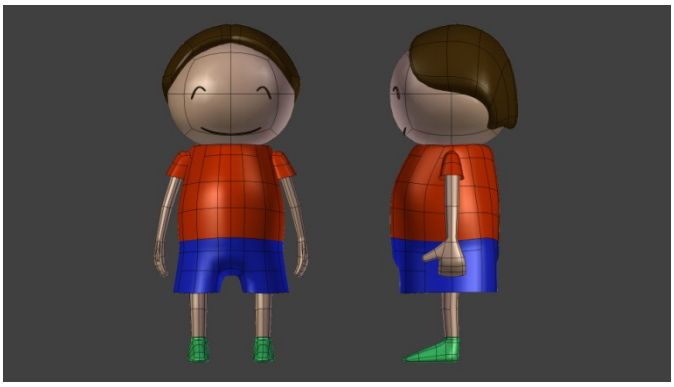
E’ stato svolto un gioco con Unity 2D e relativa modellazione dei personaggi con Blender.

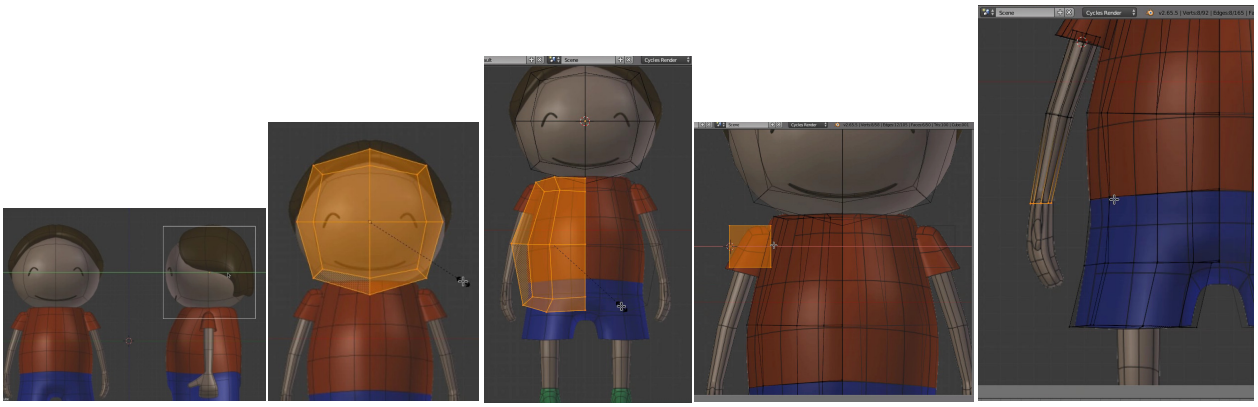
Su Blender è stata fatta la modellazione, il rigging e l’animazione dei personaggi. Poi, una volta decisa la logica del gioco, esso è stato sviluppato su Unity.

