

Università degli Studi di Perugia



Sistemi di realtà virtuale

Anno accademico 2019/2020

Eraldo Marku – 300749

INTRODUZIONE

L'obiettivo del progetto era quello di realizzare un videogioco. Nel progetto è stata scelta la tematica delle corse automobilistiche. Il videogioco presenta un menu iniziale dove è possibile scegliere se giocare direttamente oppure uscire dal gioco. Il gioco prevede una gara su pista contro un'altra auto dove chi arriva primo vince la gara. Il videogioco implementa anche il conteggio del tempo del giro e salva anche il miglior tempo per la mappa dove si gareggia. Quindi gli obiettivi del gioco possono essere due. Uno è come accennato prima arrivare primi e il secondo potrebbe essere anche battere il proprio tempo di giro su pista.



TECNOLOGIE UTILIZZATE

Unity: è motore grafico multipiattaforma sviluppato da Unity Technologies che consente lo sviluppo di videogiochi e altri contenuti interattivi, quali visualizzazioni architettoniche e animazioni 3d in tempo reale.



Blender: è un software libero e multipiattaforma di modellazione, rigging, animazione , montaggio video, composizione e rendering di immagini tridimensionali e bidimensionali



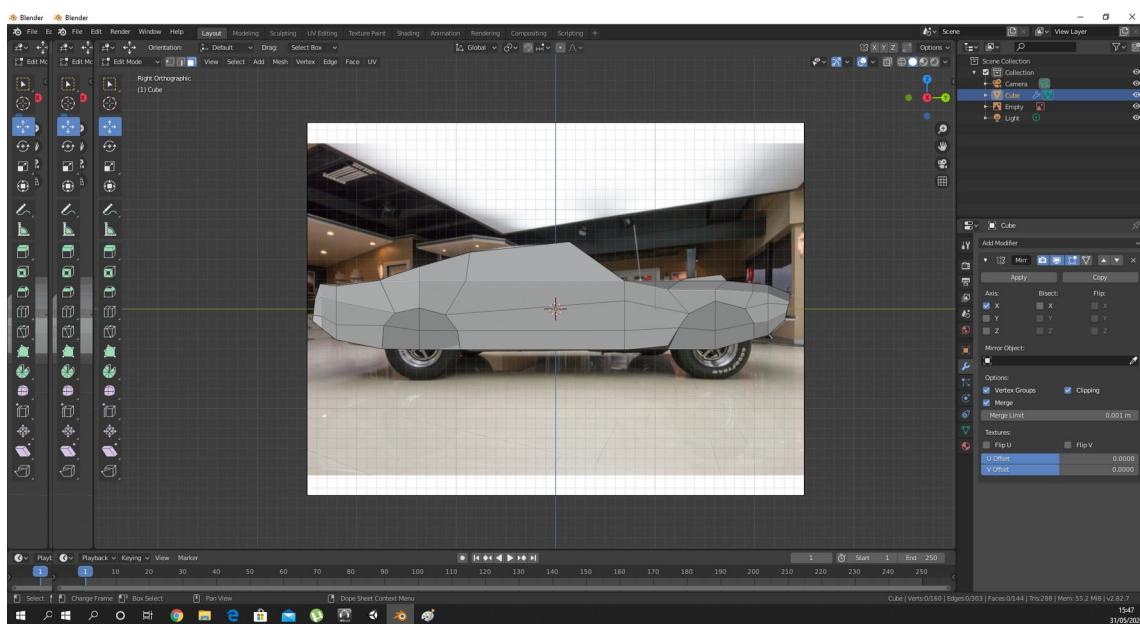
BLENDER

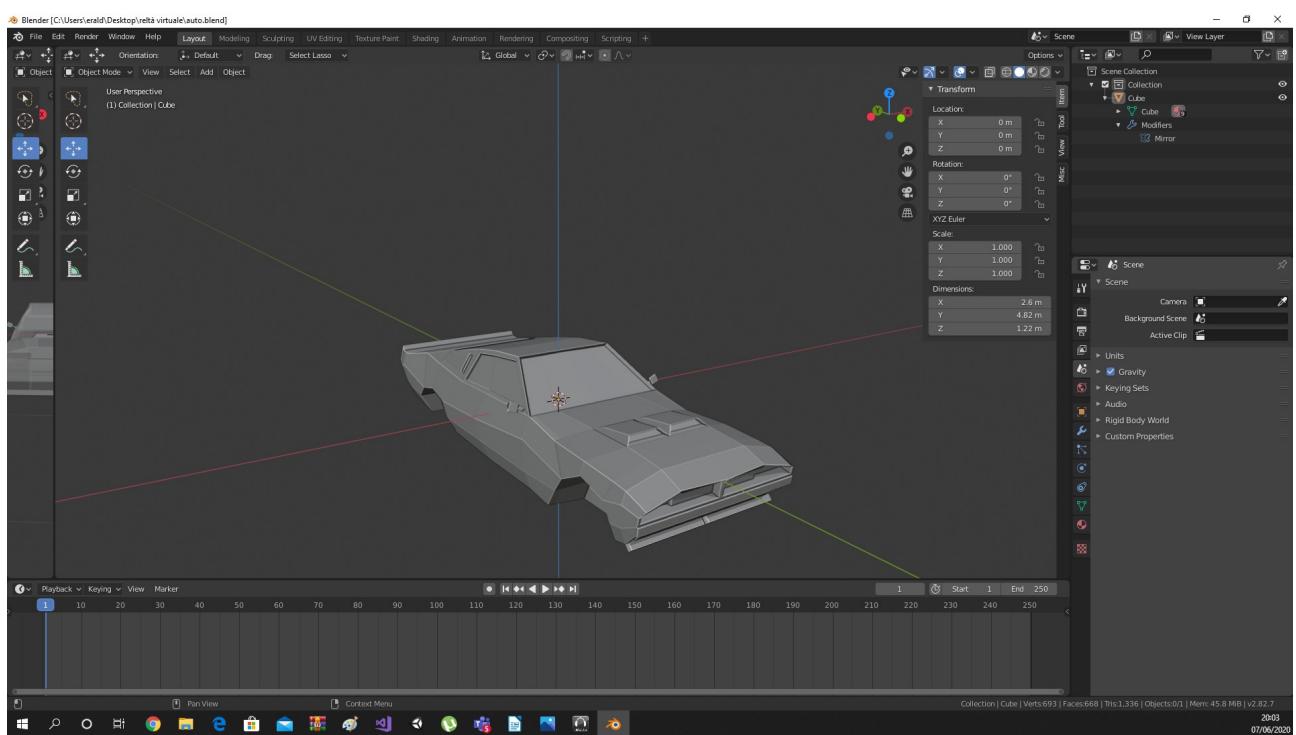
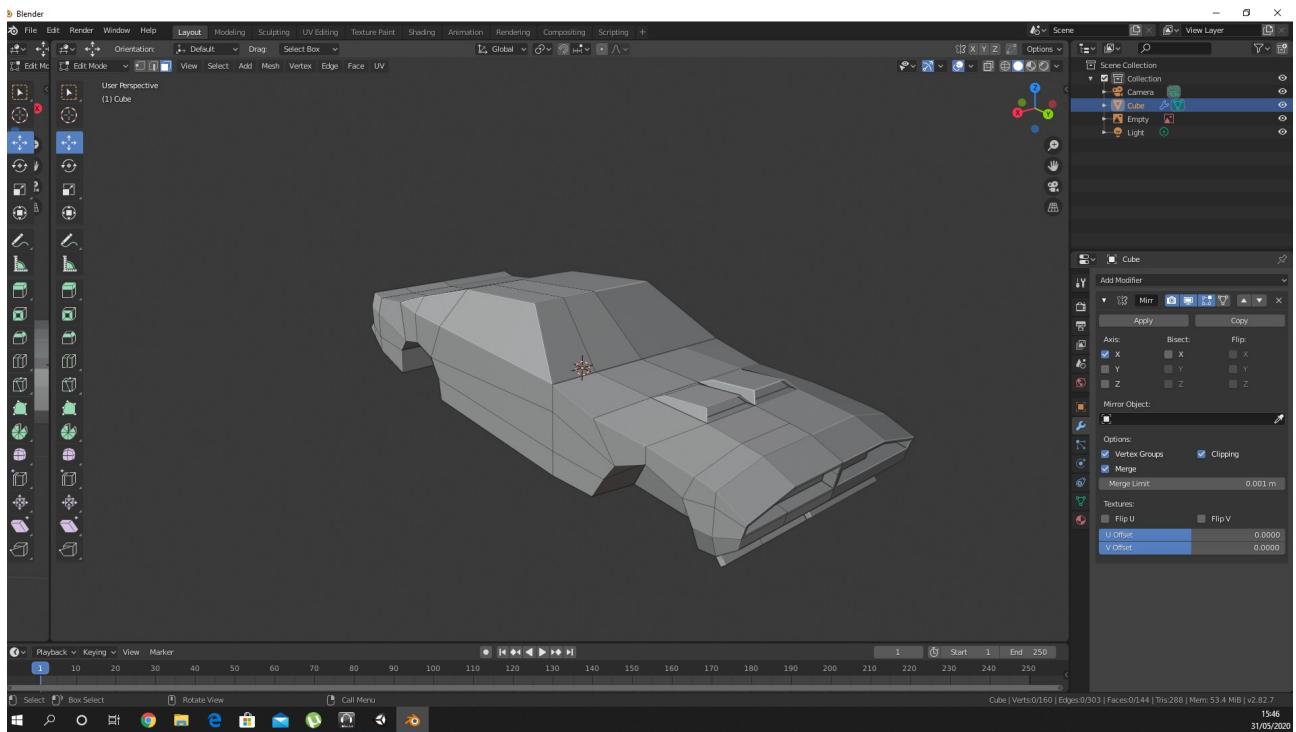
Nel seguente progetto è stato utilizzato Blender per la modellazione di 3 oggetti: l'auto, le gomme dell'auto e gli alberi.

