A.A. 2017/2018

Progetto di Sistemi di Realtà Virtuale

Raffaele Raggi

For Carrots: RUN!!

ForCarrots:RUN!! è un gioco in stile “endless runner”. Il giocatore vestirà i panni di un coniglietto nero che correndo su di una strada infinita deve raccogliere carote fluttuanti e soprattutto evitare gli ostacoli che troverà lungo il percorso, per rimanere in vita e accumulando in questo modo dei punti.

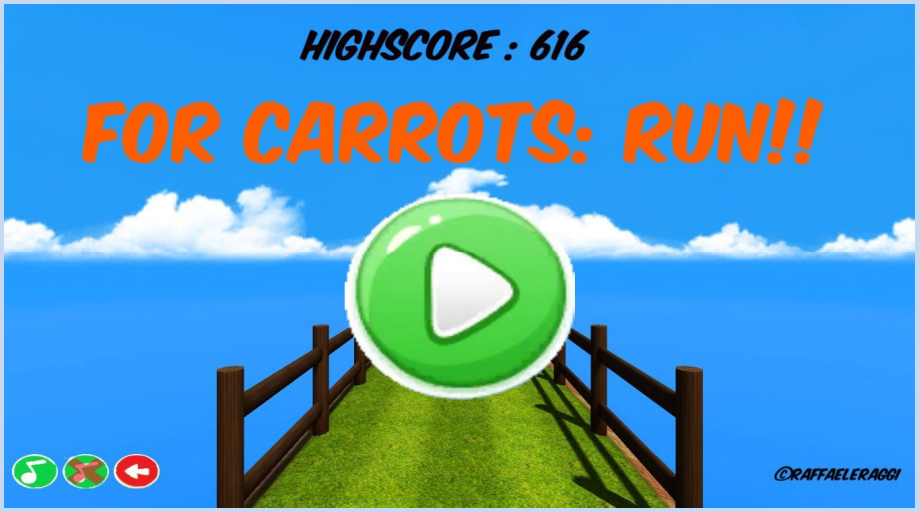
Software utilizzato

- **Blender** - rigging e animazione del player
- **Unity 3D** - struttura e logica del progetto
- **GIMP** – modifiche di alcuni sprites presenti negli assets

I menu di gioco

Oltre la scena di gioco, che vedremo poi, il gioco è composto da tre menu: MainMenu, PauseMenu e DeathMenu.

Nel **MainMenu** troviamo un grande pulsante centrale per avviare la scena di gioco e altri tre bottoni più piccoli che hanno il compito di togliere o restituire l'audio al gioco e di chiudere l'applicazione. Oltre ovviamente a contenere il titolo e il contatore per l' *highscore*.



Il **PauseMenu** è accessibile tramite il PauseButton presente nella scena di gioco principale.

La pressione sul bottone attiva il relativo canvas sul quale troviamo diversi pulsanti: Attiva/Disattiva suono, Quit, Restart, Menu (riporta al MainMenu) e il tasto Resume che disabilita il canvas relativo al PauseMenu riportando in attivo la scena di gioco.

