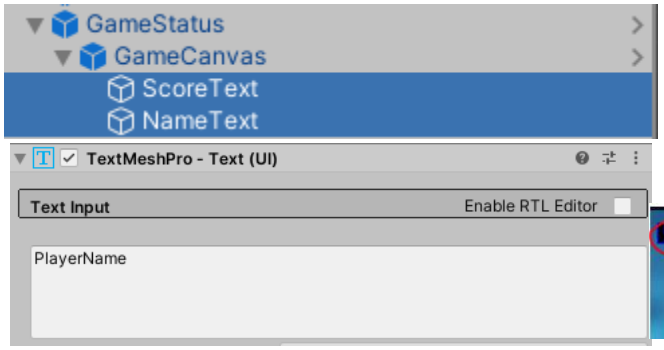
1. Zobrazenie mena hráča v leveli

Keďže pridanie mena hráča je už nastavené (spomenuté v tutoriáli Ukladanie dát v Unity pomcoou XML.pdf), našim cieľom je zobraziť meno aktuálneho hráča v leveli.

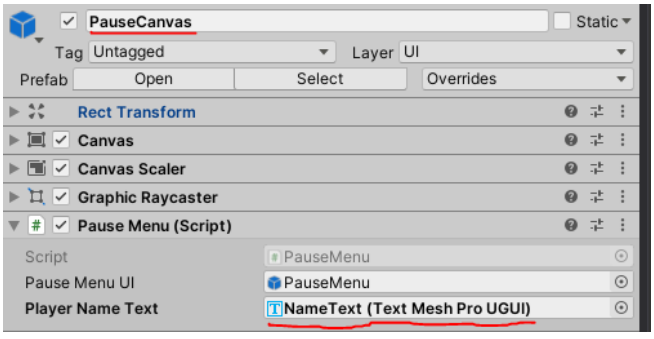


V Leveli1 si pod GameStatusom -> GameCanvasom skopírujeme objekt typu TextMeshPRO, **ScoreText** a premenujeme ho na NameText, pričom zmeníme aj text na PlayerName. Následne ho umiestnime do ľavého horného rohu.

V skripte PauseMenu.cs vytvoríme premennú playerNameText ako SerializedField typu TextMeshProUGUI, do ktorej budeme následne vkladať tento náš objekt NameText. Nesmieme zabudnúť pridať „using TMPro;“ pretože pracujeme s textmeshpro☺.

Následne v metóde Start() vložíme do našej novej premennej playerNameText premennú playername, ktorá bola vytvorená za účelom ukladania a načítania mena (a statov) hráča z predchádzajúceho tutoriálu, takže obsahuje meno hráča ktorý hrá. →

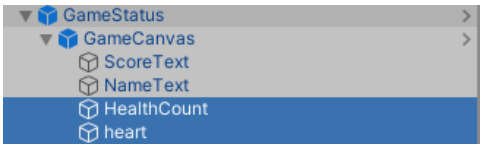
```
using TMPro;

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

    public GameObject pauseMenuUI;

    public static string playername;

    //zobrazit playername
    [SerializeField] TextMeshProUGUI playerNameText;

    Unity Message | 0 references
    private void Start()
    {
        playerNameText.text = playername;
    }
```



← Ďalším krokom je pretiahnutie NameText objektu do PauseMenu skriptu, ktorý sa nachádza v PauseCanvas-e.

2. Životy

Podobne ako objekt NameText vytvoríme objekt HealthCount, ktorý bude slúžiť na zobrazenie životov v hre. Do textu mu napíšeme 0. Umiestnime ho na našej hracej ploche pod NameText. Následne si nakreslíme/stiahneme srdiečko.png a pridáme do nášho projektu čisto ako obrázok, nech je jasné, že to číslo pod menom je život. Vložíme ho do hierarchy pod HealthCount, nech sa nám nestratí. →




```
public class GameStatus : MonoBehaviour
{
    [Range(0.1f, 10f)] [SerializeField] float gameSpeed = 1f;
    [SerializeField] int pointsPerBlockDestroyed = 2;

    [SerializeField] public static int currentScore = 0;

    [SerializeField] TextMeshProUGUI scoreText;

    [SerializeField] bool isAutoPlayEnabled;

    public static int zivot;
```

← Premennú na náš život si vytvoríme v skripte GameStatus(), nech to v budúcnosti využívame pri vytiahnutí uložených dát života.

Túto funkcionalitu však chceme riešiť priamo v Paddle, keď sa vytvorí padlo chcem, aby malo životy. Chcem taktiež aby sa priamo v ňom pridávali a odoberali životy.

Preto si v skripte Paddle.cs vytvorím premennú

```
[SerializeField] TextMeshProUGUI healthCount;
```

Do ktorej budem neskôr vkladať náš nový objekt HealthCount. Taktiež potrebujeme pridať „using TMPro;“ , pracujeme s textmeshpro ☺

Keď máme našu premennú, potrebujeme do nej keď sa paddlo na začiatku naštartuje priradiť život, ktorý je definovaný v GameStatus-e. Preto v metóde Start() v skripte Paddle.cs spravíme nasledovné →

```
void Start()
{
    healthCount.text = GameStatus.zivot.ToString();
}
```

Túto funkcionalitu, odoberanie životov, však nechceme meniť v metóde Update() keď sa nám updatuje paddlo, ale keď loptička padne do LoseCollider-a. Preto si v skripte Paddle.cs vytvoríme funkciu

```
1 reference
public void AddToHealth()
{
    healthCount.text = GameStatus.zivot.ToString();
}
```

`AddToHealth()`, ktorú zavoláme teda z toho LoseCollider.cs ☺. Bude obsahovať to isté ako Start() metóda, ale zdalo sa mi nevhodné volať v LoseCollider-i metódu Start() z iného skriptu.

