Laboratorul 9

Laboratorul acesta se bazează pe proiectul din laboratorul anterior, disponibil aici.

În acest laborator vom lucra cu date persistente.

1 Salvarea scenei

Primul lucru pe care îl vom implementa va fi un sistem de salvare a stării jocului și de revenire la starea anterioară. Vom folosi fișiere *JSON* pentru asta. Vom defini o structură care conține informațiile referitoarea la starea curentă a jocului, pe care o vom serializa într-un *JSON* string pe care îl vom salva într-un fișier. Când jucătorul apasă pe un buton, aceste informații vor fi salvate în fișier, iar la apăsarea altui buton, scena se va reseta la starea salvată anterior. Primul lucru pe care îl vom face va fi să declarăm structura datelor pe care dorim să le salvăm în fișierul *GameController.cs*. Această structură trebuie sa fie [*Serializable*] pentru a putea fi salvată într-un fișier *JSON*.

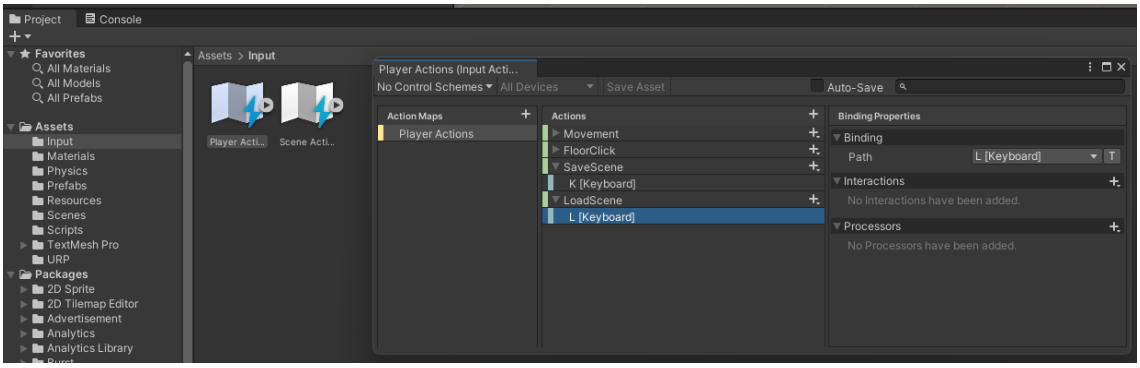
```
[Serializable]
public struct GameSceneState
{
    public Vector3        PlayerPosition;
    public int            Score;
    public float          RemainingTime;
    public List<int>      PickupTypes;
    public List<Vector3> PickupsPositions;
}
```

Vom defini două metode pentru salvarea și încărcarea configurației scenei. Cele două metode vor avea ca argument un obiect de tip *InputAction.CallbackContext* deoarece acestea vor fi apelate la apăsarea butoanelor K, respectiv L de pe tastatură.