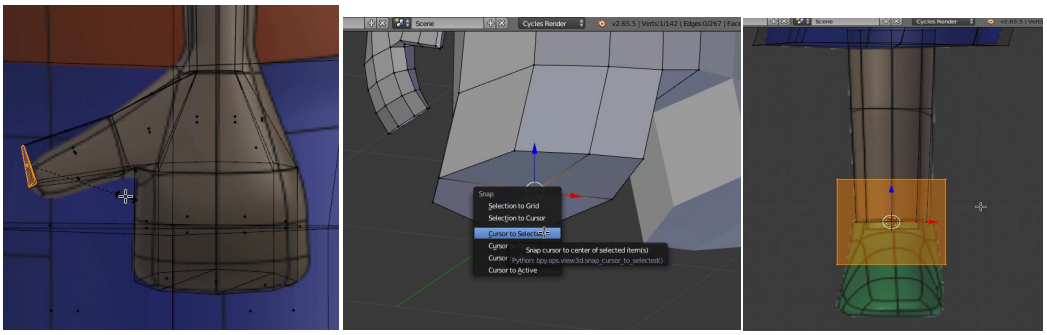
Parte 1: Utilizzo di Blender

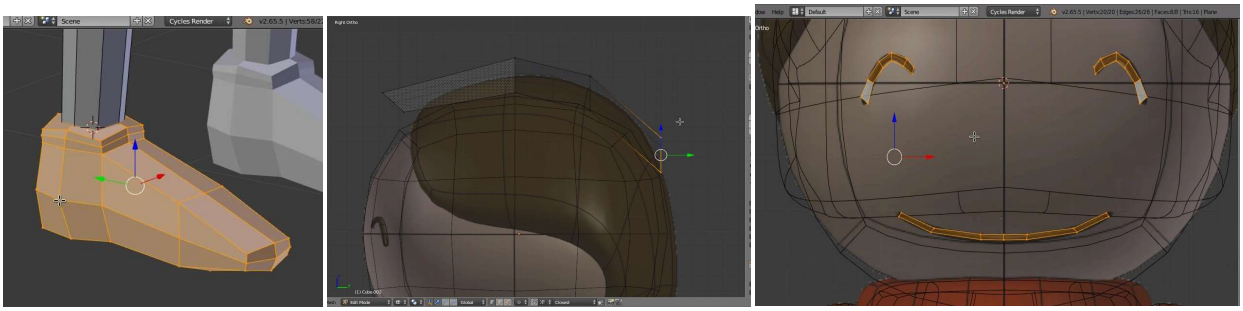
Si parte della modellazione del personaggio caricando una sua vista frontale e di profilo come sfondo della 3d view di Blender.



Partendo da un cubo opportunamente scalato, usando estrusioni, subdivide smooth, spostamento di vertici si arriva alla modellazione della struttura del personaggio:







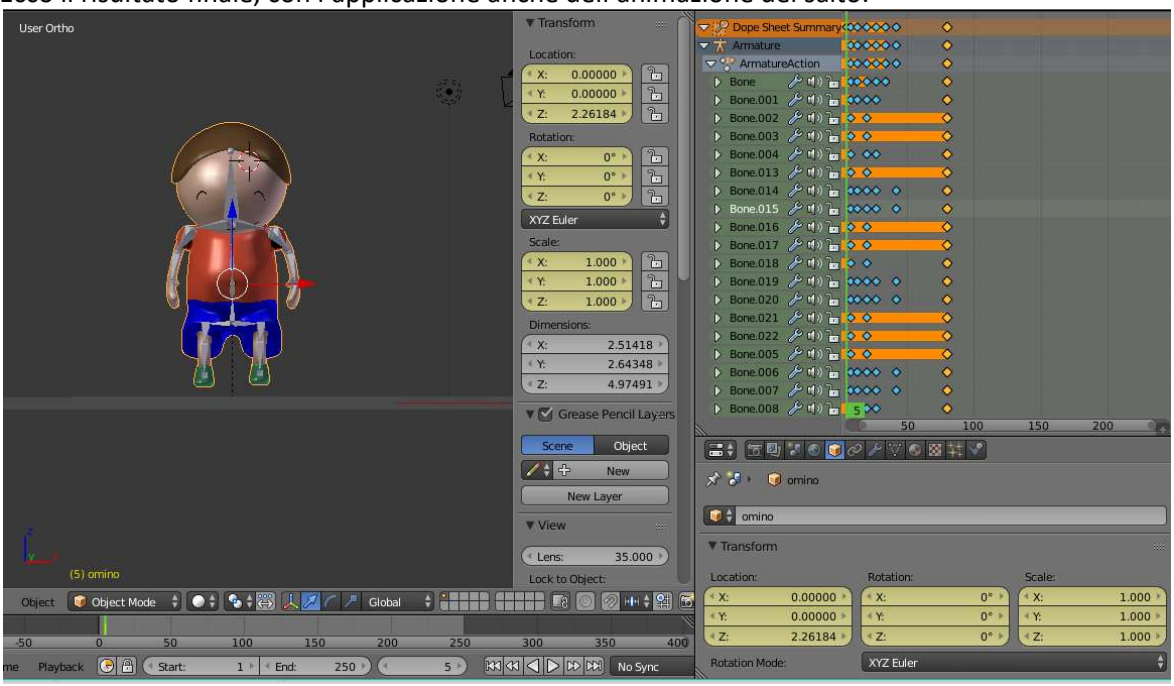
Blender ha inoltre una funzione chiamata mirror che indica la possibilità di realizzare velocemente oggetti simmetrici, lavorando solo su di una metà, duplicando e capovolgendo il tutto al termine del lavoro per ottenere l'altra parte.

