singleton a táto kontrola sa odohráva na začiatku funkcie) skontrolujeme či súbor **save.json** existuje (týmto zároveň kontrolujeme či bola hra spustená po prvý krát). Ak súbor existuje načítame dáta do premennej **saveData**, ktorá je inštanciou triedy **SaveData** aby sme následne k dátam mohli pristupovať. Ak súbor neexistuje tak premennú **playerName** v premennej saveData nastavíme na hodnotu „UnknownPlayer“ a dáta uložíme, čím sa zároveň aj vytvorí súbor **save.json**.

        if(!File.Exists(Application.dataPath + "/save.json"))

        {

            SetPlayerName("UnknowPlayer");

            Save(saveData);

        }

        else

        {

            saveData = Load();

        }

Po úspešnom dohraní levelu ja na skripte **GameManager** vždy zavolaná funkcia **LastLevelStats()**, ktorá slúži na dve hlavné veci.

1. Uloží štatistiky z posledného hraného levelu do premennej **lastLevel**, ktorá je typu **LevelData** aby mohli byť zobrazené v scéne **WinScene**, keďže **GameManager** uchováva dáta aj medzi scénami.
2. Porovná skóre získané po dohraní levelu so skóre z predchádzajúceho hrania daného levelu, ak je skóre lepšie, tieto nové dáta nahradia staré (prioritu má skóre, následne sa porovnáva počet hviezdičiek)

public void LastLevelStats(int levelID, int score, int stars)

    {

        lastLevel.levelID = levelID;

        lastLevel.score = score;

        lastLevel.stars = stars;

        if(levelID >= 0)

        {

            LevelData savedLevel = LoadLevelProgress(lastLevel.levelID);

            if((savedLevel.score < lastLevel.score) || (savedLevel.score == lastLevel.score && savedLevel.stars < lastLevel.stars))

                SaveLevelProgress(lastLevel);

        }

    }

Taktiež sa kontroluje či je **levelID** väčšie alebo rovné 0, keďže **levelID** môže byť aj záporné. Záporné **levelID** reprezentuje level vytvorený hráčom a jeho skóre sa neporovnáva ani neukladá.

Funkcia **SaveLevelProgress()** uloží dáta konkrétneho levelu do objektu **saveData**. Z **levelID** sa odčíta 1, keďže počítanie id levelov začína od 1 avšak list levelov začína indexom 0.

    public void SaveLevelProgress(LevelData levelData)

    {

        saveData.levelSaves[levelData.levelID-1] = levelData;

    }

Funkcia **LoadLevelProgress()** vráti dáta konkrétneho levelu z objektu **saveData**. Ak je id levelu 0, vráti dáta posledného hraného levelu, inak uložené dáta levelu.

public LevelData LoadLevelProgress(int levelID)

    {

        if(levelID == 0)

            return lastLevel;

        LevelData loadedLevelData = saveData.levelSaves[levelID-1];

        if(loadedLevelData == null)

            return new LevelData();

        else

            return loadedLevelData;

    }

# **Ukladanie a načítanie dát levelu**

## **Trieda CustomLevelSaveData**

Táto trieda obsahuje dáta dizajnu levelu vytvoreného hráčom. premenná **background** typu integer uchováva id pozadia levelu a list **blocks** obsahuje objekty typu **BlockData**, ktorý sa skladá z premennej **blockType** typu integer, ktorá uchováva id typu bloku a premennej **position** typu Vector3, ktorá uchováva pozíciu bloku. Triedy musia znovu obsahovať atribút **Serializable**.

using System.Collections.Generic;

using UnityEngine;

[System.Serializable]

public class BlockData

{

    public int blockType;

    public Vector3 position;

}

[System.Serializable]

public class CustomLevelSaveData

{

    public int background;

    public List<BlockData> blocks = new List<BlockData>();

}

## **LevelData Scriptable Object**

**LevelDataScriptableObject** obsahuje list prefabov blokov a list pozadí. Používame ho na získanie konkrétneho pozadia alebo bloku podľa id z triedy **CustomLevelSaveData**.