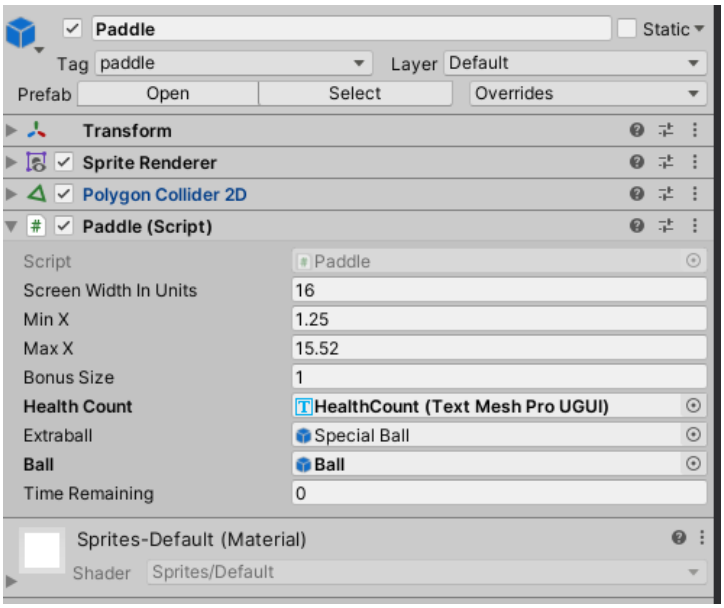
Tak si teda otvoríme LoseCollider.cs, pridáme si premenné,

```
public Ball lopticka;
public Paddle padlo;
```

```
if (GameStatus.zivot != 0)
{
    GameStatus.zivot--;
    padlo.AddToHealth();
    lopticka.hasStarted = false; //kvoli metoode update v ball.cs
    lopticka.Update();

}
else { SceneManager.LoadScene("GameOver"); }
```

← a metódu `OnTriggerEnter2D(Collider2D collision)` upravíme ako na obrázku.

(vidíme, že tam voláme aj naše AddToHealth z Paddle.cs)



← Potom si nesmieme zabudnúť pretiahnuť HealthCount Object do Paddle Skriptu ☺ Nevšímame si BonusSize, Ball, ExtraBall a Time Remaining, to si vysvetlíme v kapitole Bonusy.

Musíme však ešte zabezpečiť ukladanie a načítanie života do/zo súboru. Do skriptu ActorData.cs pridáme nasledovné, pre vytvorenie elementu health. →

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

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

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

    [XmlElement("Health")]
    public int health;
}
```

V skripte PauseMenu.cs upravíme metódy SaveDatas() a LoadData() nasledovne:

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

    SaveData.SaveActor(data);
}

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

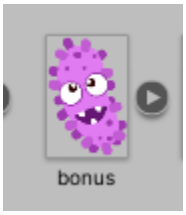
Vidíme že život ťaháme zo skriptu GameStatus, takže preto sme doňho pridávali na začiatku premennú zivot. Nemôžeme to ťahať priamo z Paddla, pretože keď dáme v StartScene Load Game, paddlo ešte vytvorené nemáme – vytvára sa až v Leveli1, preto nám to nenačíta životy ☺

Pre správnosť je ešte potrebné premazať existujúci dokument actors.xml, ktorý sa nachádza v priečinku Assets -> Resources, aspoň tak, že vymažeme všetkých Actors :

```
<?xml version="1.0"?>
<ActorCollection xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Actor>

  </Actor>
</ActorCollection>
```

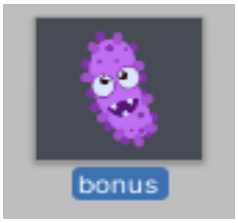
A znova hru odohrať na pár krát, aby sme naťahali nejaké dáta pre nasledujúcu kapitolu Top 5 hráčov, prípadne dáta pridať ručne.

3. Bonusy – zvýšenie počtu loptičiek a zmena veľkosti paddla



← Prvú vec čo spravíme je, že si nájdeme/vytvoríme obrázok bonusu, ktorý bude padať zo zničeného Blocku. Ja som si stiahla free png image malej baktérie.

Vložíme si ho medzi naše Sprites a vložíme ho na našu scénu, upravíme jeho veľkosť a rovno ho vložíme ho do prefabov a vymažeme. →



V prefaboch mu pridíme Tag -> „Collectables“ priradíme Circle Collider 2D v ktorom zaškrtneme IsTrigger a pridáme RigidBody 2D. Inak to neupravujeme. Následne mu priradíme nový skript Bonus.cs.

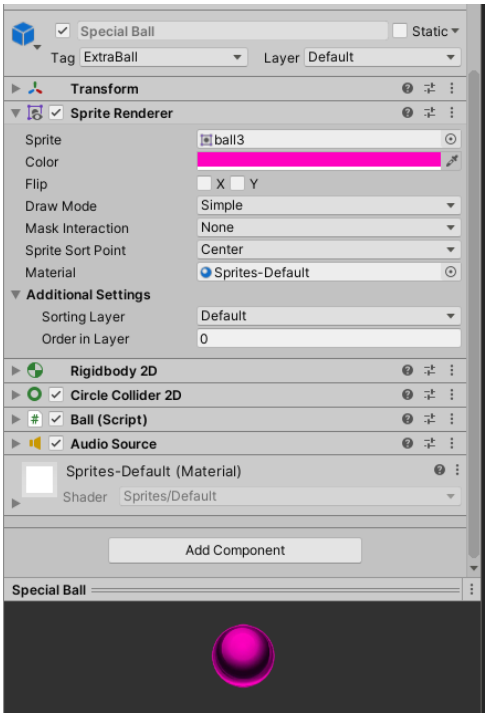
V scripte Bonus.cs v metóde Update() potrebujeme nastaviť, aby padal. Pridáme tam preto `transform.position += new Vector3(0,(-0.01f),0);`

Vrátime sa do unity, a paddlu pridáme tag “paddle”. Robíme to preto, aby sa bonus po stretu s paddlom vymazal, ale aby sa nevymazal keď sa stretne s loptičkou. Vrátime sa do scriptu Bonus.cs, a vytvoríme funkciu `private void OnCollisionEnter2D(Collision2D collision),` v ktorej definujeme ak sa bonus dotkne paddla, zničí sa.

Metódy Update() a OnCollisionEnter2D vyzerajú nasledovne:

```
public class Bonus : MonoBehaviour
{
    Unity Message | 0 references
    void Update()
    {
        transform.position += new Vector3(0,(-0.01f),0);