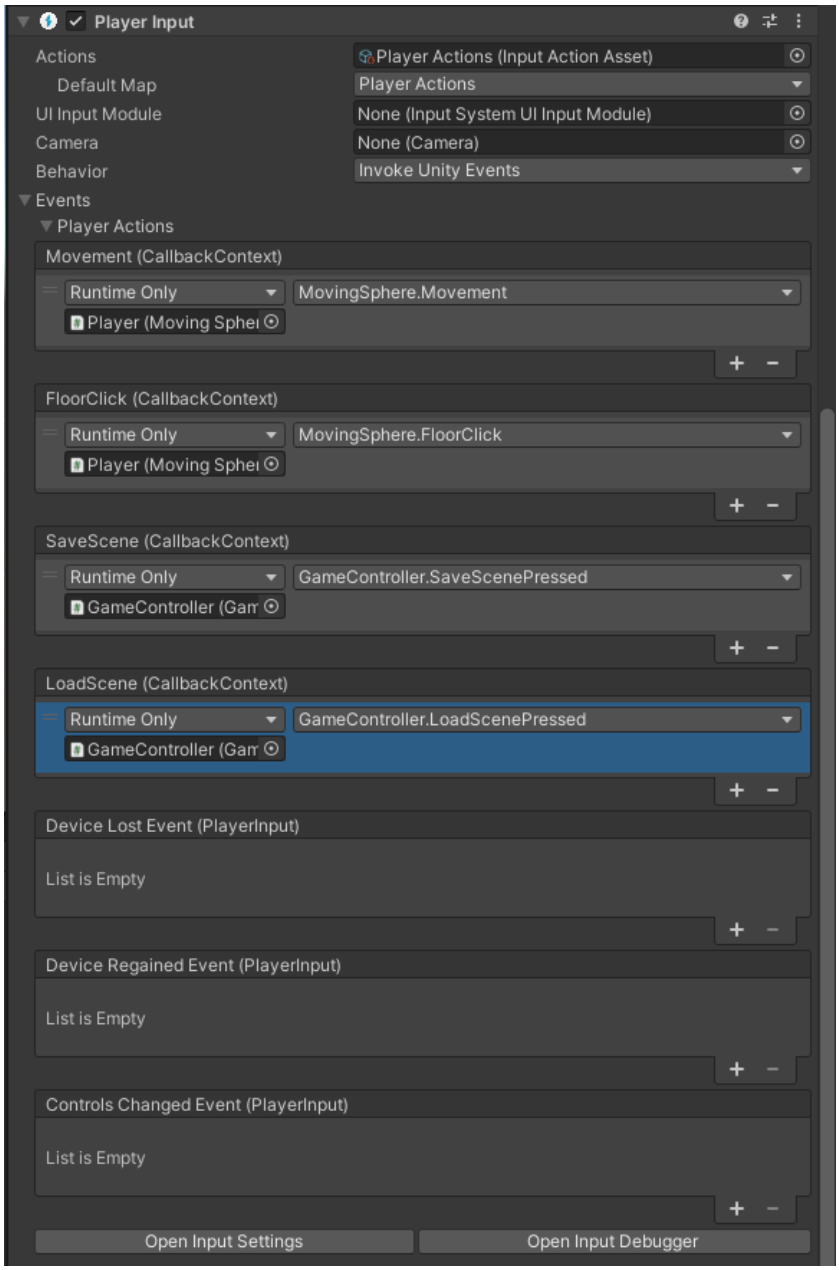
```
...
using UnityEngine.InputSystem;
...
public class GameController : MonoBehaviour
{
    ...
    public void SaveScenePressed(InputAction.CallbackContext context)
    {
        if (!context.started)
            return;

        Debug.Log("Save");
    }

    public void LoadScenePressed(InputAction.CallbackContext context)
    {
        if (!context.started)
            return;
```

```
            Debug.Log("Load");
        }
        ...
}
```

Vom adăuga două *binding*-uri de tip *button* pentru butoanele K și L.

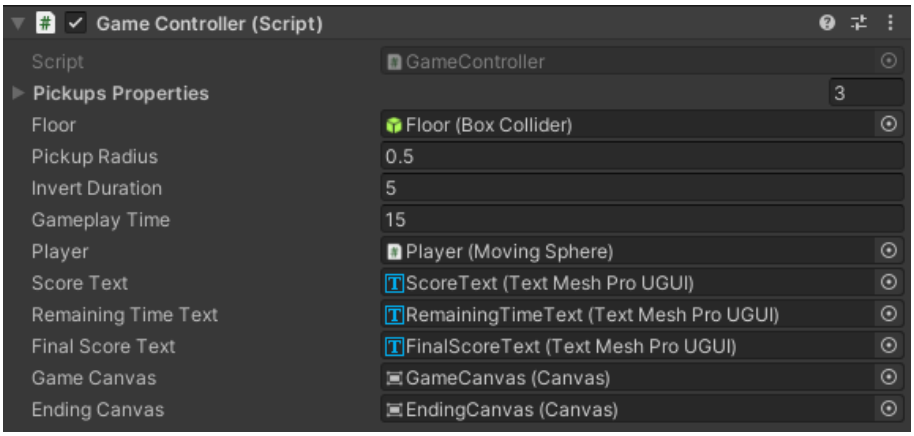


Vom atribui aceste *binding*-uri metodelor definite anterior.