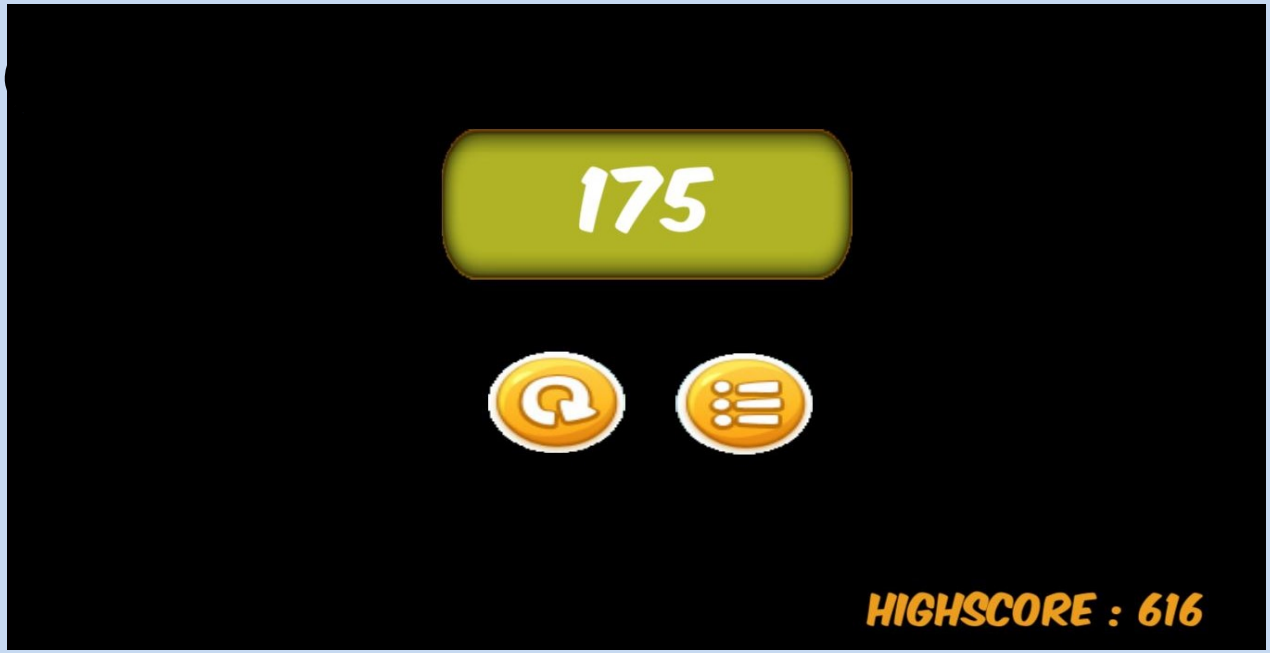

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PauseButton : MonoBehaviour {

  public GameObject pauseButton, pauseMenu;

  public void Start()
  {
    OnUnPause();
  }

  public void OnPause()
  {
    pauseMenu.SetActive(true);
    pauseButton.SetActive(false);
    Time.timeScale = 0;
  }

  public void OnUnPause()
  {
    pauseMenu.SetActive(false);
    pauseButton.SetActive(true);
    Time.timeScale = 1;
  }

}
```

Come intuibile si accede al **DeathMenu** quando il Player muore, perdendo la partita.
In questo canvas abbiamo Il contatore di highscore:

```
highscoreText.text = "Highscore : " + ((int)PlayerPrefs.GetFloat("Highscore")).ToString();
```

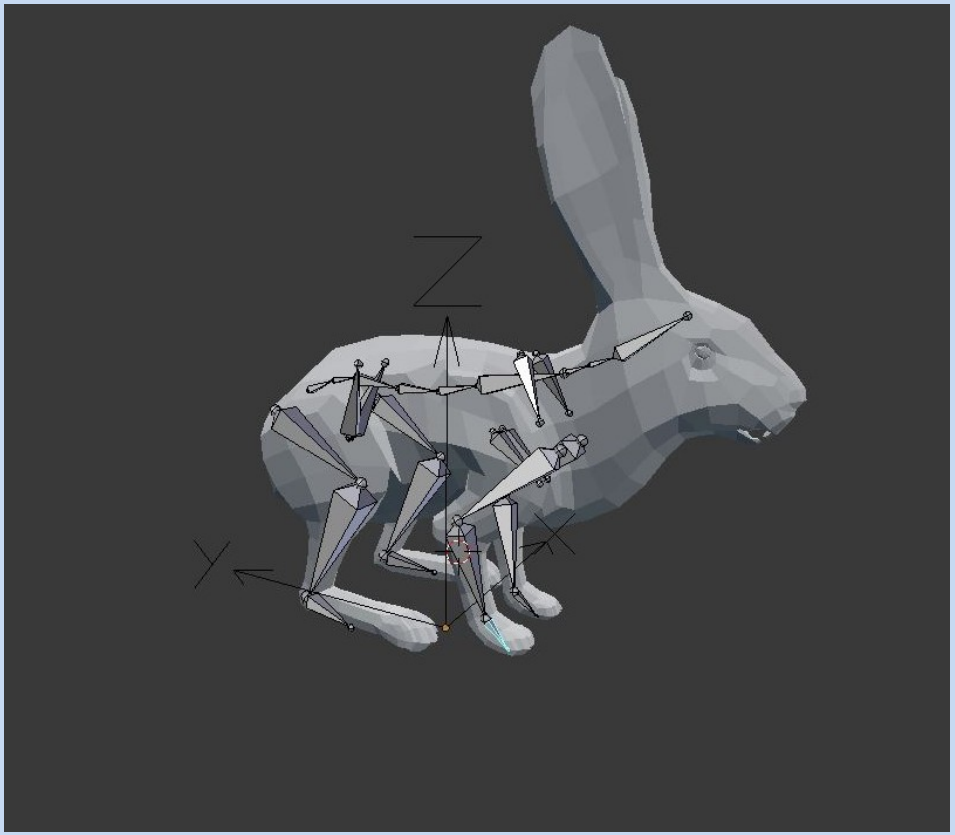
Inoltre sono presenti i bottoni di Restart e Menu, che come nel **PauseMenu** lavorano caricando le rispettive scene:

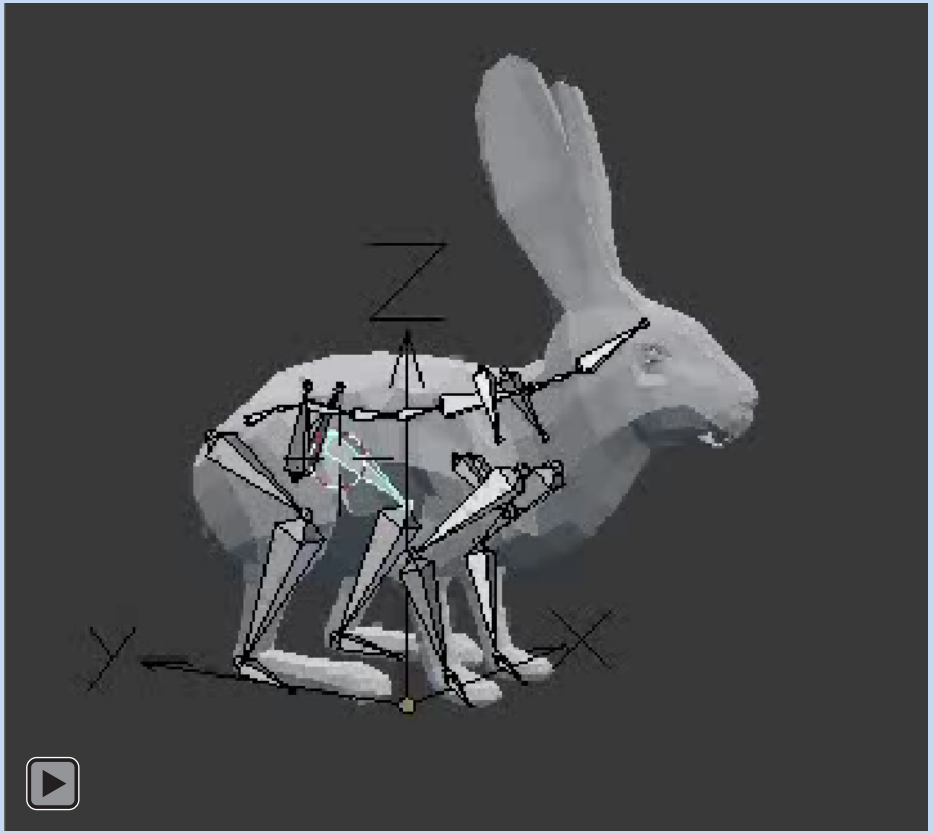
```
public void Restart()
{
  //SceneManager.LoadScene(SceneManager.GetActiveScene().name);
  SceneManager.LoadScene("Game"); //più pratico
}