Auto: L'auto è stata realizzata prendendo come esempio il seguente modello di una Shelby GT-500

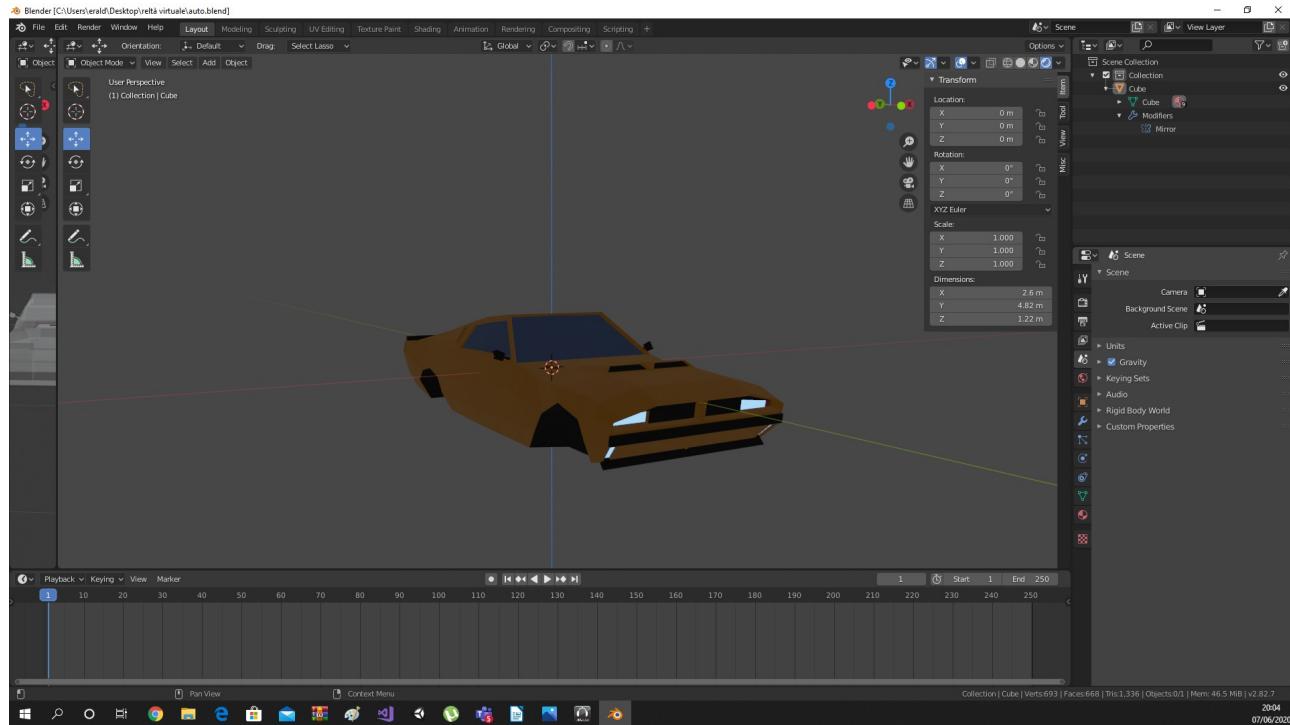


La seguente immagine è stata inserita come background image dentro blender e partendo da un cubo si è cercato di imitarne le forme lateralmente e poi con le dovute trasformazioni del cubo si è arrivati al seguente risultato