Dopo aver creato il personaggio e prima di passare all'animazione vera e propria, si deve fare il **rigging**. Il termine "rig" normalmente si utilizza per definire il sistema di movimenti e di controllo delle marionette: lo stesso si fa con il nostro personaggio, ne si costruisce uno scheletro che poi deve essere mosso come una marionetta per il posizionamento di braccia e gambe

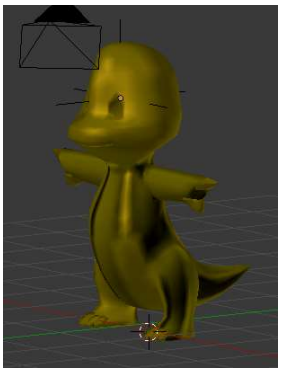
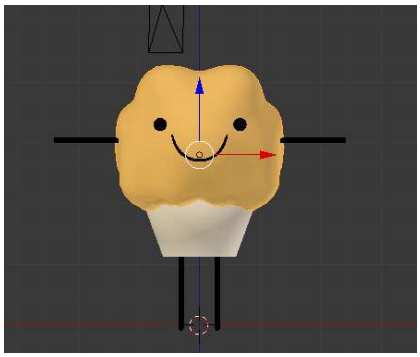
Dopo aver creato ossatura e articolazioni, si lega l'armatura al personaggio modellato.

La teoria vede l'assegnazione ai vertici della mesh di un'**influenza delle ossa** sul movimento, misurata in pesi (o **Weight**). Un vertice può essere influenzato da più ossa: quando le muoviamo, il vertice si sposterà verso quella con peso maggiore. È facile quindi comprendere come esista una grande varietà di movimenti che possiamo far compiere alla mesh solo modificando l'influenza dei pesi sui suoi vertici.

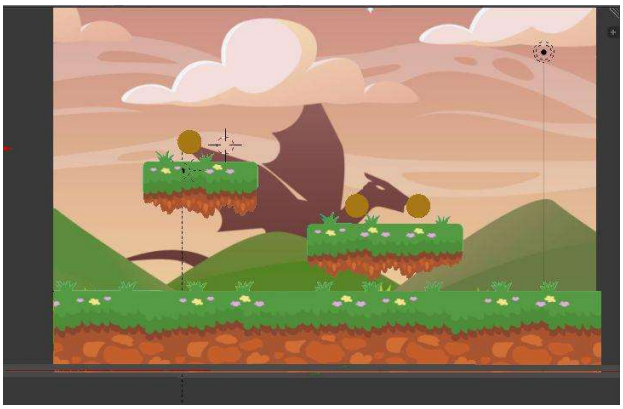
Ecco il risultato finale, con l'applicazione anche dell'animazione del salto:



L’ultimo step come si può vedere dall’immagine è la creazione delle varie animazioni dei personaggi: si può considerare animazione una sequenza di immagini proiettate ad alta velocità, capaci di rimandare l’**illusione del movimento**.
Il metodo che si usa consiste nel creare le pose principali – dette “**key frame**” (anche “keyframes”, fotogrammi chiave) – per rappresentare l’animazione in pochi disegni, collegati fra loro da pose di transizione chiamate “**inbetween**”.
Si effettua la stessa cosa per gli altri personaggi che si vogliono inserire nel gioco:

L’ultima parte è la modellazione della scena su Blender dove il gioco sarà ambientato.