    }

    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (tag == "paddle")
        {
            Destroy(gameObject);
        }
    }
}
```

Padajúci Bonus máme takto vytvorený (keď ho umiestnime naspäť na scénu, vidíme, že padá a pri kontakte s paddlom sa zničí).



Čo potrebujeme je ale to, že keď sa dotkne bonus paddla, buď sa vytvoria ďalšie loptičky (v pozícií hlavnej loptičky), alebo sa zväčší paddlo.

Pre začiatok si duplikujeme Ball v prefabe, premenujeme ju na Special Ball, zmeníme jej pre odlišnosť farbu a pridáme jej nový tag ExtraBall →

Teraz k funkcionalite. Otvoríme si script Paddle.cs, a prejdeme si k metóde `OnTriggerEnter2D(Collider2D collision)`. Ak tag (bonusu) je Collectables, tak by sa malo vykonať nasledovné:

- Randomne sa vyberie hodnota od 0 do 2 floatov
- Ak je táto hodnota väčšia alebo rovná 1 tak sa randomne vyberie počet loptičiek, ktoré sa majú vytvoriť, a pre každú z nich sa nastaví pozícia na pozícií hlavnej loptičky
- Ak je táto hodnota menšia ako 1 a ak sa ešte paddlo nezväčšilo, zväčší sa paddlo na 10 sekúnd

Pre toto potrebujeme nové premenné:

```
//bonusy
[SerializeField] private GameObject extraball;
[SerializeField] private GameObject ball;
bool bonusStarted = false;
public float timeRemaining;
public float BonusSize = 1f;
```

A metóda vyzerá nasledovne:

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.tag == "Collectables")
    {

        float bonusValue = Random.Range(0f, 2f);//vloz co sa ma stat, takze random 1 a 2 moznost - bud zvacsi padlo alebo prida loptick
        Debug.Log(bonusValue);
        if (bonusValue >= 1f)
        {
            int numberNewBalls = Random.Range(2, 4);

            for (int i = 0; i < numberNewBalls; i++)
            {
                GameObject newBall = Instantiate(extraball, transform.position, Quaternion.identity) as GameObject;
                Vector2 ballPos = new Vector2(ball.transform.position.x, ball.transform.position.y);
                newBall.transform.position = ballPos;
            }
        }
        else
        {
            if (!bonusStarted)
            {

                bonusStarted = true;
                timeRemaining = 10;

                transform.localScale += new Vector3(BonusSize, 0, 0);
                minX += BonusSize * 1.2f;
                maxX -= BonusSize * 1.2f;
            }
        }
    }

    Destroy(collision.gameObject);
}
```

Nadväzne na to potrebujeme upratať v skriptoch. V scripte Ball.cs nemôžeme transformnúť pozíciu v metóde Start() ani Update() pre Extra ball:

```
void Start()
{
    if (gameObject.tag != "ExtraBall")
    {
        paddleToBallVector = transform.position - paddle1.transform.position;
    }
    myAudioSource = GetComponent<AudioSource>();
    myRigidBody2D = GetComponent<Rigidbody2D>();

}
```

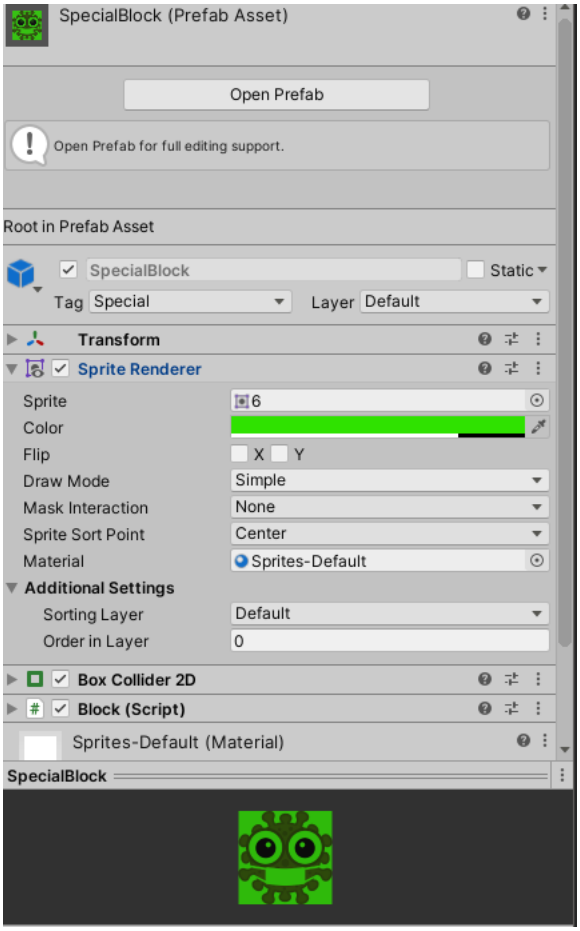
```
public void Update()
{
    if (gameObject.tag != "ExtraBall")
    {
        if (hasStarted != true)
        {
            LockBallToPaddle();
            LaunchOnMouseClick();
        }
    }
}
```

A taktiež nechceme, aby sa odrážala od bonusov:

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (tag != "Collectables")
    {
        Vector2 velocityTweak = new Vector2(UnityEngine.Random.Range(0f, randomFactor), UnityEngine.Random.Range(0f, randomFactor));
        if (hasStarted)
        {
            AudioClip clip = ballSounds[UnityEngine.Random.Range(0, ballSounds.Length)];
            myAudioSource.PlayOneShot(clip);
            myRigidBody2D.velocity += velocityTweak;
        }
    }
}
```

Stále však nemáme vyriešené to, že potrebujeme zabezpečiť, že sa náš Bonus odniekiaľ vynorí – keď sa zničí určitý Block. Preto si duplikujeme Block_1 a pre odlíšenie mu zmeníme farbu: →

Pridáme mu ale iný tag: „Special“. Tým odlíšime, že je to špeciálny block a niečo sa stane, keď ho trafíme. MaxHits v Inspectore v scripte Block.cs ponecháme na hodnote 1.



Pre začiatok si v scripte Block.cs si vytvoríme premennú `[SerializeField] private GameObject bonus;` do ktorej si budeme vkladať náš game object Bonusu.

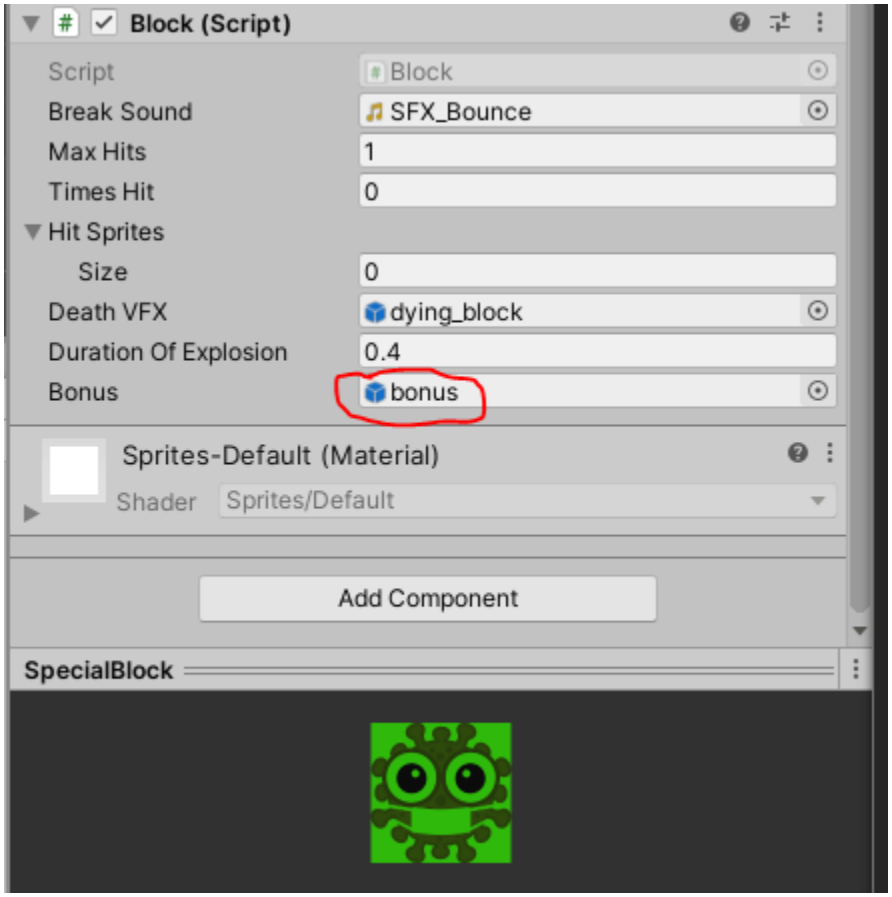
Naše Special Blocky nechceme handlovať pri Hite tak isto, ako normálne blocky, ktorých „hittovanie“ zabezpečuje metóda HandleHit(). Vytvoríme si preto metódu HandleSpecial, ktorá bude obsahovať taktiež metódu DestroyBlock(), ale nechceme, aby sa tento block hittoval viac krát (vždy chcem aby sa zničil po 1 Hite), takže nič iné z metódy HandleHits tam nebude. Potrebujeme však zabezpečiť, aby sa pred zničením Special Blocku vytvoril náš bonus. To zabezpečíme takto:

```
1 reference
void HandleSpecial()
{

    GameObject newBonus = Instantiate(bonus, transform.position, Quaternion.identity) as GameObject;
    DestroyBlock();

}
```

Následne upravíme metódu `OnCollisionEnter2D(Collision2D collision)` tak, aby keď sa normálny block, s tagom „Breakable“ dostal ku kolízii, zavolá sa metóda HandleHit(), ale keď sa Special Block s tagom „Special“ dostal ku kolízii, zavolá sa metóda HandleSpecial():

```
private void OnCollisionEnter2D(Collision2D collision)
{
    if (tag == "Breakable")
    {
        HandleHit();
    }
    if (tag == "Special")
    {
        HandleSpecial();
    }
}
```

Následne si do prefab SpecialBlock vložíme náš prefab Bonus:

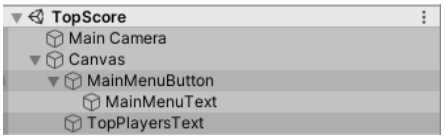


Umiestnime si naše Special Blocks do hracej plochy, a hotovo☺

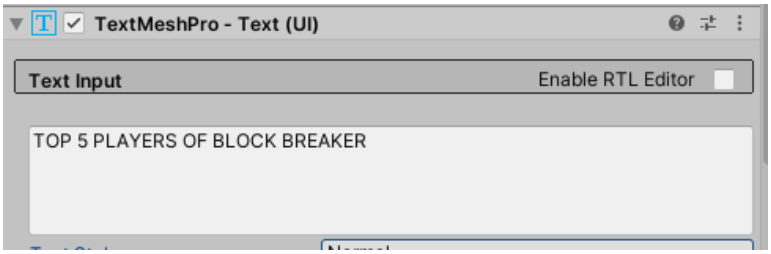
4. Top 5 hráčov

Začneme duplikovaním scény GameOver a premenovaním ju na TopScore.

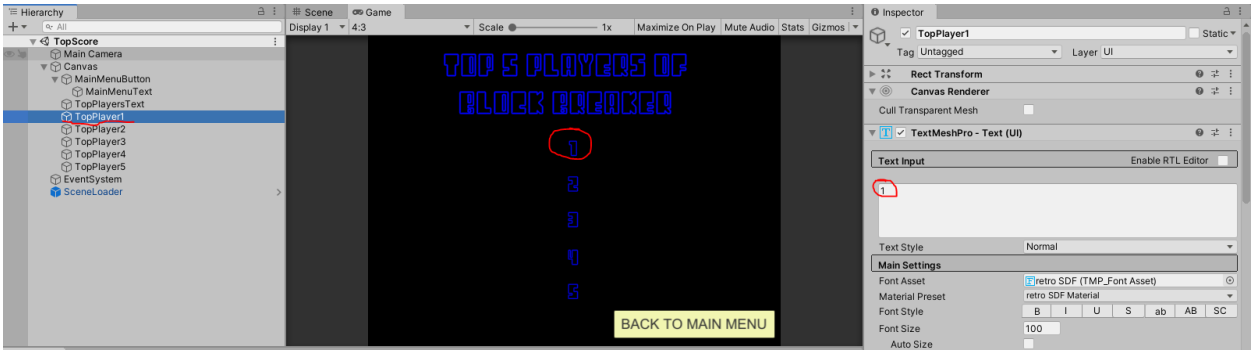
V hierarchií si prepíšeme PlayAgain Button na **MainMenuButton** (rovnako aj popisky k buttonu) a GameOverText na **TopPlayersText,** →



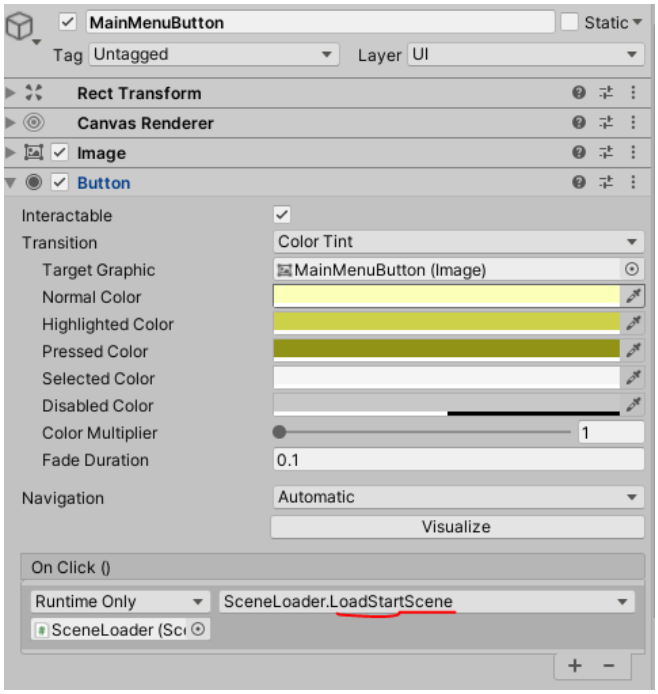
v ktorom si zmeníme aj Text Input na**:**



Následne si skopírujeme TopPlayerText, pohráme sa s veľkosťami a umiestnením, až nám vyjde niečo takéto :

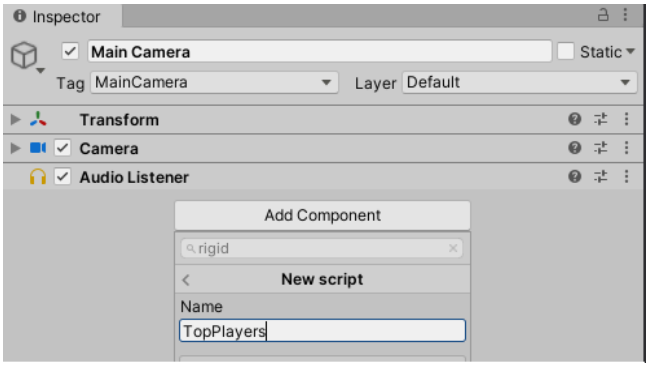


MainMenuButton je nastavený správne, presunie nás na StartScene ☺



Teraz potrebujeme vytiahnuť najlepších 5 hráčov z actors.xml dokumentu. To spravíme v novom skripte TopPlayers.cs, ktorý umiestnime na MainCameru, a plus budeme potrebovať pomoc od SaveData.cs skriptu.

Takže priradíme Main Camera skript →