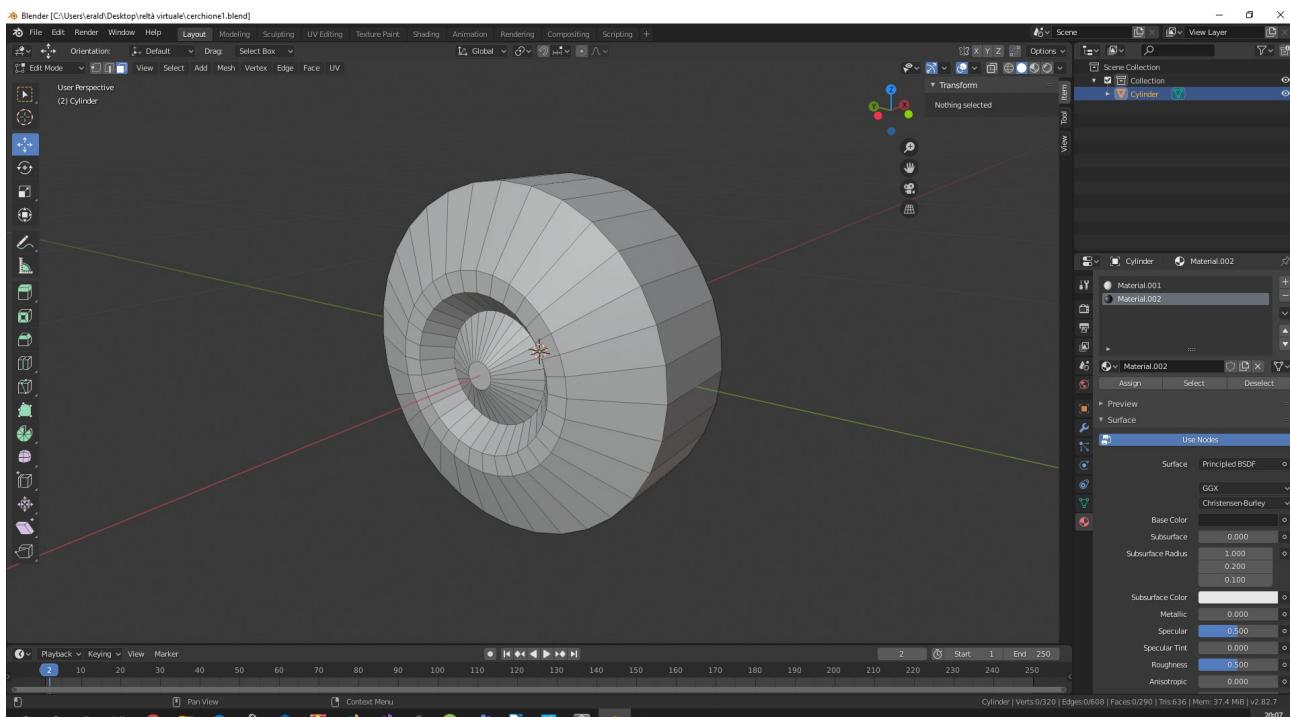




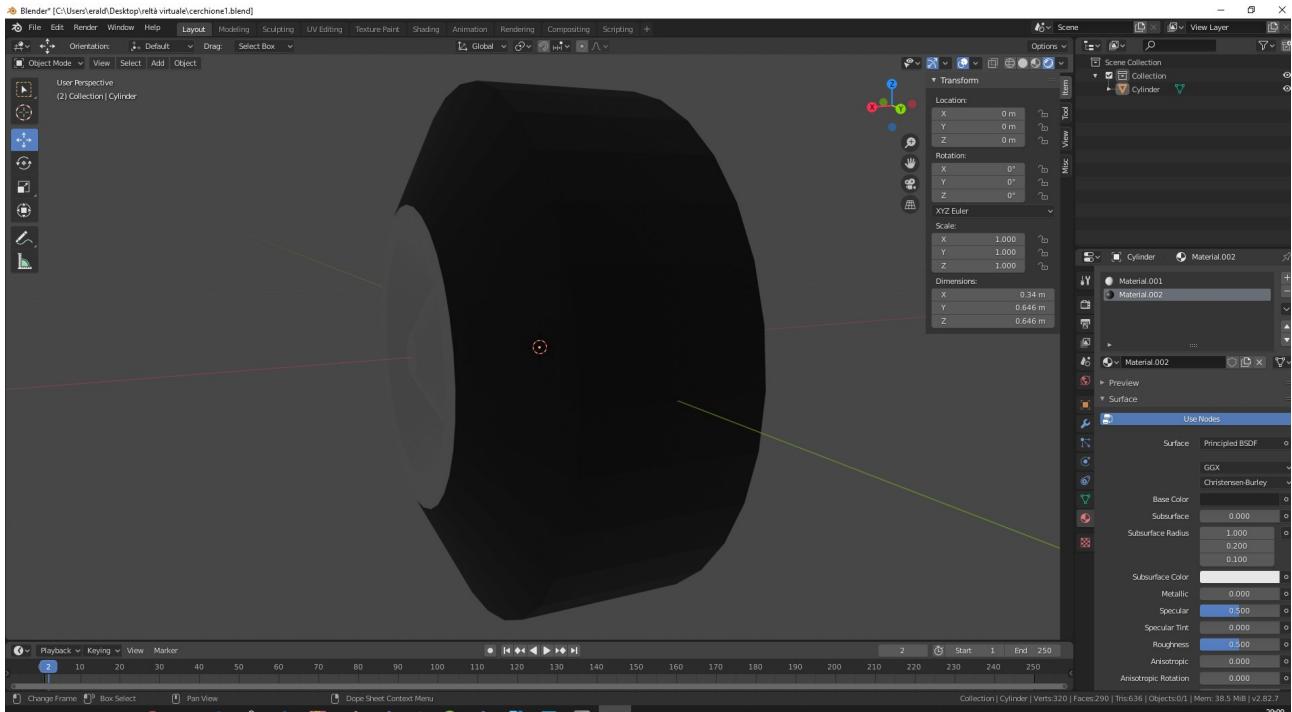
Alla fine sono stati aggiunti i materiali e come risultato finale



Ruote: la ruota è stata realizzata partendo da un cilindro che con le opportune operazioni di modellazione si è arrivati al seguente risultato



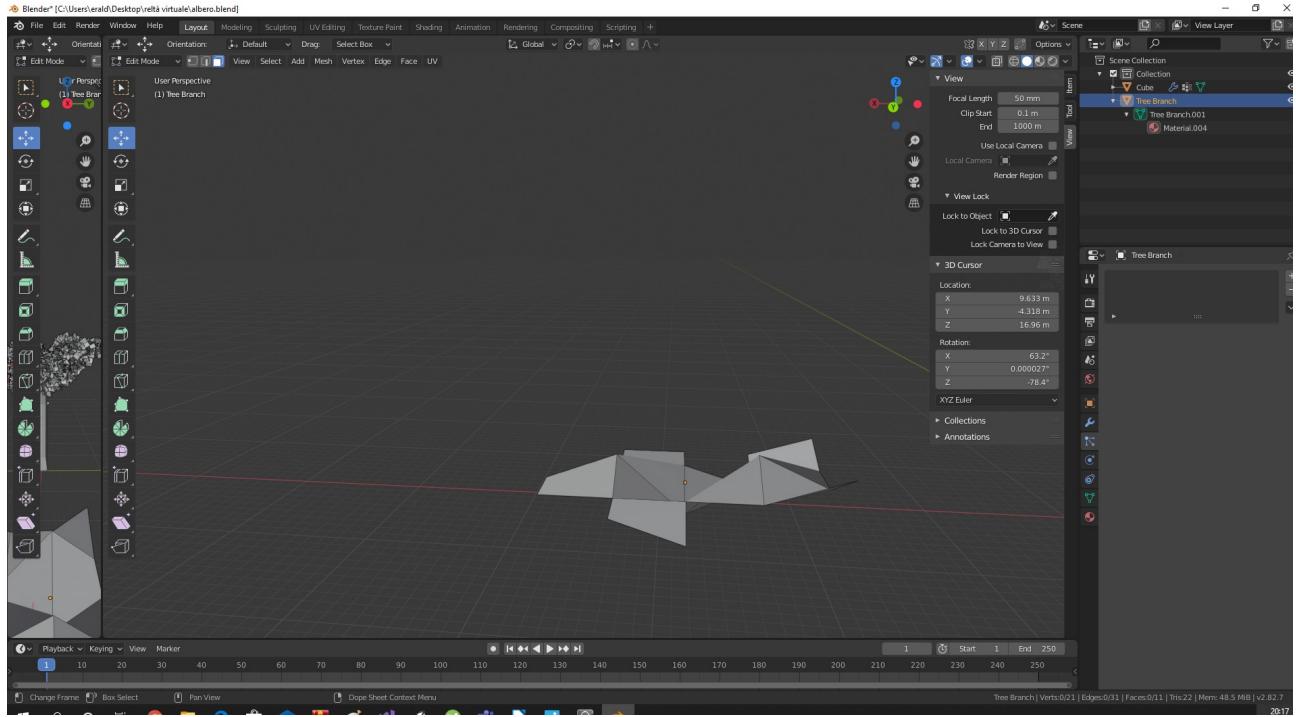
Alla fine sono stati aggiunti gli opportuni materiali per arrivare al risultato finale



Albero: la base dell'albero è stata realizzata partendo da un cubo opportunamente tagliato e allungato tramite il comando “extrude” per arrivare al seguente risultato



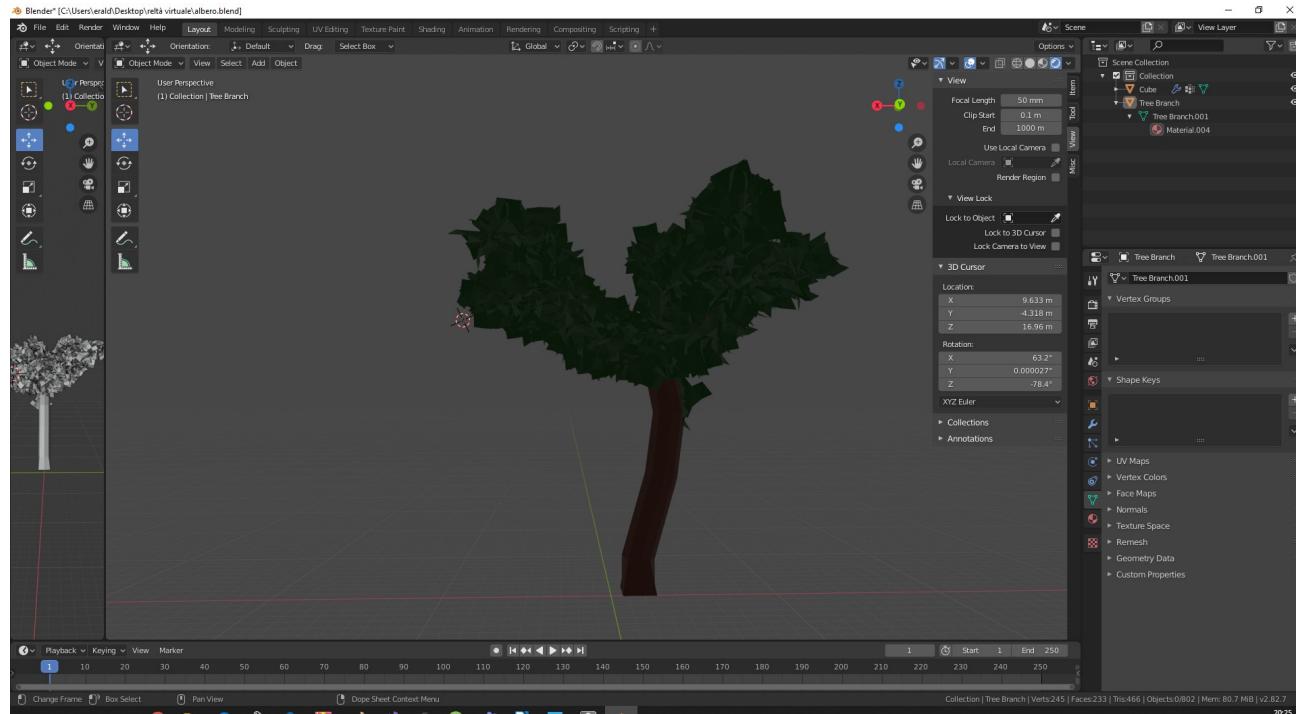
Successivamente per realizzare le foglie si è partiti dal modellare il seguente oggetto



Dopodichè alla base dell'albero è stato aggiunto un sistema di particelle con l'oggetto per le foglie utilizzato come instanza dell'oggetto nel sistema di particelle. Todo ciò è stato applicato al nuovo gruppo di vertici creato che fanno riferimento ai rami superiori dell'albero per arrivare a questo risultato



Aggiungendo i materiali opportuni si è arrivati al risultato finale



UNITY

E' stato utilizzato UNITY principalmente per lo scripting dell'applicazione ma anche per la creazione del tracciato e dell'ambiente intorno. In UNITY non è stato utilizzato nessun asset esterno quindi è stato realizzato tutto da zero.