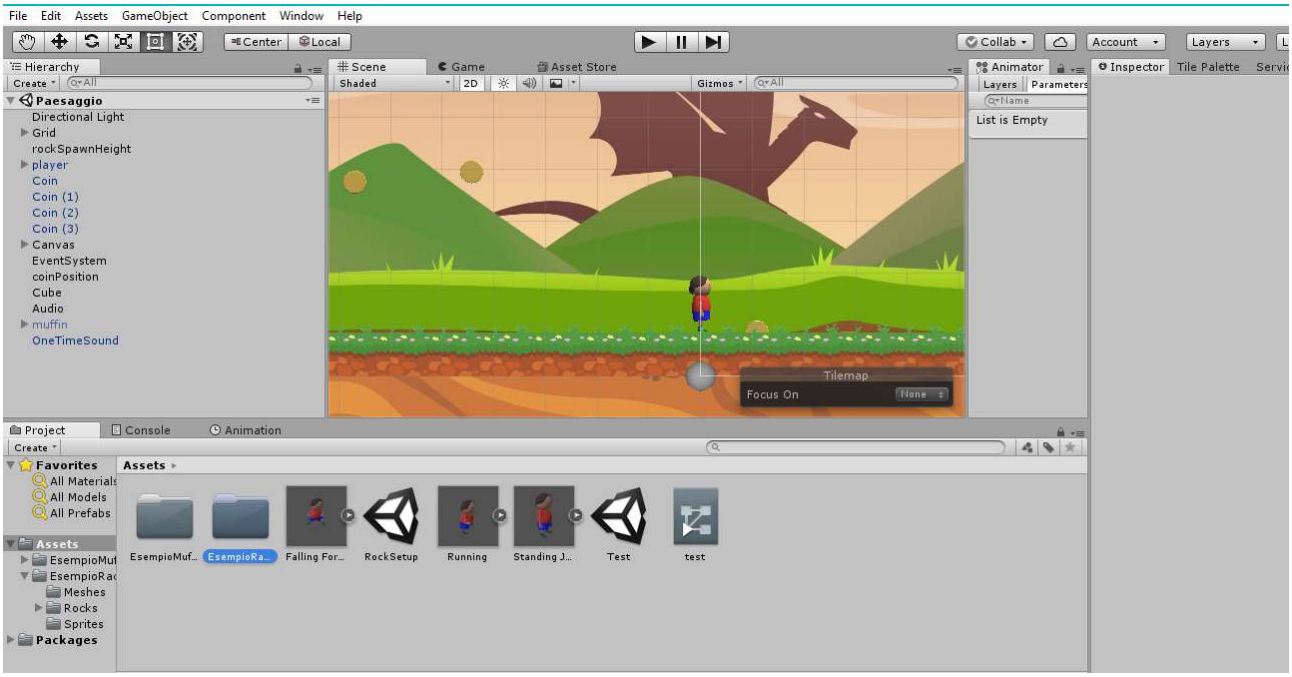


Parte 2- Utilizzo di Unity 2D

Infine bisogna importare i personaggi, la scena, e i relativi materiali,textures e shaders su Unity e iniziare a dare loro vita.

Unity è un ambiente di sviluppo creato per la creazione di videogiochi 3D o 2D o altri contenuti interattivi. Unity 3D è composto soprattutto da un motore grafico 3D, un motore fisico, componenti per l’audio, componenti per l’animazione, un IDE grafico per la gestione dei progetti e permette di scrivere i propri script utilizzando JavaScript e C#. Unity è definito multipiattaforma perché il suo motore permette di “scrivere il gioco” una sola volta e realizzarlo o trasformarlo per ambienti diversi: parliamo della creazione di uno stesso gioco per PC (Windows, Mac), Play Station, Wii, Xbox, Iphone, Ipad, cellulari con sistema Android. E’ disponibile una versione gratuita destinata ad uso privato o a piccole aziende.

Ecco la schermata una volta aperto il gioco:



Gli Asset Packages sono dei pacchetti predefiniti contenenti script, mesh, animazioni e molte altre cose che possono essere utili durante la creazione di un videogioco.

In Unity si devono poi aggiungere dei componenti della fisica per gestire i personaggi e le loro collisioni con il mondo che li circonda.

L'aggiunta di un componente RigidBody ad un oggetto vincolerà il suo movimento al controllo del motore fisico di Unity. Gli oggetti che possiedono un Colliders son statici perché anche dopo un urto non si spostano. Per il personaggio abbiamo bisogno di un RigidBody, perché questo permette di ricevere urti anche da altri collider. Collider è la classe base da cui ereditano tutti i tipi di collider, ognuno dei quali ha una forma diversa. I colliders non sempre devono corrispondere alla geometria reale di un oggetto. Io ho usato il Capsule Collider per ogni personaggio, mi è sembrato molto preciso durante le collisioni con il suolo o altri personaggi.

Per aggiungere interattività in Unity è necessario usare degli script. Gli script stanno nel progetto come gli altri Asset (texture, modelli,…) ed ogni script è rappresentato da un file apribile con Visual Studio. Unity prevede l'utilizzo di 3 linguaggi, tra cui C#, quello che ho usato. Gli script in Unity sono tutti classi che ereditano da Monobehavior, una classe identifica una categoria di oggetti che hanno tutti le stesse proprietà.