```
using TMPro;

Unity Script | 0 references
public class TopPlayers : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI player1;
    [SerializeField] TextMeshProUGUI player2;
    [SerializeField] TextMeshProUGUI player3;
    [SerializeField] TextMeshProUGUI player4;
    [SerializeField] TextMeshProUGUI player5;
```

← A v ňom vytvoríme premenné player1 – player5 (keďže chceme top 5 hráčov), taktiež sú typu `TextMeshProUGUI` a ako `[SerializeField],` pretože budeme do tohto skriptu vkladať objekty TopPlayer1 – TopPlayer5, ktoré sú typu TextMeshPro. Nezabudneme definovať „`using TMPro;`“☺

Potrebujeme však tých hráčov nejak dostať z actors.xml. Preto si v SaveData.cs skripte vytvoríme novú metódu TopPlayers(), ktorú budeme volať v TopPlayers.cs skripte.

```
1 reference
public static List<ActorData> TopPlayers()
{
    ActorContainer actors = LoadActors(Path.Combine(Application.dataPath, "Resources/actors.xml"));

    List<ActorData> unsorted = new List<ActorData>();

    foreach (ActorData data in actors.actors)
    {

        unsorted.Add(data);

    }

    List<ActorData> sorrted = unsorted.OrderBy(o => o.score).ToList();
    Debug.Log(sorrted);
    return sorrted;
}
```

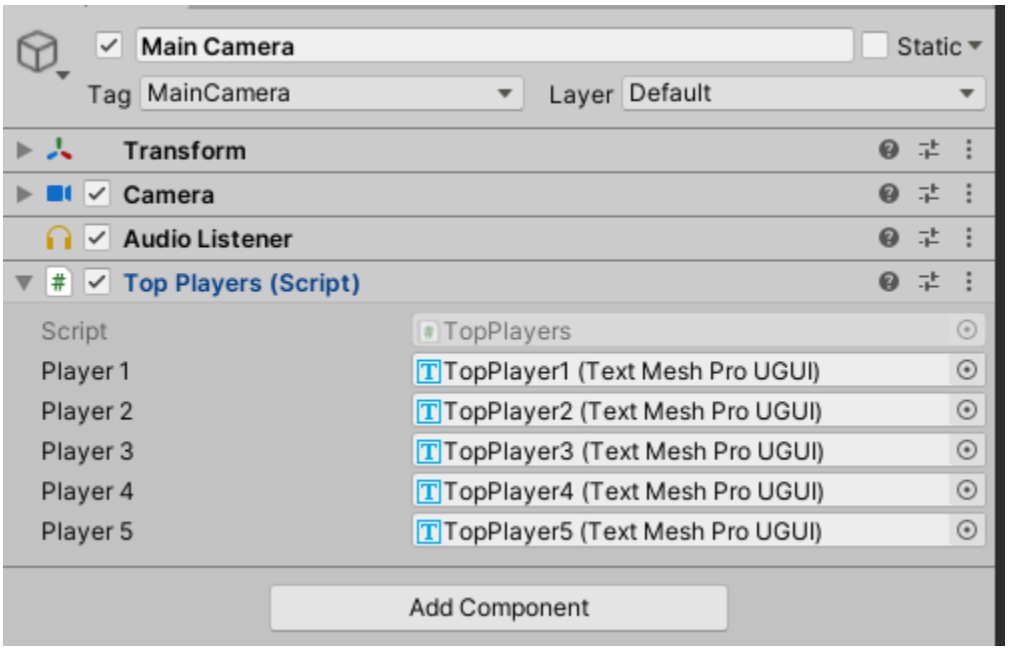
Metóda TopPlayers() v skripte SaveData.cs bude typu `List<ActorData>` , pretože chceme, aby vracala list obsahujúci všetky Actor Data z .xml dokumentu. Návratovú hodnotu bude mať usporiadaný list hráčov (meno, level, skore, zivoty). V tejto metóde potrebujeme vytiahnuť hráčov do ActionContainer-a pomocou metódy LoadActors(), definovanej vyššie v tomto skripte. Následne vytvorím nezoradený `List<ActorData> unsorted new List<ActorData>();`, ktorý v cykle for naplníme všetkými hráčmi, ale ešte ich nezoraďujeme. Zoraďujeme až neskôr, kde si vytvoríme nový `List<ActorData> sorted`, a pomocou OrderBy (závisle od skóre) naťaháme od najnižšieho po najvyššie skóre hráčov. Vraciame, ako už bolo spomínané, sorted list ☺

Túto metódu potrebujeme zavolať v skripte TopPlayers.cs. Zavoláme ju priamo v metóde Start(), pretože chceme, nech ten zoznam vidíme hneď, keď sa prepneme na TopScore scénu.

Vytvoríme si nový `List<ActorData> players`, do ktorého si pomocou `SaveData.TopPlayers();`načítame našich zoradených hráčov. Následne do TMP(textmeshpro) premenných vložíme podľa indexov (4 = najlepší, 0 = najhorší) hráčov a ich skóre na zobrazenie v tejto premennej. Vyzerá to takto:

```
public class TopPlayers : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI player1;
    [SerializeField] TextMeshProUGUI player2;
    [SerializeField] TextMeshProUGUI player3;
    [SerializeField] TextMeshProUGUI player4;
    [SerializeField] TextMeshProUGUI player5;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        List<ActorData> players = SaveData.TopPlayers();
        player1.text = "1st place: " + players[4].name + ", Score: " + players[4].score;
        player2.text = "2nd place: " + players[3].name + ", Score: " + players[3].score;
        player3.text = "3rd place: " + players[2].name + ", Score: " + players[2].score;
        player4.text = "4th place: " + players[1].name + ", Score: " + players[1].score;
        player5.text = "5th place: " + players[0].name + ", Score: " + players[0].score;
    }
```

Tým je funkcionalita dokončená, a zostáva len priradiť do skriptu TopPlayers v Main Camera naše TMP objekty.



Výsledok vyzerá nejak takto:

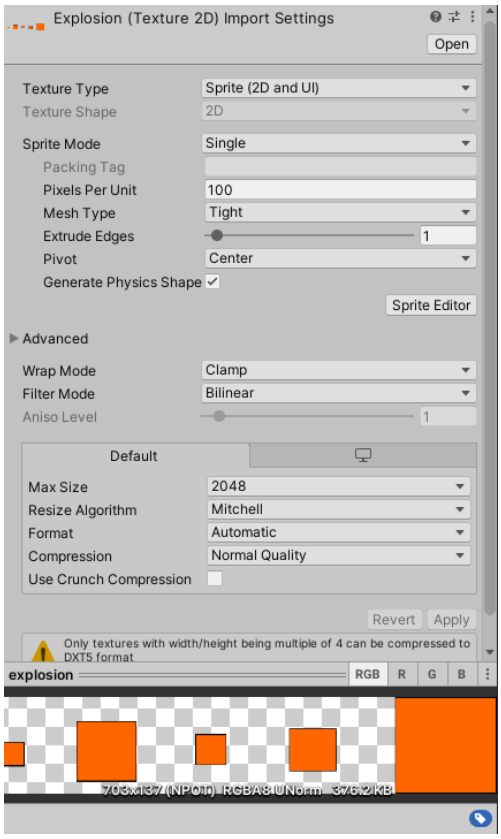
5. Grafika

a. Particle effect zrušenia bloku

Nový particle effect zrušenia bloku je potrebné začať pred tým, než zabudneme a nebudeme chápať divné správanie, priamo v skripte Block.cs. Prejdeme si do metódy DestroyBlock(), a zakomentujeme si už existujúci particle effect Sparkles a taktiež `//[SerializeField] GameObject blockSparkleVFX;`:

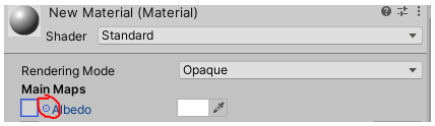
```
2 references
private void DestroyBlock()
{
    AudioSource.PlayClipAtPoint(breakSound, Camera.main.transform.position);
    Destroy(gameObject);
    level.BlockDestroyed();
    FindObjectOfType<GameStatus>().AddToScore();
    //TriggerSparklesVFX(); //metoda ktora vytvori particle effect
    GameObject explosion = Instantiate(deathVFX, transform.position, transform.rotation);
    Destroy(explosion, durationOfExplosion);
}