using UnityEngine;

using System.Collections.Generic;

[CreateAssetMenu(fileName = "Data", menuName = "ScriptableObjects/LevelDataScriptableObject", order = 1)]

public class LevelDataScriptableObject : ScriptableObject

{

    [SerializeField] List<GameObject> blocks = new List<GameObject>();

    [SerializeField] List<Sprite> backgrounds = new List<Sprite>();

    public List<GameObject> GetBlocks()

    {

        return blocks;

    }

    public List<Sprite> GetBackgrounds()

    {

        return backgrounds;

    }

}

## **Funkcia Save() pre Level Editor**

Táto funkcia je volaná po kliknutí na tlačidlo Save v scnéne LevelEditor. Zodpovedá za uloženie hráčom vytvoreného levela do súboru JSON.

Ukladanie hráčom vytvorených levelov funguje na princípe jeden súbor pre jeden level. Z tohto dôvodu budú všetky súbory pre dáta levelov uložené v samostatnom adresári.

Ako prvé vo funkcii skontrolujeme či daný adresár existuje ak nie vytvoríme ho.

V ďalšom kroku skontrolujeme či hráč zadal meno levelu, ktoré bude zároveň použité ako meno súboru. Ak meno nezadal funkcia ďalej nepokračuje.

Následne prejdeme pomocou funkcie **foreach** všetky child objekty objektu **grid**, keďže všetky bloky existujú ako child objekty tohto objektu. Pre každý blok vytvoríme nový objekt triedy **blockData** a nastavíme jeho premenné. Ak sa na objekte bloku nenachádza skript **Block** jedná sa o nezničiteľný blok a jeho typ (premennú **blockType**) nastavíme na 0, inak získame typ bloku zo skriptu **Block** nachádzajúcom sa na objekte. Premennú **position** nastavíme na pozíciu bloku a blok pridáme do listu blokov v objekte **saveData** triedy **CustomLevelSaveData**.

Taktiež nastavíme premennú **background** objektu **saveData** podľa pozadia zvoleného hráčom v level editore.

Následne dáta uložíme do súboru ale pri predchádzajúcej implementácii s rozdielom, že do cesty pridáme nami vytvorený adresár a ako názov súboru použijeme meno levelu zadané hráčom.

Nakoniec načítame scénu **CustomLevels**, ktorá obsahuje list existujúcich levelov vytvorených hráčom.

    public void Save()

    {

        if(!Directory.Exists(Application.dataPath + "/CustomLevels"))

            Directory.CreateDirectory(Application.dataPath + "/CustomLevels");

        string levelName = levelNameInput.text;

        if(levelName == "")

            return;

        foreach(Transform existingBlock in grid.gameObject.transform)

        {

            BlockData block = new BlockData();

            if(existingBlock.GetComponent<Block>() == null)

                block.blockType = 0;

            else

                block.blockType = existingBlock.gameObject.GetComponent<Block>().GetBlockID();

            block.position = existingBlock.position;

            saveData.blocks.Add(block);

        }

        saveData.background = background;

        string json = JsonUtility.ToJson(saveData, true);

        File.WriteAllText(Application.dataPath + "/CustomLevels/" + levelName + ".json", json);

        sceneLoader.LoadScene("CustomLevels");

    }

## **Skript CustomLevelListLoader**

V scéne **CustomLevels** zobrazíme list levelov vytvorených hráčom pomocou vytvorenia tlačidiel na objekte **Canvas** vo funkcii **Start()** podľa všetkých súborov nachádzajúcich sa v priečinku **CustomLevels**.

Na začiatku skontrolujeme či daný priečinok vôbec existuje, ak nie funkcia ďalej nepokračuje.

Ak existuje do poľa načítame zoznam názvov súborov (aj s cestou) nachádzajúcich sa v priečinku **CustomLevels**.