Nel nostro caso di devono scrivere script per la gestione dei vari personaggi, delle monete, dei sassi, per la generazione della scena.

Riporto qui sotto alcuni esempi di script:

La generazione della scena consiste nel disporre gli oggetti nella scena (quindi i vari personaggi, le monete,i sassi, le piattaforme) decidendone anche la frequenza con cui questi compaiono.

```
public class MapGenerator : MonoBehaviour
{
    public Transform player;
    public Tile tile;
    public int startTiles = 20;
    public float tileSize = 1;
```

```
    public GameObject coin;
    public GameObject enemy;
    public GameObject bonusLife;
    public GameObject cangrejo;
    public GameObject[] rocks;
    public Transform rockSpawnHeigh;
    Tilemap tilemap;
    private int lastTileXposition;
    public bool canGenerateRock = true;
    int generatedPlatforms = 0;
    private Coroutine stopRocksRoutine;
    List<GameObject> spawnedRocks;

    private void Awake()
    {
        tilemap = GetComponent<Tilemap>();
        lastTileXposition = (int)(startTiles * tileSize);
        spawnedRocks = new List<GameObject>();
    }

    void Update()
    {
        float distance = lastTileXposition - player.position.x;
        if (distance < 10 * tileSize)
        {
            bool spawnEnemy = Random.Range(0, 100) < 4;
            bool spawnBonusLife = Random.Range(0, 100) < 9;

            //generiamo altri tiles;
            lastTileXposition++;
            generatedPlatforms++;

            if (generatedPlatforms % Random.Range(10, 21) == 0)
            {
                //gbenera piattaforma
                int height = Random.Range(2, 6);
                for (int i = 0; i < Random.Range(5, 10); i++)
                {
                    bool spawnCoin = Random.Range(0, 2) > 0;
                    if (spawnCoin)
                    {
                        Vector3 coinPosition = new Vector3(lastTileXposition + i + 0.5f, height + 1 + 0.5f,
0);

                        Instantiate(coin, coinPosition, Quaternion.identity, null);
                    }
                    spawnEnemy = SpawnPrefab(enemy, spawnEnemy, height);
                    spawnBonusLife = SpawnPrefab(bonusLife, spawnBonusLife, height);

                    tilemap.SetTile(new Vector3Int(lastTileXposition + i, height, 0), tile);

                }

                height = Random.Range(5, 8);
                int shift = Random.Range(0, 3);

                for (int i = 0; i < Random.Range(2, 4); i++)
                {
                    bool spawnCoin = Random.Range(0, 2) > 0;
                    if (spawnCoin)
                    {
                        Vector3 coinPosition = new Vector3(lastTileXposition + i + shift + 0.5f, height + 1 +
0.5f, 0);

                        Instantiate(coin, coinPosition, Quaternion.identity, null);
                    }
                    spawnEnemy = SpawnPrefab(enemy, spawnEnemy, height);
                    spawnBonusLife = SpawnPrefab(bonusLife, spawnBonusLife, height);

                    tilemap.SetTile(new Vector3Int(lastTileXposition + i + shift, height, 0), tile);
                }
            }

            spawnEnemy = SpawnPrefab(enemy, spawnEnemy, 1);

            tilemap.SetTile(Vector3Int.right * lastTileXposition, tile);

            if (canGenerateRock)
```

```
            {
                if (Random.Range(0, 100) < 10)
                {
                    //spawn rock
                    GameObject rockInstance = Instantiate(rocks[Random.Range(0, rocks.Length)]);
                    spawnedRocks.Add(rockInstance);
                    Vector3 currentPosition = rockInstance.transform.position;
                    currentPosition.y = rockSpawnHeigh.position.y;
                    currentPosition.z = rockSpawnHeigh.position.z;
                    currentPosition.x = lastTileXposition + 0.5f;
                    rockInstance.transform.position = currentPosition;
                }
            }

            if (Random.Range(0, 100) < 1)
            {
                SpawnPrefab(cangrejo, true, 0);
            }
        }
    }

    public void StopRocksFor(float stoneGenerationPause)
    {
        List<GameObject> toRemove = new List<GameObject>();
        for (int i = 0; i < spawnedRocks.Count; i++)
        {
            if(spawnedRocks[i].GetComponentInChildren<Renderer>().isVisible)
            {
                toRemove.Add(spawnedRocks[i]);
            }
        }

        for (int i = 0; i < toRemove.Count; i++)
        {
            spawnedRocks.Remove(toRemove[i]);
            Destroy(toRemove[i]);
        }

        if (stopRocksRoutine != null)
        {
            StopCoroutine(stopRocksRoutine);
        }
        stopRocksRoutine = StartCoroutine(StopRocksRoutine(stoneGenerationPause));
    }

    private IEnumerator StopRocksRoutine(float stoneGenerationPause)
    {
        canGenerateRock = false;
        yield return new WaitForSeconds(stoneGenerationPause);
        canGenerateRock = true;
        stopRocksRoutine = null;
    }

    private bool SpawnPrefab(GameObject prefab, bool needToSpawn, int height)
    {
        if (needToSpawn)
        {
            Vector3 position = new Vector3(lastTileXposition + 0.5f, height + 0.5f, 0);
            Instantiate(prefab, position, Quaternion.identity, null);
            needToSpawn = false;
        }

        return needToSpawn;
    }

    private void OnDrawGizmos()
    {
        Gizmos.color = Color.white;
        Gizmos.DrawSphere(Vector3.right * lastTileXposition, 0.3f);
    }
}
```