Terreno_circuito: è stato creato un oggetto “Terrain” per la creazione del circuito che è stato opportunamente modellato per dare la forma deisderata tramite i comandi implementati in UNITY per arrivare al seguente risultato



texture



Terreno_montagne: è stato creato un’altro oggetto “Terrain” per modellare le montagne intorno al tracciato arrivando al seguente risultato



texture



Cielo: è stato realizzato partendo da un’immagine .hdr a 360° presa nel web. Successivamente dentro UNITY è stato creato un nuovo materiale dove è stato impostato come shader l’impostazione “skybox/cubemap” dentro al quale è stato inserita l’immagine hdr precedentemente scaricata e così è stato realizzato lo skybox arrivando al seguente risultato



alberi/muschi: sono stati importati dal modello albero creato con BLENDER.



oggetto_partenza: è stato realizzato tramite l'uso di due cilindri e un cubo e sono stati applicati gli opportuni materiali appositamente creati



Auto: è stato importato dentro gli assets il modello creato con Blender dell’auto e anche la ruota. Successivamente è stato creato un prefab dell’auto con le 4 ruote inserite. Il prefab è stato inserito nella scena del gioco.



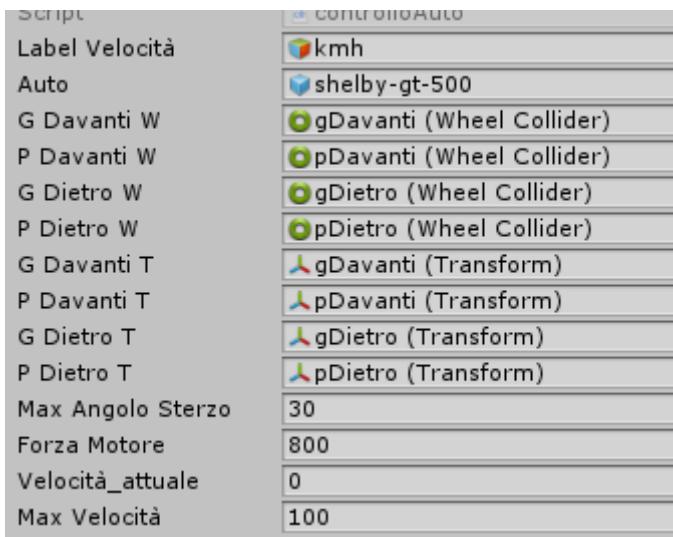
Dopodichè all’auto è stato aggiunto un componente “Rigid Body” per rendere l’auto un corpo rigido con gli opportuni valori. All’oggetto auto sono stati aggiunti quindi due “children” uno chiamato “cerchioni” e uno chiamato “cerchioniCollider”. Dentro l’oggetto “cerchioni” sono state inserite solamente le geometrie delle ruote. Invece “cerchioniCollider” è il duplicato dell’oggetto cerchioni ma senza le geometrie delle ruote. Ad ogni figlio di “cerchioniCollider” è stato aggiunto un componente “Wheel Collider” il quale ci permetterà in seguito di dare vita alle ruote con lo scripting e all’auto è stato aggiunto un componente “Box collider” per gestire le collisioni. All’oggetto auto è stato aggiunto uno script realizzato in C# chiamato “Controllo auto” per permettere il controllo dell’auto da parte del giocatore.

```

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
public class controlloAuto : MonoBehaviour {
    public GameObject labelVelocità;
    public GameObject auto;
    private float m_inputOrizzontale;
    private float m_inputVerticale;
    private float m_angoloSterzo;
    public WheelCollider gDavantiL, pDavantiL;
    public WheelCollider gDietroL, pDietroL;
    public Transform gDavantiL, pDavantiL;
    public Transform gDietroL, pDietroL;
    public float maxAngoloSterzo = 30;
    public float forzaMotore = 50;
    public float velocità_attuale;
    public float maxVelocità = 100;
    private float vel_string;
    public void GetInput()
    {
        m_inputOrizzontale = Input.GetAxis("Horizontal");
        m_inputVerticale = Input.GetAxis("Vertical");
    }
    public void Sterza()
    {
        m_angoloSterzo = maxAngoloSterzo * m_inputOrizzontale;
        gDavantiL.steerAngle = m_angoloSterzo;
        pDavantiL.steerAngle = m_angoloSterzo;
    }
    public void Acceler()
    {
        velocità_attuale = 2 * Mathf.PI * gDavantiL.radius * gDavantiL.rpm * 60 / 10000;
        vel_string = auto.GetComponent().velocity.magnitude;
        labelVelocità.GetComponent<Text>().text = (vel_string*5).ToString("F0");
        if (velocità_attuale < maxVelocità)
        {
            gDavantiL.motorTorque = m_inputVerticale * forzaMotore;
            pDavantiL.motorTorque = m_inputVerticale * forzaMotore;
        }
        else
        {
            gDavantiL.motorTorque = 0;
            pDavantiL.motorTorque = 0;
        }
    }
    if (velocità_attuale < maxVelocità)
    {
        gDavantiW.motorTorque = m_inputVerticale * forzaMotore;
        pDavantiW.motorTorque = m_inputVerticale * forzaMotore;
    }
    else
    {
        gDavantiW.motorTorque = 0;
        pDavantiW.motorTorque = 0;
    }
    private void aggiornaPosCerchioni()
    {
        aggiornaPosCerchione(gDavantiL, gDavantiT);
        aggiornaPosCerchione(pDavantiL, pDavantiT);
        aggiornaPosCerchione(gDietroL, gDietroT);
        aggiornaPosCerchione(pDietroL, pDietroT);
    }
    private void aggiornaPosCerchione(WheelCollider _collider, Transform _transform)
    {
        Vector3 _pos = _transform.position;
        Quaternion _quat = _transform.rotation;
        _collider.GetWorldPose(out _pos, out _quat);
        _transform.position = _pos;
        _transform.rotation = _quat;
    }
    private void FixedUpdate()
    {
        GetInput();
        Sterza();
        Acceler();
        aggiornaPosCerchioni();
    }
}

```

Il seguente script come si può vedere prende in input i seguenti oggetti



E' provvisto di quattro funzioni che esegue in loop.
La prima è **GetInput()** la quale si occupa ogni volta di assegnare a un valore float negativo o positivo a due variabili a seconda se l'utente prema su e giù oppure destra e sinistra. La seconda funzione è la funzione **Sterza()** che assegna ai "cerchioniCollider(delle ruote davanti)"

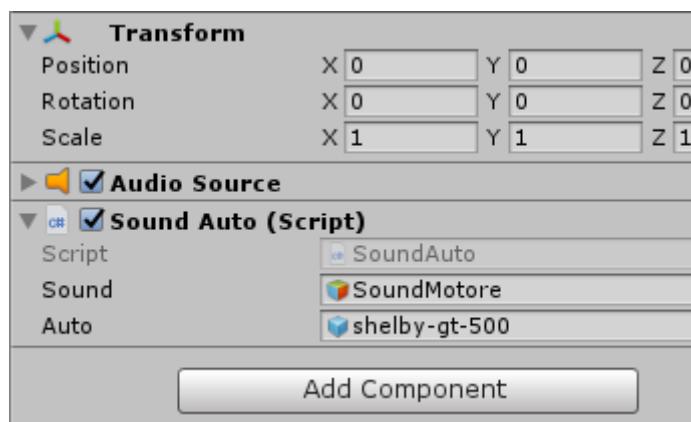
tramite **.steerAngle** il valore del massimo angolo di sterzo assegnato all'auto moltiplicato per l'input memorizzato nella variabile prima.

La terza funzione **Accelera() assegna** ai “cerchioniCollider(sempre le ruote davanti)” tramite **.motorTorque** il valore del prodotto tra la “forza_motore” assegnata all'auto e l'input ricevuto dall'utente. Inoltre controlla anche che l'auto non superi la velocità assegnata. Questa funzione come ultima cosa calcola anche la velocità dell'auto tramite **<RigidBody>.velocity.magnitude** e la assegna alla label velocità per mostrare all'utente la velocità attuale.

La quarta funzione è **aggiornaPosCerchioni()** la quale per ogni cerchione richiama un'altra funzione

aggiornaPosCerchione() che assegna ai figli dell'oggetto “cerchioni” le nuove posizione del collider dato che solamente “cerchioni possiedono le geometrie”. Con l'ultima funzione aggiorniamo fisicamente la posizione e la rotazione delle ruote.