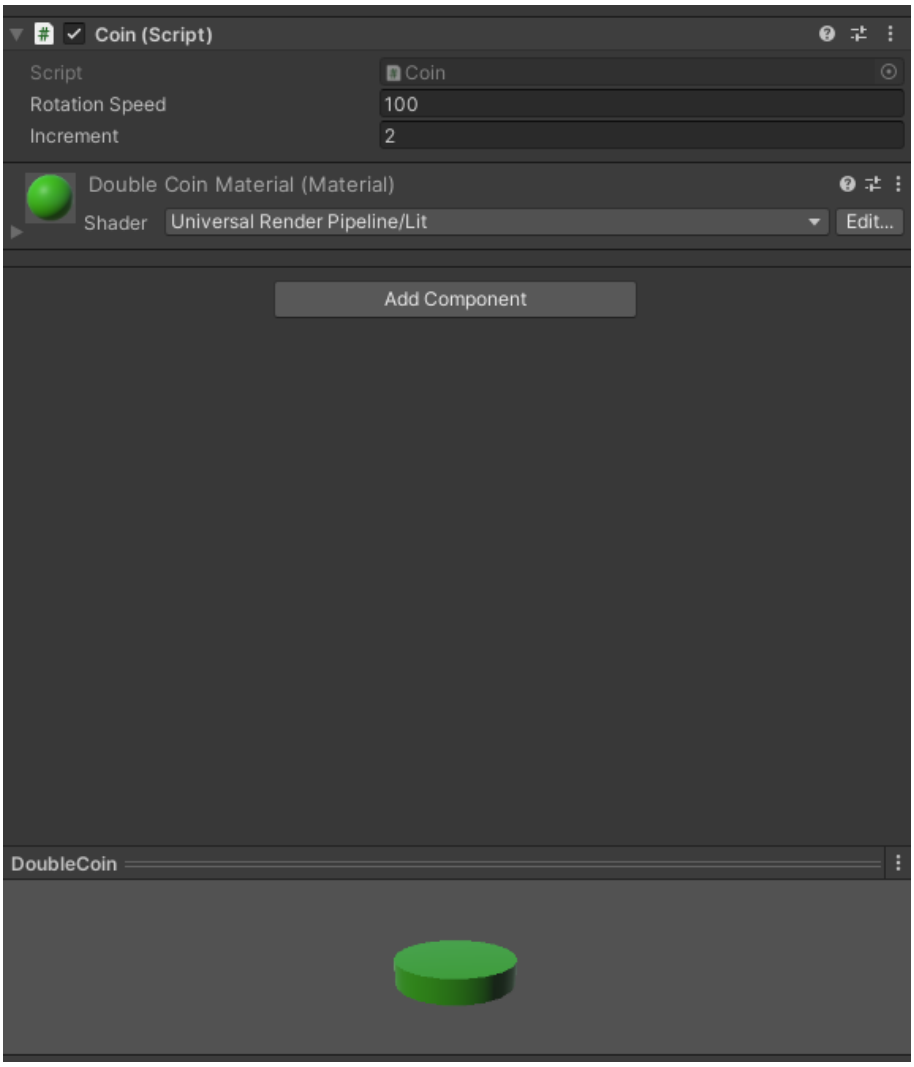
În interiorul metodei *SaveScenePressed* prima dată vom instanția un obiect de tipul *GameSceneState* și vom salva în acesta informațiile necesare. Avem nevoie de informații legate de Player, deci vom lua o referință către acesta.

```
[SerializeField]
private MovingSphere _player;
```



Mai avem nevoie și de valoarea lui *_increment* al clasei *Coin*, așa că vom face această variabilă publică și vom actualiza aceasteia în prefab-ul *DoubleCoin*.

```
public int Increment = 1;
```



Acum putem salva informațiile necesare în structură. În cazul pickup-urilor, vom salva indexul prefab-ului