Lo script player gestisce punti (determinati dalla quantità di monete) e vite del giocatore e vittorie e sconfitte

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

public class Player : MonoBehaviour
{
    public int coinScore = 1;

    [Header("UI")]
    public Text labelCoins;
    public Text labelScore;
    public Text labelLives;
    public GameObject panelLose;

    private int numberOfCoins = 0;
    private int totalScore = 0;
    private int lives = 0;
    private Animator animator;
    private Coroutine deathCoroutine;

    private void Awake()
    {
        animator = GetComponentInChildren<Animator>();
    }
    private void Start()
    {
        AddLives(1);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Coin"))
        {
            numberOfCoins++;
            AddScore(coinScore);
            UpdateLabelCoins();
            Destroy(collision.gameObject);
        }
    }

    private void OnCollisionEnter2D(Collision2D collision)
    {
        if (collision.gameObject.CompareTag("Enemy"))
        {
            TakeDamage(collision.gameObject);
        }
        else if (collision.gameObject.CompareTag("Muffin"))
        {
            Destroy(collision.gameObject);
            AddLives(1);
        }
    }

    private void AddLives(int quantity)
    {
        lives += quantity;
        if (lives < 0)
        {
            lives = 0;
        }
        labelLives.text = "Lives: " + lives;
    }

    public void TakeDamage(GameObject damager)
    {
        numberOfCoins -= 5;

        if (numberOfCoins < 0)
        {
```

```
            AddLives(-1);
            if (lives < 1 && deathCoroutine == null)
            {
                deathCoroutine = StartCoroutine(Death());
            }
            else
            {
                numberOfCoins = 0;
            }
        }

        if (numberOfCoins > 60)
        {
            numberOfCoins -= 10;
        }


        UpdateLabelCoins();
        Destroy(damager);
    }


    public void Die()
    {
        AddLives(-1);
        if (lives < 1 && deathCoroutine == null)
        {
            deathCoroutine = StartCoroutine(Death());
        }
    }

    private IEnumerator Death()
    {
        animator.SetTrigger("die");
        labelScore.text = "Score: " + totalScore;
        yield return new WaitForSeconds(2.5f);
        panelLose.SetActive(true);
        Time.timeScale = 0;
    }

    private void UpdateLabelCoins()
    {
        labelCoins.text = "Coins: " + numberOfCoins;
    }