public void BackToMenu()
{
   SceneManager.LoadScene("Menu");
}
```



Player

Per quanto riguarda il Player ho scaricato un avatar in 3D model (.obj) e tramite il software di modellazione 3D Blender ho eseguito per prima cosa il *rigging* del modello e poi successivamente ho generato l'animazione da importare poi nel Game Engine.





Player Controller

Lo Script *PlayerController* ha una funziona molto importante, infatti è grazie a lui che il player non solo ha una velocità e una gravità, ma sa di essere morto quando colpisce un ostacolo, e cosa ancora più importante può essere controllato nei movimenti lungo l'asse X e l'asse Y. Oltre i comandi con input da tastiera ho deciso anche di inserire quelli con input da click che possono essere usati con un mouse o con un touchscreen, così da poter *"buildare"* il progetto anche per dispositivi mobile.

```
moveVector = Vector3.zero;

if(controller.isGrounded)
{
  verticalVelocity = -0.5f;
}else{
  verticalVelocity-= gravity * Time.deltaTime;
}

// X - Left and Right
moveVector.x = Input.GetAxisRaw("Horizontal") * speed;

if (Input.GetMouseButton(0))
if(Input.mousePosition.y < Screen.height /2){

{
  if (Input.mousePosition.x > Screen.width /2)
    moveVector.x = speed;
    else
    moveVector.x = -speed;
}



// Y - Up and Down
moveVector.y = verticalVelocity;

if(Input.GetKeyDown("space"))

  controller.transform.Translate(Vector3.up * 2.2f);

  if(controller.isGrounded)
  {
  if (Input.GetMouseButton(0))
  if(Input.mousePosition.y > Screen.height /2){

  {
    if (Input.mousePosition.x < Screen.width /2)
      controller.transform.Translate(Vector3.up * 2.2f);
      else
      controller.transform.Translate(Vector3.up * 2.2f);
  }
}
```

I comandi "touch" funzionano dividendo idealmente lo schermo in 3 quadranti: due quadranti inferiori che consento di andare a destra e sinistra, ed un pulsante superiore che consente di saltare.

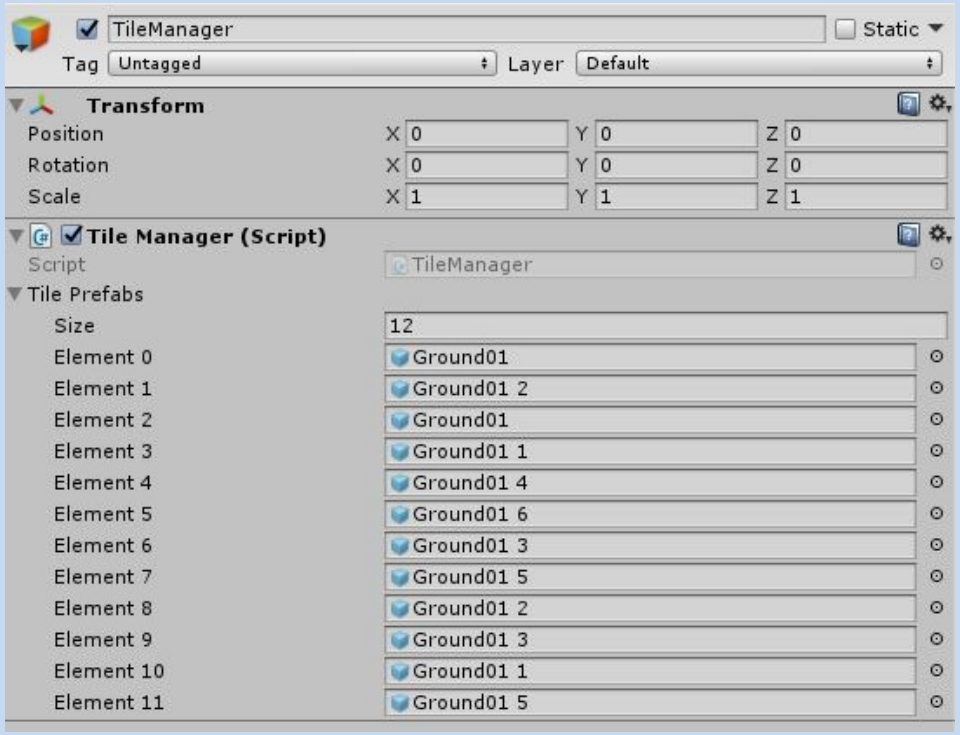
Terreno di gioco

Dato che per le mie esigenze serviva un terreno che riuscisse a trasmettere l'idea di infinito, invece di usare un *terrain* ho deciso di generare delle *Tiles Prefabs* e affiancarle allo script *TileManager.*

Il *TileManager* si occupa di richiamare il numero dei *GameObject* da usare (nel mio caso 11), successivamente decide quanto ogni Tile deve essere lunga e quante *Tiles* devono comparire sullo schermo.
Successivamente lo script definisce una *SafeZone*, tale limite è il trigger che, al passaggio del player da il via allo "*spawn*" di una nuova *tile*. Per ogni nuova *tile* che viene generata, la *tile* più vecchia viene rimossa.