L'oggetto “auto” possiede anche un'oggetto “SoundMotore” il quale ha come componenti un “Audio Source” e uno script chiamato “SoundMotore”. Lo script prende come input il l'oggetto “SoundMotore” e anche l'oggetto “Auto”



Lo script in questione

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

public class SoundAuto : MonoBehaviour {

    public GameObject Sound;
    public GameObject auto;
    private float velocità;
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        soundAuto();
    }

    void soundAuto()
    {
        velocità = auto.GetComponent<Rigidbody>().velocity.magnitude;
        if (velocità >= 6)
        {
            Sound.GetComponent< AudioSource >().pitch = velocità * (float)0.1;
        }
        else
        {
            Sound.GetComponent< AudioSource >().pitch = (float)0.6;
        }
    }
}
```

aumenta il pitch dell' AudioSource in base alla velocità dell'auto in modo da dare l'idea che l'auto stia accelerando. L'oggetto "Auto" possiede anche un'altro oggetto "stabilizzatoreCamera" il quale ha come figlio la "MainCamera". "stabilizzaCamera" ha uno script come componente chiamato "stabilizzaCamera"

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

public class stabilizzaCamera : MonoBehaviour {

    public GameObject auto;
    public float autoX;
    public float autoY;
    public float autoZ;

    // Update is called once per frame
    void Update () {
        autoX = auto.transform.eulerAngles.x;
        autoY = auto.transform.eulerAngles.y;
        autoZ = auto.transform.eulerAngles.z;

        transform.eulerAngles = new Vector3(autoX - autoX, autoY, autoZ - autoZ);
    }
}
```

Lo script aggiorna la posizione di “stabilizzaCamera” e conseguentemente anche di “MainCamera” rimanendo ferma anche nel caso in cui l’auto modifichi la posizione nei vettori X e Z ma seguendo solamente il vettore Y.
“Auto” possiede anche questi ultimi oggetti

```
Musica  
CameraMiniMappa  
CameraSpecchiettoRetrovisore  
PosizioneAuto
```

dove Musica è semplicemente un “ AudioSource” con la musica in background, le due camere fanno riferimento alla minimappa e allo specchietto retrovisore che verranno usati nella UI. “Posizione” auto verrà usato in seguito per verificare che l’auto dell’utente sia prima o seconda in classifica tramite script apposita.

Grafo circuito: dentro al seguente oggetto sono stati inseriti dei figli chiamati “nodi” che saranno i nodi che l’**Alauto** dovrà seguire durante il percorso. All’oggetto “Grafo circuito” è stato assegnato un script apposito per rendere più facile il disegno del percorso nella scena.

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

public class GrafoCircuito : MonoBehaviour {

    public Color coloreLinea;

    private List<Transform> nodi = new List<Transform>();

    void OnDrawGizmos()
    {
        Gizmos.color = coloreLinea;

        Transform[] trasformaGrafo = GetComponentsInChildren<Transform>();

        nodi = new List<Transform>();

        for(int i = 0; i < trasformaGrafo.Length; i++)
        {
            if(trasformaGrafo[i] != transform)
            {
                nodi.Add(trasformaGrafo[i]);
            }
        }

        for (int i = 0; i < nodi.Count; i++)
        {
            Vector3 nodoCorrente = nodi[i].position;
            Vector3 nodoPrecedente = Vector3.zero;
            if (i > 0)
            {
                nodoPrecedente = nodi[i - 1].position;
            }
            else if (i == 0 && nodi.Count > 1)
            {
                nodoPrecedente = nodi[nodi.Count - 1].position;
            }

            Gizmos.DrawLine(nodoPrecedente, nodoCorrente);
        }
    }
}
```

Lo script prende in input l’oggetto “Grafo Circuito” e aggiunge a una lista tutti i nodi dell’oggetto e successivamente tramite “Gizmos” va a disegnare le linee che corrispondono alle distanze tra un nodo e l’altro.

Come possiamo vedere nella foto in seguito il risultato finale è questo



Quindi l'**Alauto** dovrà seguire il percorso delineato in bianco.

Alauto: questo oggetto sarà l'auto contro la quale l'utente dovrà competere. L'oggetto è una copia dell'oggetto "Auto" solamente che non possiede più lo script "Controllo Auto" ma bensì un'altro script apposito chiamato Aimotore che darà vita alla nostra Alauto. Questo script prende in input i seguenti oggetti



Come possiamo vedere gli oggetti che prende in input sono gli stessi dell'oggetto "Auto" solamente che in questo caso bisogna inserire anche il grafo contenete i nodi che l'auto dovrà seguire.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class AIMotore : MonoBehaviour {
6      //Grafo circuito inseriamo il GrafoPadre
7      public Transform GrafoCircuito;
8      public float maxAngoloSterzo = 30f;
9      public float forzaMotore = 600f;
10     private float f_motore_attuale = 0;
11     public float maxVelocità = 100;
12     public float velocità_attuale;
13     public WheelCollider gDavantiW, pDavantiW;
14     public WheelCollider gDietroW, pDietroW;
15     public Transform gDavantiT, pDavantiT;
16     public Transform gDietroT, pDietroT;
17     //Qui inseriamo tutto i nodi creati
18     private List<Transform> nodi;
19     private int nodoCorrente = 0;
20
21     void Start () {
22         Transform[] trasformaGrafo = GrafoCircuito.GetComponentsInChildren<Transform>();
23
24         nodi = new List<Transform>();
25
26         for (int i = 0; i < trasformaGrafo.Length; i++)
27         {
28             if (trasformaGrafo[i] != GrafoCircuito.transform)
29             {
30                 nodi.Add(trasformaGrafo[i]);
31             }
32         }
33     }
34
35     private void FixedUpdate () {
36         Sterza();
37         Accelerata();
38         controlloDistanzaNodo();
39         aggiornaPosCerchioni();
40     }
41
42     private void Sterza()
43     {
44         Vector3 vettoreRelativo = transform.InverseTransformPoint(nodi[nodoCorrente].position);
45         float nuovaSterzata = (vettoreRelativo.x / vettoreRelativo.magnitude) * maxAngoloSterzo;
46         gDavantiW.steerAngle = nuovaSterzata;
47         pDavantiW.steerAngle = nuovaSterzata;
48     }
49
50 }
51
52
private void Accelerata()
{
    velocità_attuale = 2 * Mathf.PI * gDavantiW.radius * gDavantiW.rpm * 60 / 10000;

    if (velocità_attuale < maxVelocità)
    {
        gDavantiW.motorTorque = f_motore_attuale;
        pDavantiW.motorTorque = f_motore_attuale;
    }
    else
    {
        gDavantiW.motorTorque = 0;
        pDavantiW.motorTorque = 0;
    }
}

//controlla se l'auto è molto vicina al nodo cambia nodo e va al nodo successivo nel caso lo sia
private void controlloDistanzaNodo()
{
    //frena se in prossimità di un nodo
    if (Vector3.Distance(transform.position, nodi[nodoCorrente].position) < 10.6f && Vector3.Distance(transform.position, nodi[nodoCorrente].position) > 0.5f)
    {
        f_motore_attuale = -forzaMotore;
    }
    else
    {
        f_motore_attuale = forzaMotore;
    }
    //aggiorna nodo
    if (Vector3.Distance(transform.position, nodi[nodoCorrente].position) < 1f)
    {
        if(nodoCorrente == nodi.Count - 1)
        {
            nodoCorrente = 0;
        }
        else
        {
            nodoCorrente++;
        }
    }
}

private void aggiornaPosCerchioni()
{
    aggiornaPosCerchione(gDavantiW, gDavantiT);
    aggiornaPosCerchione(pDavantiW, pDavantiT);
    aggiornaPosCerchione(gDietroW, gDietroT);
    aggiornaPosCerchione(pDietroW, pDietroT);
}
private void aggiornaPosCerchione(WheelCollider _collider, Transform _transform)
{
    Vector3 _pos = _transform.position;
    Quaternion _quat = _transform.rotation;

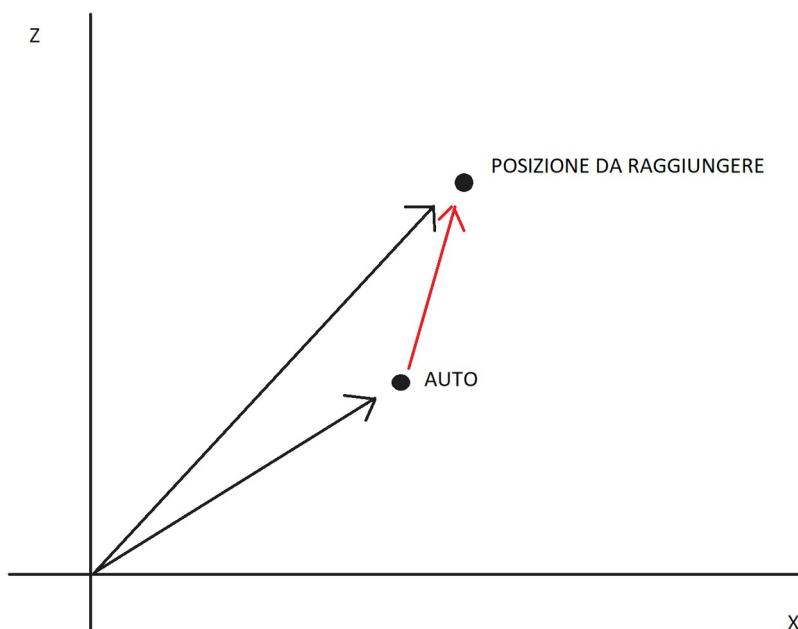
    _collider.GetWorldPose(out _pos, out _quat);

    _transform.position = _pos;
    _transform.rotation = _quat;
}

```

Lo script in questione nell'inizializzazione aggiunge i nodi del grafo in una lista “nodi”. Successivamente come nell'oggetto “Auto” va ad eseguire il loop le seguenti funzioni **Sterza()**, **Accelera()**, **controllaDistanzaNodo()**, **aggiornaPosCerchioni()**.