    public void AddScore(int quantity)
    {
        totalScore += quantity;
    }
}
```

Si devono importare anche le animazioni dei vai personaggi fatte con blender su unity e gestirle attraverso una macchina a stati.

Infine si inseriscono dei suoni all'interno del gioco durante le collisioni, e una musica di sottofondo.

Alcuni screenshot e spiegazione del gioco

Lo scopo del gioco è di raccogliere quante più monete possibili, saltando da una piattaforma ad un'altra, cercando di raccogliere uno score elevato. Il personaggio dovrà evitare i sassi (altrimenti morirà se non avrà vite) e non lasciarsi prendere dal nemico che gli toglie ogni volta 5 monete (se le monete che ha diventano minori di zero il personaggio morirà).

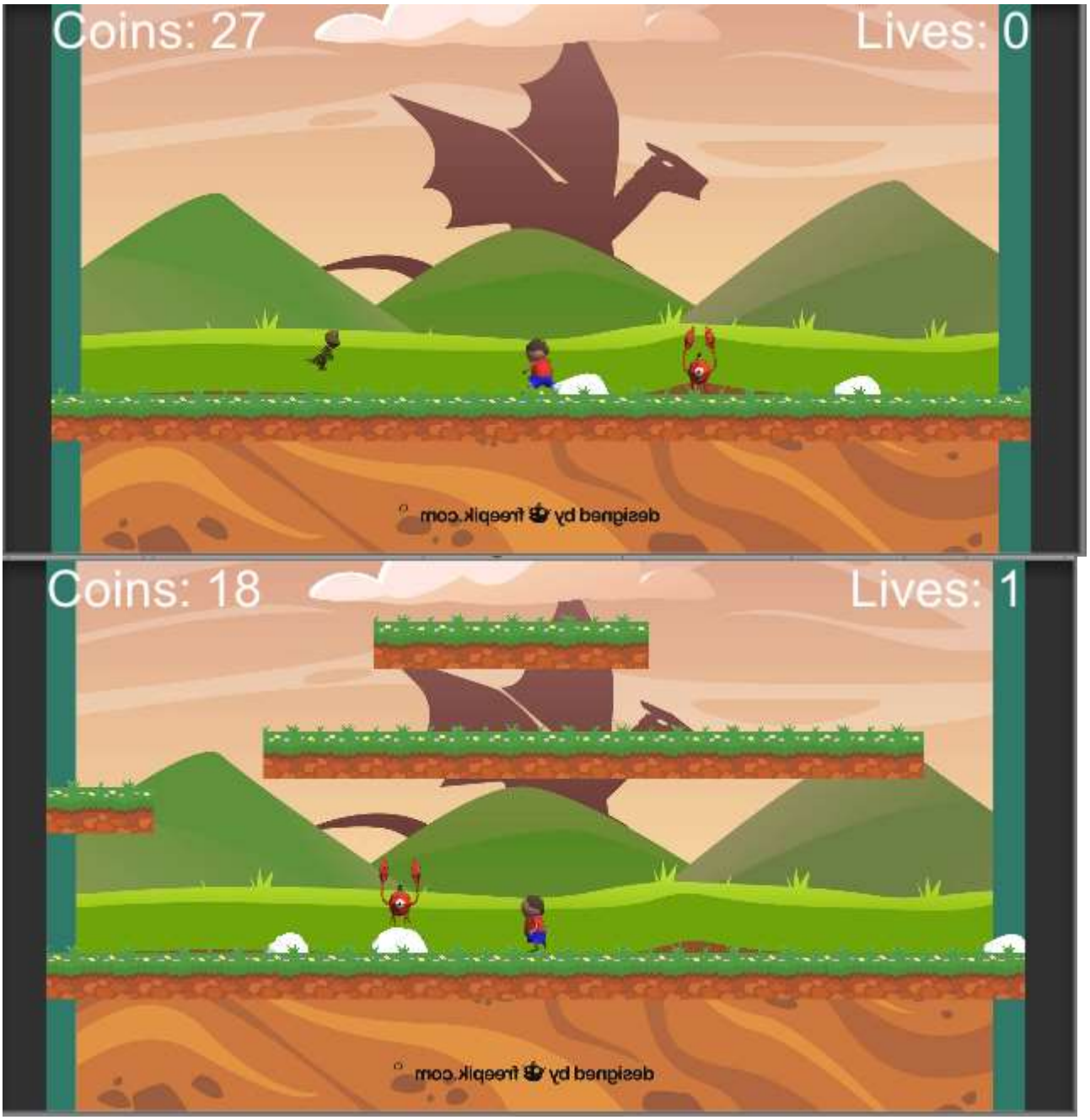
Il nostro personaggio avrà due aiutanti: il muffin e il granchio. A volte può capitare di vedere un muffin camminare per una piattaforma. Se il nostro protagonista riuscirà a prenderlo, allora potrà avere una vita in più.

Se riuscirà a prendere il granchio, i sassi spariranno sulla scena per 10 secondi.

Alcuni screenshot del gioco concluso:

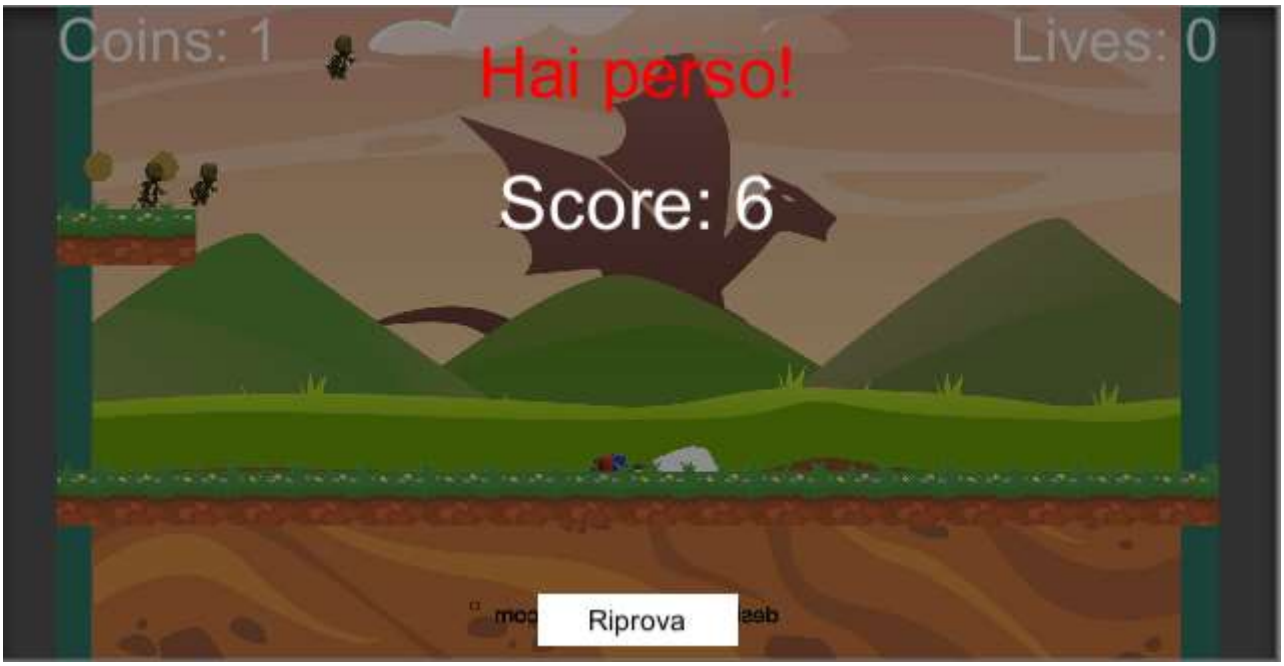
Il nostro protagonista con i nemici, le monete e i sassi:

Il protagonista con il nemico e il granchio:



Il protagonista, i nemici, e il countdown di 10 secondi (dopo aver preso un granchio):

Il muffin e il protagonista:



Ogni volta che si perde si potrà riprovare il gioco da capo.

Mie conclusioni finali:

Il lavoro necessario per creare tutto quello serve per la realizzazione di un gioco è lungo ed impegnativo. Una buona parte del lavoro consiste nella precisione nel creare l'effetto desiderato nella modellazione degli oggetti e nell'applicazione di textures o materiali. Spesso ci vuole un lavoro di precisione e anche molto ripetitivo per arrivare al risultato finale che avevamo desiderato. E' fondamentale anche una buona conoscenza della computer Grafica per la realizzazione di un videogioco, tra cui argomenti quali il 3D Modeling, la programmazione delle logiche di gioco e l'audio. Per avere dei buoni risultati finali, bisogna avere molta esperienza nella pratica della modellazione e della logica di gioco, bisogna quindi essere esperti e aver fatto tanti altri progetti. Molto importante all'inizio è farsi un'idea di quello che si vuole realizzare.

Personalmente questa è stata la mia prima esperienza nella realizzazione di un videogioco. Ho realizzato un lavoro semplice, 2D, ma sotto ho dovuto lavorare tanto, anche per iniziare a capire da principiante come utilizzare le due piattaforme e rendermi amica di Blender e Unity. A tratti mi è parso un lavoro macchinoso e un po' snervante, ma la mia esperienza nel complesso è stata positiva. E' soddisfacente vedere alla fine che tutto quello che hai creato prende vita.

Zandoli Silvia