```
public class TileManager : MonoBehaviour {

  public GameObject [] tilePrefabs;

  private Transform playerTransform;
  private float spawnZ = -6.0f;
  private float tileLength = 20.0f;
  private int amnTilesOnScreen = 12;
  private float safeZone = 15.0f;
  private int lastPrefabIndex = 0;

  private List<GameObject> activeTiles;

  // Use this for initialization
  private void Start () {
    activeTiles = new List <GameObject> ();
    playerTransform = GameObject.FindGameObjectWithTag ("Player").transform;
    for (int i = 0; i < amnTilesOnScreen; i++) {

      if (i < 2)
        SpawnTile (0);
      else
        SpawnTile ();
    }
  }

  // Update is called once per frame
  private void Update () {
    if (playerTransform.position.z - safeZone > (spawnZ - amnTilesOnScreen * tileLength)) {
      SpawnTile ();
      DeleteTile ();
    }
  }
```

Lo *spawn* delle *tiles* viene viene gestito instanziando come *GameObject* le *Tileprefabs* per un *PrefabIndex*.
A sua volta il *PrefabIndex* viene istanziato come *randomindex,* ovvero un indice che sceglie casualmente la *tile* da *"spawnnare"* volta per volta, scegliendo dalla posizione 0 fino alla lunghezza massima del *Tileprefabs.*



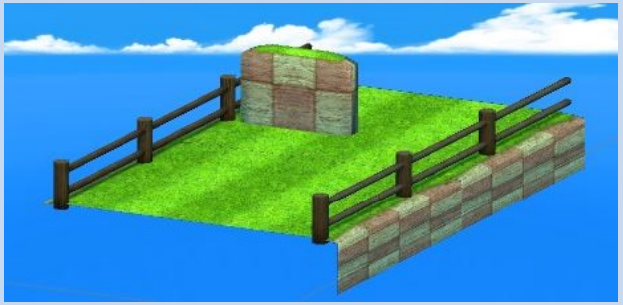