/*    private void TriggerSparklesVFX()
    {
        GameObject sparkles = Instantiate(blockSparkleVFX, transform.position,transform.rotation);
        //vytvori instanciu na urcenej pozicii s prislusnou rotaciou
        Destroy(sparkles, 1f);//sposobi znicenie instancie po 1 sekunde
    }
*/
1 reference
```

Následne si vytvoríme vlastný Sprite pre explosion. Ja som vytvorila 5 kusov oranžových štvorčekov, ktoré budú preblikávať. →

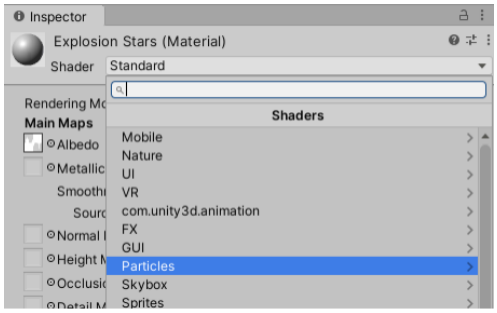


Následne je potrebné vytvoriť si materiál. Medzičasom som si vytvorila nový priečinok v Assets s názvom Effects, do ktorého som vložila vyššie vytvorený sprite. Teraz v tom priečinku vytvorím Material s názvom dying_block, ktorému neskôr tento sprite priradím.

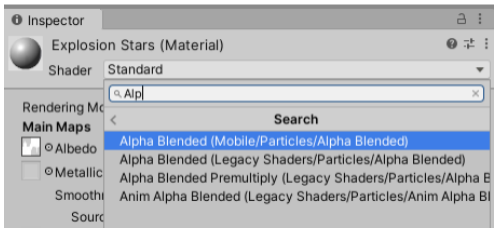
Kliknutím na Albedo →

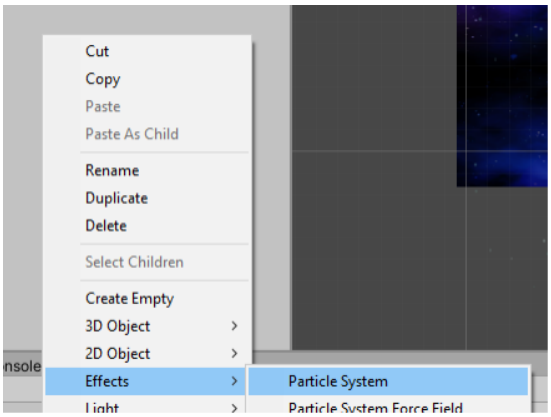


pridám môjmu materiálu môj sprite,

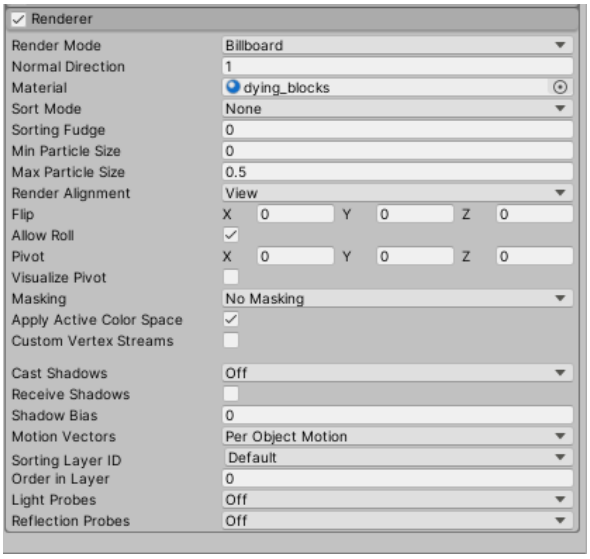


vyhľadám si to v zozname. Následne potrebujem zmeniť Shader. Kliknem si na Shader, nalistujem Particles záložku a vyhľadám Alpha Blended (Mobile/....., a zvolím ho. Dovolím si ukradnúť screenshot z manuálu LaserDefender, lebo to neviem rozumne odfotiť (nefunguje mi printscreen tlačidlo, a SnippingTools mi to vždy zatvorí) →



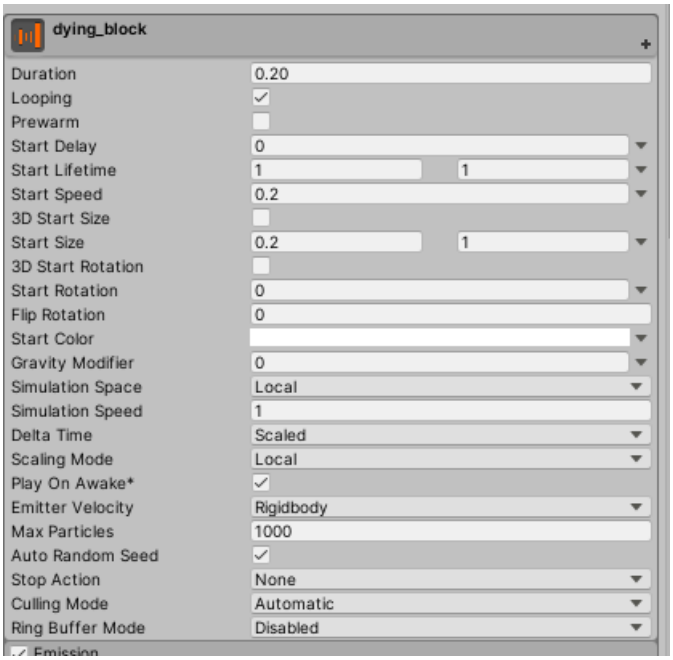


← Keď máme hotovo, vytvoríme si particle system (ukradnutý obrázok) a pomenujeme ho taktiež dying%block, a priradíme mu materiál dying_block (v záložke Renderer)

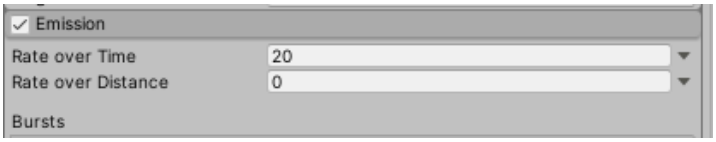


Následne si odklikneme záložku Texture Sheet Animation, nastavíme si Tiles na 4 a 1 (pretože máme síce 5 kocočiek, ale krajšie ich rozseká takto po stĺpcoch, a máme ich len v 1 riadku). Nastavím si Cycles na 3, krajšie to preblikáva počas života animácie.

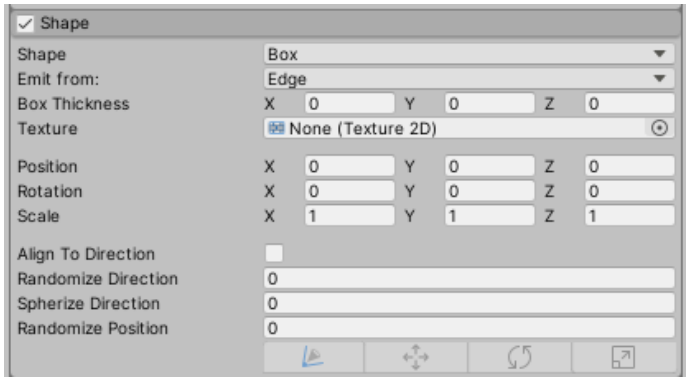
Prejdeme si na začiatok Particle System, a pomeníme nastavenia hodnôt podľa toho, ako sa nám to páči:



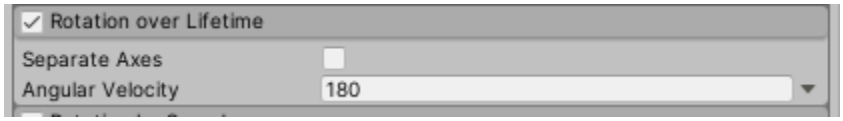
Neskôr som nastavila Emission Rate Over Time na 20, najlepšie mi tak emitovali častice v okolí.



Zakliknem aj Shape, kde zmením Shape na Box:



Premýšľala som nad rotáciou, a nakoniec som ju nastavila na 180, nech sa to aspoň nejak točí ☺



Po tomto si môžeme particle system pretiahnuť do prefabov a vymazať ho z hierarchie.

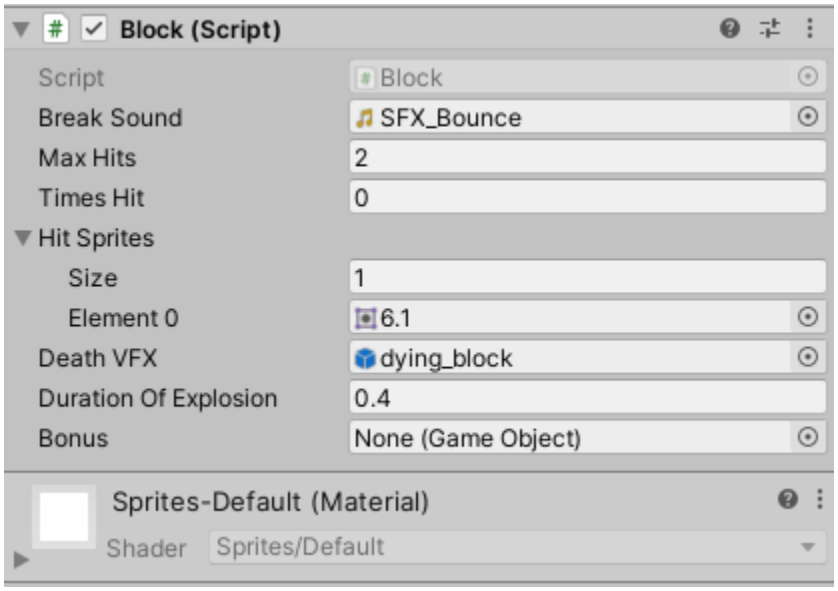
Pre nasadenie particle systému si otvoríme skript Block.cs, a pridáme doňho dve premenné:

```
[SerializeField] GameObject deathVFX;
[SerializeField] float durationOfExplosion = 0.4f;
```

Potom metódu DestrozBlock() upravíme nasledovne:

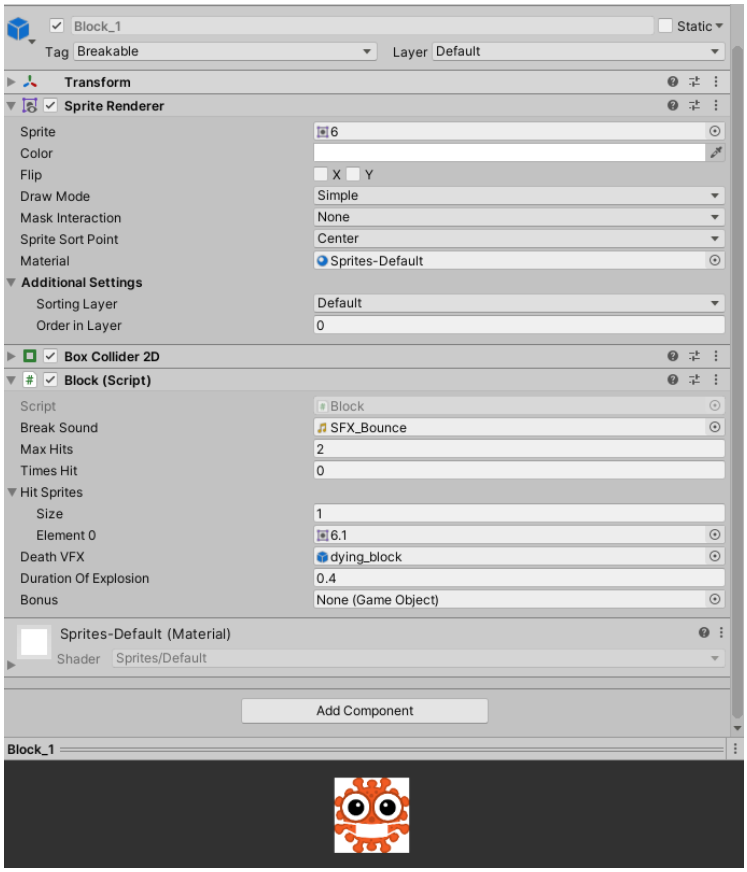
```
private void DestroyBlock()
{
    AudioSource.PlayClipAtPoint(breakSound, Camera.main.transform.position);
    Destroy(gameObject);
    level.BlockDestroyed();
    FindObjectOfType<GameStatus>().AddToScore();
    //TriggerSparklesVFX(); //metoda ktora vytvori particle effect

    GameObject explosion = Instantiate(deathVFX, transform.position, transform.rotation);
    Destroy(explosion, durationOfExplosion);
}
```

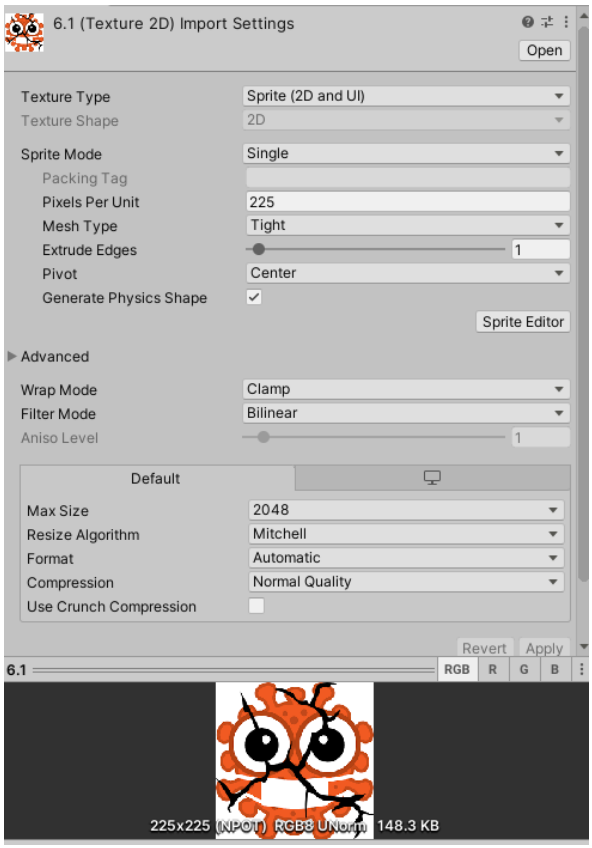
Vrátime sa do Unity, a objektu Block priradíme do Death VFX v Block skripte náš particle system dying_block:



b. Rozbitie bloku na 2 etapy

Vytvoríme si nový sprite upravením obrázku č. 6 na „potrhlinovaný“ obrázok blocku, v ktorom bolo potrebné nastaviť pixels pr unit na 225 (zahovala som veľkosť pôvodného obrázku). →





← Následne zmeníme v Block_1 Sprite na sprite č. 6, pretože ten sme dostali zadaný. Nastavíme si v Inspectore v Block (Script) Hit Sprites -> Size na 1, a elementu 0, ktorý nám tam vyskočí, prídáme náš novovytvorený sprite s trhlinou. Dodatočne je potrebné nastaviť MaxHits na 2, aby sme zrušili Block až keď do neho 2x ťukneme ☺