Sterza() è una funzione che come prima cosa si ricava il vettore rosso in figura che corrisponde al vettore tra l'auto e il punto da raggiungere.



Per fare questo è stata usata la funzione “`inverseTransformPoint`”. Successivamente è stato assegnato ai “`cerchioniCollider.steerAngle`” davanti il valore della divisione tra il punto x del vettore rosso e la lunghezza del vettore rosso, il tutto moltiplicato per il massimo angolo di sterzo dell'auto.

Accelera() calcola la velocità attuale dell'auto e controlla che non superi la velocità massima dell'automobile. Quando la

velocità massima non è superata assegna ai “cerchioniCollider” davanti il valore della forza motore. **ControlloDistanzaNodo()** funzione che come prima cosa controlla che in caso la distanza tra l’auto e il punto di arrivo sia minore di **alpha** e maggiore di **beta** assegna alla variabile forza_motore il valore inverso in modo che l’auto freni. E’ stato deciso di utilizzare **beta** in modo che quando l’auto sia molto vicina al punto di arrivo non freni più da fermarsi ma ricominci ad accellerare. Successivamente questa funzione controlla che la posizione dell’auto sia minore di **gamma** e nel cosi dovesse essere così allora provvede ad aggiornare il nodo successivo che l’auto dovrà seguire.

AggiornaPosCerchi() funziona esattamente come nell’oggetto “Auto”;

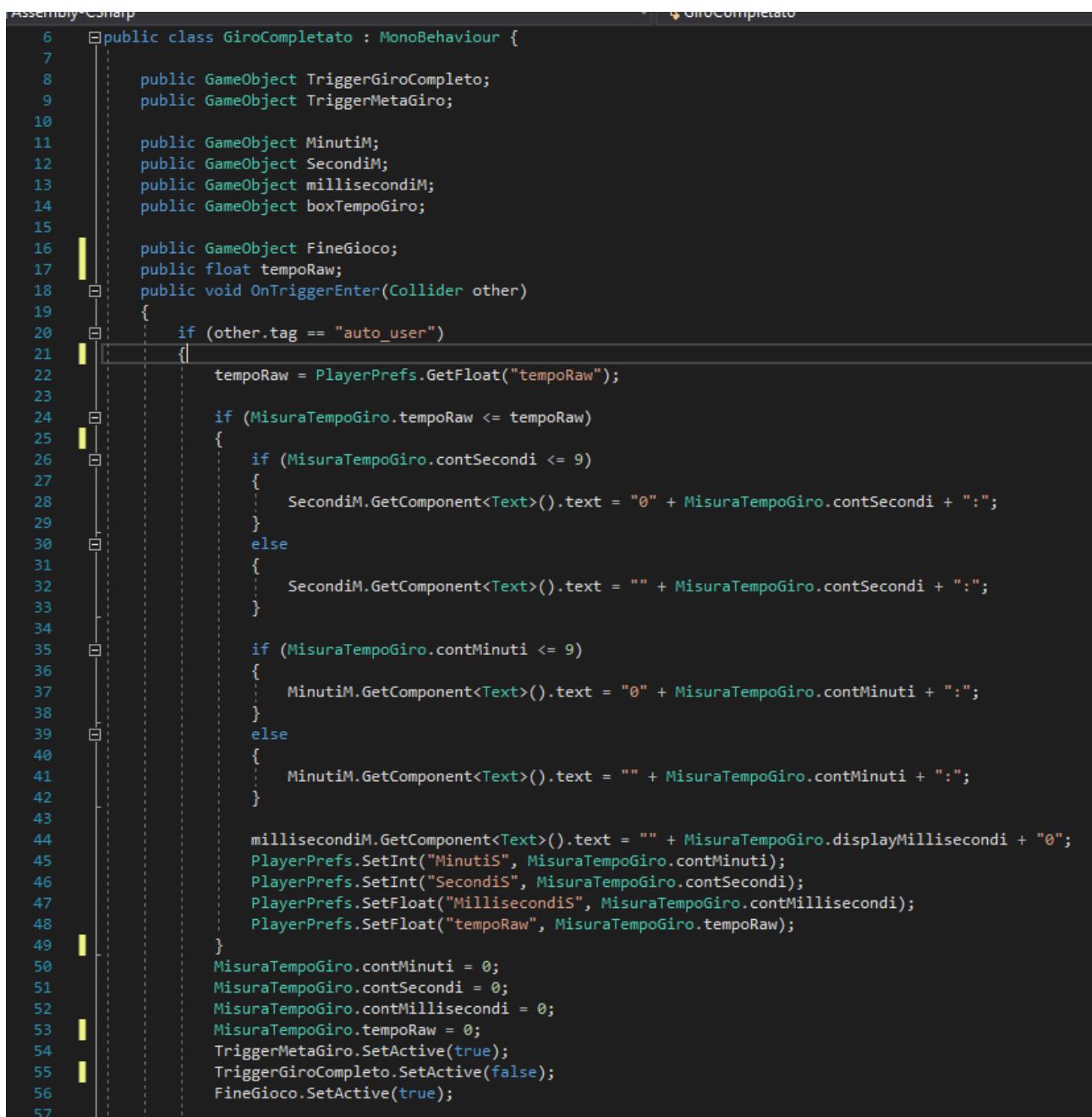
L’oggetto “Aiauto” contiene anche altri due figli chiamati rispettivamente posizione_primo e posizione_secondo che contengono due script dove ognuno controlla se il cubo definito in “Auto” entra in collisione con questi due cubi distanziati. Nel caso il cubo dell’oggetto “Auto” stia uscendo dal trigger posizione_primo posizionato più avanti dell’Alauto allora lo script scrive sulla UI che la posizione dell’utente è 1 invece nel caso il cubo di auto stia uscendo dal trigger posizione_secondo il quale è posizionato più dietro del trigger posizione_primo allora provvederà a scrivere nella UI che la posizione del giocatore è la 2.