```
private void SpawnTile(int prefabIndex = -1){
  GameObject go;
  if (prefabIndex == -1)
    go = Instantiate (tilePrefabs [RandomPrefabIndex ()]) as GameObject;
  else
    go = Instantiate (tilePrefabs [prefabIndex]) as GameObject;
  go.transform.SetParent (transform);
  go.transform.position = Vector3.forward * spawnZ;
  spawnZ += tileLength;
  activeTiles.Add (go);
}

private void DeleteTile(){
  Destroy (activeTiles [0]);
  activeTiles.RemoveAt (0);
}

private int RandomPrefabIndex (){
  if (tilePrefabs.Length <= 1)
    return 0;

  int randomindex = lastPrefabIndex;
  while (randomindex == lastPrefabIndex) {
    randomindex = Random.Range (0, tilePrefabs.Length);
  }

  lastPrefabIndex = randomindex;
  return randomindex;
}
```

Gli ostacoli e i collezionabili

Durante il gioco il Player si imbatte in due tipi di oggetti: Gli ostacoli e i collezionabili (carote).
Gli ostacoli sono inseriti nelle *TilesPrefab*, ognuna delle quali ha un ostacolo diverso in una diversa posizione. Vengono quindi generati insieme alle *Tiles* nel modo descritto in precedenza.
I collezionabili invece sono generati dallo Script *RandomSpawnScript*. Lo script non fa altro che generare con un intervallo di tempo i *GameObject* che gli vengono assegnati, scegliendo casualmente quale *GameObject* usare e dove posizionarlo.

Per quanto riguarda gli ostacoli sono contrassegnati dal *tag "Enemy"* e il loro funzionamento è descritto in queste stringhe:

```
private void OnControllerColliderHit(ControllerColliderHit hit)
{
  if(hit.gameObject.tag == "Enemy")
     Death();
}
```



Invece le carote hanno un *BoxCollider* con un *Trigger*, il che consente di simulare la raccolta dell'oggetto tramite questo script:

```
public class PickUpItem : MonoBehaviour {

  void OnTriggerEnter(Collider other){

        Destroy(other.gameObject);
```

Il punteggio

Il punteggio viene accumulato in due modi, entrambi regolati nello script *Score:*
correndo e quindi rimanendo in vita per più tempo possibile, e raccogliendo gli oggetti collezionabili.

Il gioco è idealmente diviso in 10 livelli di difficoltà. Il primo livello finisce a 10 punti, il secondo a 20 punti, il terzo a 40 e così via raddoppiando ogni volta il valore da raggiungere per salire di livello. Ogni volta che si sale di livello il player aumenta la propria velocità.

Per quanto riguarda i collezionabili sono divisi in due tipi, uno dal valore di 5 punti e l'altro da 10 punti.

Tramite questo script vengono assegnati i punti ogni volta che si attiva il *trigger* dei collezionabili.

```
void OnTriggerEnter(Collider other){
  if (other.gameObject.tag == "Big"){
    score+= 10;
    scoreText.text=((int)score).ToString ();