Inicializujeme premennú offsetY, ktorá slúži na zmenu pozície tlačidiel v objekt **Canvas**.

Pomocou funkcie foreach prejdeme cez všetky názvy súborov v poli. Pomocou triedy **Path** a funkcie **GetFileName()** z knižnice **System.IO** získame iba názov súboru s príponou bez cesty. Následne pomocou funkcie **GetFileNameWithoutExtension()** získame iba názov súboru bez prípony.

Vytvoríme v scéne objekt tlačidla a nastavíme potrebné parametre ako pozícia, rodičovský objekt a text (názov súboru/levelu) a pridáme event listener, ktorý zavolá funkciu **LoadCustomLevel()** po kliknutí na tlačidlo.

\*poznámka: takáto implementácia spôsobuje, žee pri spustení hry v editore sa v zozname levelov zobrazujú duplicitné tlačidlá s príponou .json z dôvodou, že pri spúšťaní hry v editore sa priečinok CustomLevels nachádza v adresári Assets projektu hry a Unity ku všetkým súborom v tomto adresári vytvára .meta súbory. Napríklad k súboru MôjLevel.json sa vytvorí súbor MôjLevel.json.meta. Toto sa však nedeje po exportovaní hry, tam sa meta súbory nevytvárajú takže som to neriešil.

    void Start()

    {

        if(!Directory.Exists(Application.dataPath + "/CustomLevels"))

            return;

        string[] files = Directory.GetFiles(Application.dataPath + "/CustomLevels");

        int offsetY = 0;

        foreach(string file in files)

        {

            string fileName = Path.GetFileName(file);

            fileName = Path.GetFileNameWithoutExtension(fileName);

            GameObject button = Instantiate(buttonPrefab, transform.position, Quaternion.identity);

            button.transform.parent = canvas.transform;

            button.transform.localPosition = new Vector3(-300, 300 - offsetY, 0);

            button.GetComponent<Button>().onClick.AddListener(() => LoadCustomLevel(fileName));

            button.transform.GetChild(0).GetComponent<TextMeshProUGUI>().text = fileName;

            offsetY += 100;

        }

    }

Funkcia **LoadCustomLevel()** uloží do skriptu **GameManager** názov levelu, ktorý chce hráč načítať a pomocou skriptu **SceneLoader** načíta scénu **CustomLevel**.

    public void LoadCustomLevel(string levelName)

    {

        GameManager.Instance.SetCustomLevelToLoad(levelName);

        sceneLoader.LoadScene("CustomLevel");

    }

## **Skript CustomLevelLoader**

V scéne **CustomLevel** sa na skripte **CustomLevelLoader** vo funkcii **Start()** vykoná načítanie dát levelu.

Funkcia získa názov levelu, ktorý sa ma načítať zo skriptu **GameManager** a načíta dáta tohto levelu do objektu **saveData** typu **CustomLevelSaveData**.

Následne nastaví pozadie levelu získaním správneho spritu zo scriptable objektu **levelData** pomocou id pozadia uloženého v premennej **background**.

A pomocou funkcie **foreach** prejde cez všetky bloky uložené v súbory, vytvorí ich v scéne ako child objekt objektu **Grid** a nastaví ich pozíciu. Správny prefab objketu je rovnako získaný zo scriptable objektu **levelData** pomocou id objektu z premennej **blockType**.

    void Start()

    {

        string levelName = GameManager.Instance.GetCustomLevelToLoad();

        string json = File.ReadAllText(Application.dataPath + "/CustomLevels/" + levelName + ".json");

        saveData = JsonUtility.FromJson<CustomLevelSaveData>(json);

        backgroundImage.sprite = levelData.GetBackgrounds()[saveData.background];

        foreach(BlockData block in saveData.blocks)

        {

            GameObject newBlock = Instantiate(levelData.GetBlocks()[block.blockType], block.position, Quaternion.identity);

            newBlock.transform.parent = grid.gameObject.transform;

        }

    }