TriggerMetaGiro: cubo non renderizzato che copre in posizione adiacente la larghezza della pista circa a metà percorso. Questo oggetto come componente ha un box collider assegnato come Trigger e uno script che prende in input il Trigger stesso e il Trigger “TriggerGiroCompleto”.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class TriggerMetaPercorso : MonoBehaviour {
6
7      public GameObject TriggerGiroCompleto;
8      public GameObject TriggerMetaGiro;
9
10     void OnTriggerEnter(Collider other)
11     {
12         if(other.tag == "auto_user")
13         {
14             TriggerGiroCompleto.SetActive(true);
15             TriggerMetaGiro.SetActive(false);
16         }
17     }
18 }
19
20
```

Quindi se l’auto dell’utente attraversa il cubo allora lo script provvede ad attivare il trigger “TriggerGiroCompleto” e disattivare se stesso. Da notare che all’auto dell’user è stato assegnato un “tag” in modo che il lo script funzioni solamente se l’auto che è entrata in collisione sia quella dell’utente e non la “Aiauto”.

TriggerGiroCompleto: come nel trigger precedente è stato assegnato ad un cubo posizionato questa volta all'inizio della corsa pero il seguente oggetto è disattivato in modo che venga attivato successivamente dallo script del “TriggerMetaGiro”.



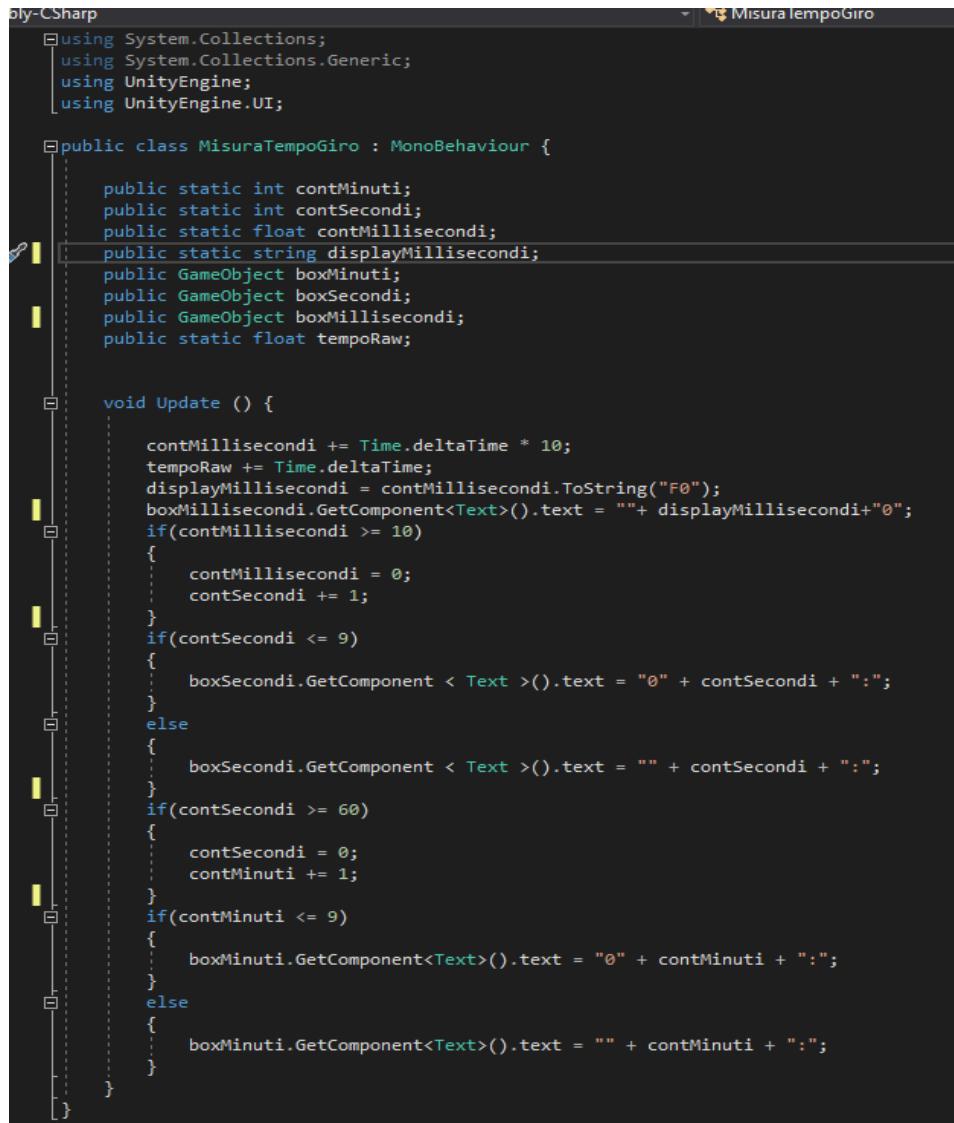
The screenshot shows the Unity Editor's code editor window with the file "GiroCompleto.cs" open. The code is written in C# and defines a class "GiroCompleto" that inherits from "MonoBehaviour". The class contains several public fields: "TriggerGiroCompleto", "TriggerMetaGiro", "MinutiM", "SecondiM", "millisecondiM", and "boxTempoGiro". It also has a field "FineGioco" and a float "tempoRaw". The "OnTriggerEnter(Collider other)" method is implemented to handle collisions. If the collision involves an "auto_user" tag, it retrieves the raw time from PlayerPrefs and updates three UI Text components (MinutiM, SecondiM, millisecondiM) to display the time in a two-digit format (e.g., "05:03"). It also saves the raw time and the current time to PlayerPrefs. Finally, it activates "TriggerMetaGiro" and deactivates "TriggerGiroCompleto" and "FineGioco".

```
Assembly-CSharp
6  public class GiroCompleto : MonoBehaviour {
7
8      public GameObject TriggerGiroCompleto;
9      public GameObject TriggerMetaGiro;
10
11     public GameObject MinutiM;
12     public GameObject SecondiM;
13     public GameObject millisecondiM;
14     public GameObject boxTempoGiro;
15
16     public GameObject FineGioco;
17     public float tempoRaw;
18     public void OnTriggerEnter(Collider other)
19     {
20         if (other.tag == "auto_user")
21         {
22             tempoRaw = PlayerPrefs.GetFloat("tempoRaw");
23
24             if (MisuraTempoGiro.tempoRaw <= tempoRaw)
25             {
26                 if (MisuraTempoGiro.contSecondi <= 9)
27                 {
28                     SecondiM.GetComponent<Text>().text = "0" + MisuraTempoGiro.contSecondi + ":";
29                 }
29                 else
30                 {
31                     SecondiM.GetComponent<Text>().text = "" + MisuraTempoGiro.contSecondi + ":";
32                 }
33
34                 if (MisuraTempoGiro.contMinuti <= 9)
35                 {
36                     MinutiM.GetComponent<Text>().text = "0" + MisuraTempoGiro.contMinuti + ":";
37                 }
37                 else
38                 {
39                     MinutiM.GetComponent<Text>().text = "" + MisuraTempoGiro.contMinuti + ":";
40                 }
41
42                 millisecondiM.GetComponent<Text>().text = "" + MisuraTempoGiro.displayMillisecondi + "0";
43                 PlayerPrefs.SetInt("MinutiS", MisuraTempoGiro.contMinuti);
44                 PlayerPrefs.SetInt("SecondiS", MisuraTempoGiro.contSecondi);
45                 PlayerPrefs.SetFloat("MillisecondiS", MisuraTempoGiro.contMillisecondi);
46                 PlayerPrefs.SetFloat("tempoRaw", MisuraTempoGiro.tempoRaw);
47             }
48         }
49     }
50     MisuraTempoGiro.contMinuti = 0;
51     MisuraTempoGiro.contSecondi = 0;
52     MisuraTempoGiro.contMillisecondi = 0;
53     MisuraTempoGiro.tempoRaw = 0;
54     TriggerMetaGiro.SetActive(true);
55     TriggerGiroCompleto.SetActive(false);
56     FineGioco.SetActive(true);
57 }
```

Questo script come nel precedente controlla se l'auto che collide il cubo sia quella dell'utente tramite il

tag "auto_user". Poi assegna ad una variabile "temporaw" il tempo migliore che è stato fatto in pista salvato in una variabile globale in PlayerPref che sarà presente anche alla prossima riapertura dell'applicazione. Poi controlla se il miglior tempo dentro allo script "MisuraTempoGiro" sia minore di quello salvato globalmente, nel caso sia così allora lo script scrive nella label del "tempo migliore" il risultato del tempo migliore, poi una volta stampato nella UI il tempo migliore provvede a salvare in delle variabili globali il nuovo tempo migliore. Così da poterlo confrontare anche latre volte. Successivamente lo script attiva lo script meta giro e disattiva se stesso ma questo è stato implementato nel caso ci siano piu giri da fare ma è stato deciso di lasciare solamente un giro di pista data la vastità della pista. Lo script alla fine attiva l'oggetto "FineGioco".

MisuraTempoGiro: oggetto che contiene uno script che misura il tempo del giro corrente e lo stampa nella label della UI “tempo giro”



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

public class MisuraTempoGiro : MonoBehaviour {

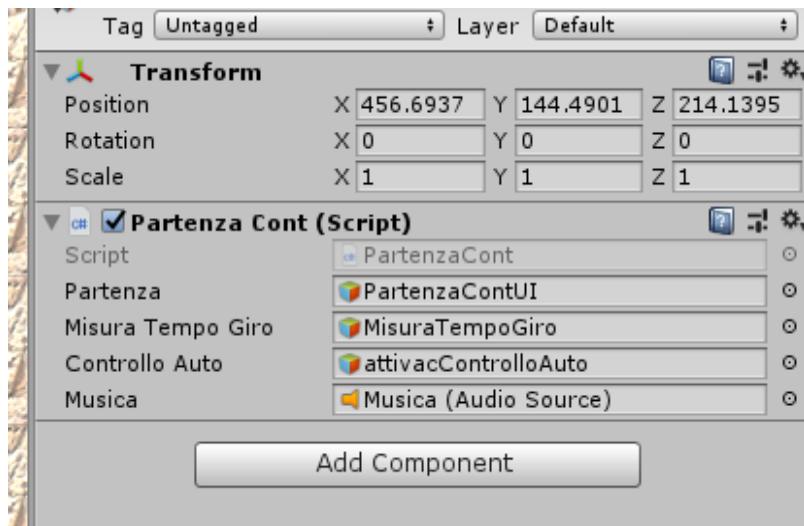
    public static int contMinuti;
    public static int contSecondi;
    public static float contMillisecondi;
    public static string displayMillisecondi;
    public GameObject boxMinuti;
    public GameObject boxSecondi;
    public GameObject boxMillisecondi;
    public static float tempoRaw;

    void Update () {

        contMillisecondi += Time.deltaTime * 10;
        tempoRaw += Time.deltaTime;
        displayMillisecondi = contMillisecondi.ToString("F0");
        boxMillisecondi.GetComponent<Text>().text = "" + displayMillisecondi + "0";
        if(contMillisecondi >= 10)
        {
            contMillisecondi = 0;
            contSecondi += 1;
        }
        if(contSecondi <= 9)
        {
            boxSecondi.GetComponent < Text >().text = "0" + contSecondi + ":";
        }
        else
        {
            boxSecondi.GetComponent < Text >().text = "" + contSecondi + ":";
        }
        if(contSecondi >= 60)
        {
            contSecondi = 0;
            contMinuti += 1;
        }
        if(contMinuti <= 9)
        {
            boxMinuti.GetComponent<Text>().text = "0" + contMinuti + ":";
        }
        else
        {
            boxMinuti.GetComponent<Text>().text = "" + contMinuti + ":";
        }
    }
}
```

Da notare anche la variabile static tempoRaw utilizzata dallo script precedente per confrontare il miglior tempo con quello appena effettuato.