  }
  else{
  score+= 5;
  scoreText.text=((int)score).ToString ();
  }
}
```

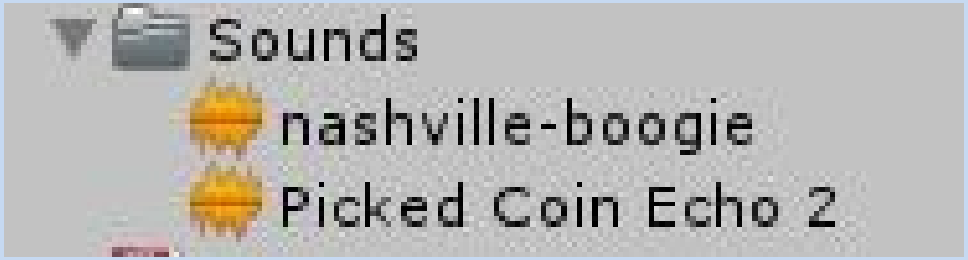
Camera Motor

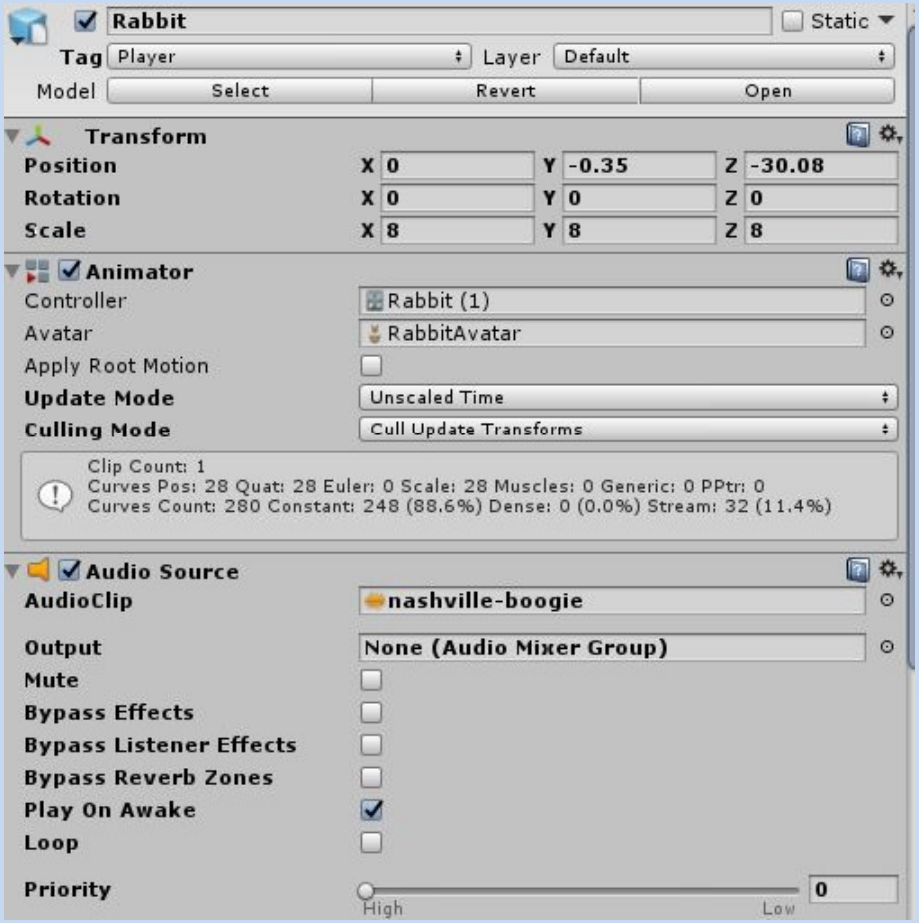
Il **CameraMotor** è un oggetto non visibile durante l'esecuzione del gioco, ma molto importante.
Infatti si occupa di fare in modo che la camera segua il player mantenendo sempre la stessa distanza con quest'ultimo.
In questo modo il giocatore può avere lo stesso punto di vista del player (coniglio).

```
public class CameraMotor : MonoBehaviour {

  private Transform lookAt;
  private Vector3 startOffset;
  private Vector3 moveVector;

  // Use this for initialization
  void Start () {
    lookAt = GameObject.FindGameObjectWithTag ("Player").transform;
    startOffset = transform.position - lookAt.position;
  }

  // Update is called once per frame
  void Update () {
    moveVector = lookAt.position + startOffset;

    //X
    moveVector.x = 0;

    //Y
    moveVector.y = Mathf.Clamp(moveVector.y,3,5);

    transform.position = lookAt.position + startOffset;
  }
}
```

Audio

In tutto il progetto sono presenti solamente due tracce audio.
Una come sottofondo musicale per il gioco e uno collegato agli oggetti collezionabili che si attiva ogni volta che un oggetto viene “triggerato”.





Assets

Per la realizzazione del progetto ho usato alcuni Assets non creati da me ma scaricati gratuitamente dall'apposito store di Unity 3D: Asset Store.
Per correttezza indico i nomi di tali assets:

•**GrassRoadRace** (per la strada e gli ostacoli)

•**Ultimate GUI Kit** (per i vari bottoni presenti nel gioco, fatta eccezione per 3 pulsanti che ho editato in GIMP)