în *array*-ul *_pickupsProperties* și poziția acestora în scenă. Pentru a afla tipul vom folosi expersii *Linq* și metode de reflection din *C#*.

```
var sceneState = new GameSceneState
{
    PlayerPosition = _player.transform.position,
    Score = Score,
    RemainingTime = _gameplayTime,
    PickupTypes = new List<int>(),
    PickupsPositions = new List<Vector3>()
};

foreach (var pickup in InstantiatedPickups)
{
    var pickupType = 0;
    switch (pickup)
    {
        case Coin current:
            for (var i = 0; i < _pickupsProperties.Length; i++)
                if (_pickupsProperties[i].Prefab is Coin prefabC)
                    if (prefabC.Increment == current.Increment)
                        pickupType = i;
            break;
        case InversePickup:
            pickupType =
                _pickupsProperties.TakeWhile(t => t.Prefab is not InversePickup)
                    .Count();
            break;
    }

    sceneState.PickupTypes.Add(pickupType);
    sceneState.PickupsPositions.Add(pickup.transform.position);
}
```

pentru a genera string-ul *JSON* vom folosi clasa utiliatară din *Unity*, *JsonUtility*. Acteastă clasă este una foarte lightweight și are funcționalități limitate (ex: singurele colecții pe care le poate serializa sunt listele). Pentru proiecte mai complexe, recomandat este să folosiți alte librării pentru manipualrea de fișiere *JSON*. O astfel de librărie este *Newtonsoft.Json*.

```
var json = JsonUtility.ToJson(sceneState);
```

Înainte de a scrie într-un fișier, trebuie să stabilim locația la care acesta se va afla. De regulă este recomandat ca în jocuri, fișierele să fie salvate la o locație relativă față de *Application.persistentDataPath*. Vom defini o proprietate care returnează calea acestui fișier.

```
public string SceneSaveFilePath => Application.persistentDataPath + "/SceneState.json";
```

Acum putem salva conținutul în fișier.

```
using System.IO;
...
public void SaveScenePressed(InputAction.CallbackContext context)
{
    ...
    var json = JsonUtility.ToJson(sceneState);
    File.WriteAllText(SceneSaveFilePath, json);
}
```

Pe Windows, path-ul la care se va salva este următorul:

$C : /Users/\%USER\%/Appdata/LocalLow/\%COMPANY\%/\%PROJECT\%$

Unde $USER$ este utilizatorul curent al calculatorului, $COMPANY$ este o valoare setată în proiectul de $Unity$. Implicit aceasta are valoarea $DefaultCompany$. $PROJECT$ este numele proiectului.

Pentru metoda de *Load* vom reseta scena curentă, și vom seta un flag static care să ne indice că trebuie să preia starea salvată în fișier. Resetăm scena deoarece dorim să oprim instant toate corutinele care rulează în aplicație.

```
private static bool _resetSceneState;

public void LoadScenePressed(InputAction.CallbackContext context)
{
    if (!context.started)
        return;

    if (!File.Exists(SceneSaveFilePath))
        return;

    _resetSceneState = true;
    OnReplayButtonPressed();
}
```

Acum, în momentul încărcării scenei va trebui să restabilim starea din fișierul salvat. Putem folosi metoda *Awake* pentru asta.

```
private void Awake()
{
    if (_resetSceneState)
    {
        var json = File.ReadAllText(SceneSaveFilePath);
        var savedData = JsonUtility.FromJson<GameSceneState>(json);

        _player.transform.position = savedData.PlayerPosition;
        Score = savedData.Score;
        _gameplayTime = savedData.RemainingTime;
        var index = 0;
        foreach (var pickupType in savedData.PickupTypes)
        {
            var position = savedData.PickupsPositions[index];
            SpawnPickup(_pickupsProperties[pickupType].Prefab, position);
            index++;
        }

        _resetSceneState = true;
    }
}
```

Ca această modificare să meargă, vom adăuga un argument opțional metodei *SpawnPickup* pentru a putea specifica manual poziția acestuia și pentru a nu alege una la întâmplare.

```
private void SpawnPickup(Pickup prefab, Vector3? desiredPosition = null)
{
    ...
    if (desiredPosition.HasValue)
```