ManagerPartenza: oggetto che contiene uno script che prende in input i seguenti oggetti



```
Assembly-CSharp
 4  using UnityEngine;
 5  using System;
 6  using System.Threading;
 7  public class PartenzaCont : MonoBehaviour {
 8
 9      public GameObject partenza;
10     public GameObject MisuraTempoGiro;
11     public GameObject controlloAuto;
12     public AudioSource Musica;
13
14
15     void Start () {
16
17         StartCoroutine(startContatore());
18
19     }
20
21     IEnumerator startContatore()
22     {
23
24         partenza.GetComponent<Text>().text = "3";
25         yield return new WaitForSeconds(1);
26         partenza.GetComponent<Text>().text = "2";
27         yield return new WaitForSeconds(1);
28         partenza.GetComponent<Text>().text = "1";
29         yield return new WaitForSeconds(1);
30         partenza.GetComponent<Text>().text = "VIA!";
31         MisuraTempoGiro.SetActive(true);
32         controlloAuto.SetActive(true);
33         yield return new WaitForSeconds(1);
34         partenza.GetComponent<Text>().text = "";
35         Musica.Play();
36
37 }
```

Questo script fa il conto alla rovescia prima della partenza nella UI. Una volta finito il conto alla rovescia lo script attiva l'oggetto “attivaControlloAuto” il quale a sua volta contiene uno script che sblocca i comandi dell’auto normale e dell’Aiauto. Poi riattiva l’oggetto “MisuraTempoGiro” e successivamente anche la musica in background.

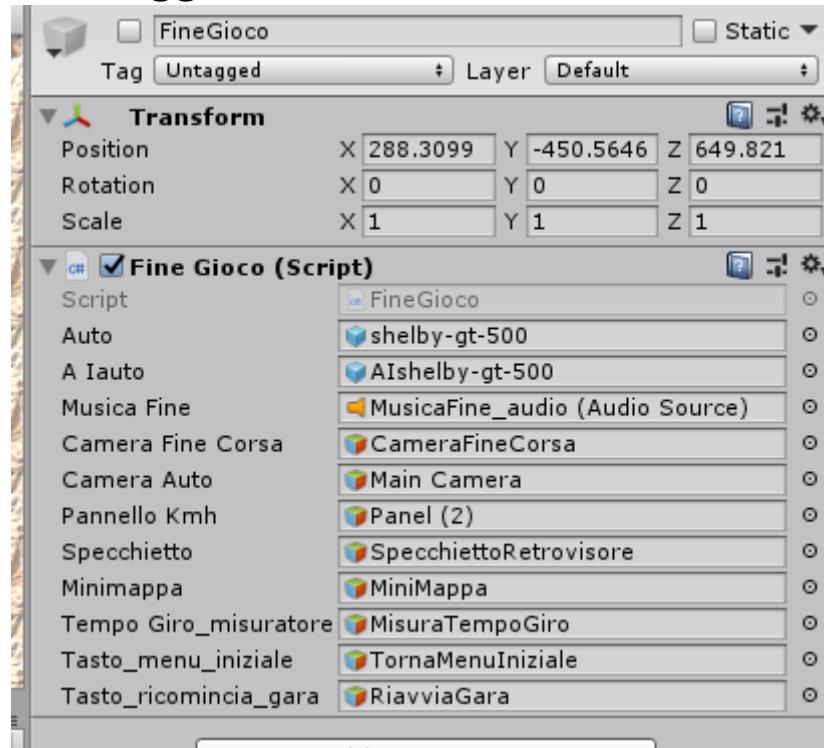
AttivaControlloAuto: oggetto contenente uno script che riattiva gli script per il controllo delle auto nei due oggetti delle auto.

CaricaTempoGiro: oggetto che contiene uno script che prende il miglior tempo salvato nelle variabili globali e lo carica nella UI sulla voce miglior tempo.

Tasti: oggetto che contiene uno script in cui sono definite delle funzioni per ogni bottone del gioco. Questo oggetto viene assegnato ai due bottoni nella UI che permettono la scelta di ricominciare il gioco richiamando la stessa scena oppure chiamando il bottone menu iniziale che richiama la scena del menu iniziale.

caduta_auto: oggetto assegnato a un cubo esteso in tutta l’area sottostante al circuito in modo che nel caso l’auto dell’utente caschi sotto al circuito allora lo script allegato all’oggetto attiva l’oggetto “FineGioco”.

FineGioco: oggetto con assegnato uno script che prende in input i seguenti oggetti



```
Assembly-CSharp
```

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class FineGioco : MonoBehaviour {
6
7      public GameObject auto;
8      public GameObject AIauto;
9      public AudioSource musicaFine;
10     public GameObject cameraFineCorsa;
11     public GameObject cameraAuto;
12     public GameObject pannelloKmh;
13     public GameObject specchietto;
14     public GameObject minimappa;
15     public GameObject tempoGiro_misuratore;
16     public GameObject tasto_menu_iniziale;
17     public GameObject tasto_ricomincia_gara;
18
19
20     void Start () {
21         auto.SetActive(false);
22         AIauto.SetActive(false);
23         musicaFine.Play();
24         tempoGiro_misuratore.GetComponent<MisuraTempoGiro>().enabled = false;
25         tasto_menu_iniziale.SetActive(true);
26         tasto_ricomincia_gara.SetActive(true);
27         pannelloKmh.SetActive(false);
28         specchietto.SetActive(false);
29         minimappa.SetActive(false);
30         cameraAuto.SetActive(false);
31         cameraFineCorsa.SetActive(true);
32     }
33
34 }
```

Lo script in questione appena viene chiamato come prima cosa disattiva le due auto. Poi fa partire la musica di fine gioco, disattiva il “MisuraTempoGiro”, attiva i bottoni per tornare al menu iniziale o ricominciare la gara, disattiva il pannello del contachilometri, disattiva lo specchietto retrovisore, la minimappa, la camera dell’auto e attiva la camera di fine gara.

SCENA menulniziale: in questa scena sono stati aggiunti due bottoni ai quali è stato assegnato lo script Tasti che richiama la scena del tracciato in caso si prema GIOCA ed esce dal gioco nel caso si scelga ESCI DAL GIOCO.



Lista oggetti utilizzati

```
▼ Pista1
  Directional Light
  ▼ shelby-gt-500
    ▼ Cerchioni
      gDietro
      gDavanti
      pDietro
      pDavanti
    ▼ colliderCerchioni
      gDietro
      gDavanti
      pDietro
      pDavanti
      SoundMotore
    ▼ stabilizzatoreCamera
      ► Main Camera
      Musica
      CameraMiniMappa
      CameraSpecchiettoRetrovisore
      PosizioneAuto
      Terrain
    ▼ finishOggetto
      Cylinder
      Cylinder (1)
      Cube
    ▼ Canvas
      ▼ Panel
        labelTempoGiro
        displayMinuti
        displaySecondi
        displayMillisecondi
        MlabelTempoGiro
        MdisplayMinuti
        MdisplaySecondi
        MdisplayMillisecondi
      ▼ Panel (1)
        PosizioneUI
        n_posizione
        PartenzaContUI
      ▼ MiniMappa
        MiniMappaRender
      ▼ SpecchiettoRetrovisore
        RenderSpecchietto
      ▼ Panel (2)
        labelKmh
        kmh
      ▼ RiavviaGara
        Text
      ▼ TornaMenuIniziale
        Text
      EventSystem
      TriggerGiroCompleto
      TriggerMetaGiro
      MisuraTempoGiro
      ManagerPartenza
      attivacControlloAuto
    ► GrafoCircuito
    ▼ A1shelby-gt-500
      ▼ Cerchioni
        gDietro
        gDavanti
        pDietro
        pDavanti
      ▼ colliderCerchioni
        gDietro
        gDavanti
        pDietro
        pDavanti
        SoundMotore
        Posizione_primo
        Posizione_secondo
        CaricaTempoGiro
        CameraFineCorsa
      ▼ MusicaFine
        MusicaFine_audio
        Tasti
        caduta_auto
        FineGioco
        terreno_montagne
        esc_button
    ► Alberi
```