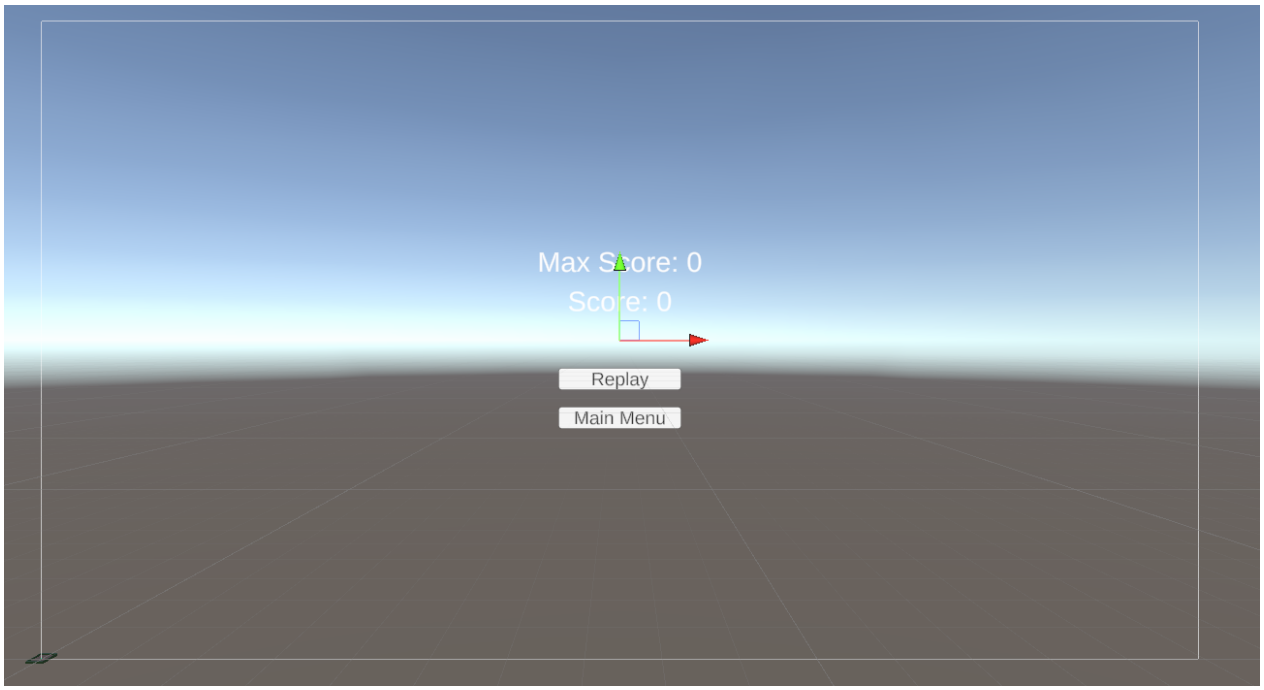
```
        position = desiredPosition.Value;
    ...
}
```

2 Scorul maxim

Unity mai oferă o variantă de a avea persistența datelor, ci anume prin intermediul clasei *PlayerPrefs* care poate salva valori de tip key-value. Noi vom folosi *PlayerPrefs* pentru a salva scorul maxim.

```
public class GameController : MonoBehaviour
{
    ...
    private int _maxScore;
    ...
    private void Awake()
    {
        ...
        _maxScore = PlayerPrefs.GetInt("MaxScore", 0);
    }
    ...
    private void Update()
    {
        ...
        if (_gameplayTime <= 0.0f)
        {
            ...
            if (GameRunning)
            {
                _gameCanvas.gameObject.SetActive(false);
                _endingCanvas.gameObject.SetActive(true);
                if (_score > _maxScore)
                    _maxScore = _score;
                PlayerPrefs.SetInt("MaxScore", _maxScore);
                _finalScoreText.text = "Score: " + _score;
                GameRunning = false;
            }
        }
        ...
    }
}
```

În ecranul de sfârșit vom adăuga un nou text care va afișa scorul maxim.



Vom schimba textul acestui label la sfârșitul jocului, cum am procedat și în laboratorul anterior.

```
public class GameController : MonoBehaviour
{
    ...
    [SerializeField]
    private TMP_Text _maxScoreText;
    ...
    private void Update()
    {
        ...
        if (_gameplayTime <= 0.0f)
        {
            ...
            if (GameRunning)
            {
                ...
                _finalScoreText.text = "Score: " + _score;
                _maxScoreText.text = "MaxScore: " + _maxScore;
                ...
            }
        }
        ...
    }
}
```

3 Exerciții

3.1 Afișați scorul maxim și în *Canvas*-ul jocului.