0.1 Interfaccia grafica

L'interfaccia grafica dell'applicativo è stata realizzata in Unity, un popolare motore grafico per videogiochi e applicazioni android e ios. E' stato realizzato un rig (un manichino) attraverso blender con uno scheletro interfacciabile con Unity, in modo da poterne controllare i movimenti attraverso l'inserimento degli angoli dei vari "ossi" dello scheletro. Lo script dell'interfaccia grafica è stato scritto in C# e si basa su un client che richiede continuamente al server i dati di uscita della rete neurale.

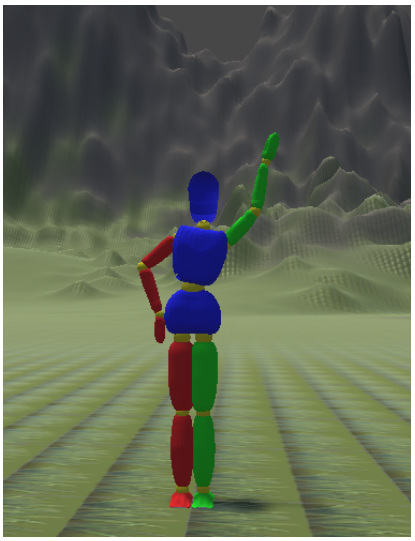

Figura 1: rig utilizzato per la renderizzazione

0.1.1 classe syncSocketServer

Questa parte illustra la classe utilizzata per far effettuare allo script in unity la richieta al server centrale dei dati elaborati.

0.1.1.1 librerie utilizzate

tale classe è stata implementata nella parte unity e ha richiesto l'importazione di UnityEngine, oltre alle librerie per la comunicazione socket e per la manipolazione dei byte ricevuti dal main server.

```
1 using System;
2 using System.Net;
3 using System.Net.Sockets;
4 using System.Runtime.InteropServices;
5 using System.Text;
6 using System.Collections;
7 using System.Collections.Generic;
8 using UnityEngine;
```

Listing 1: librerie class client C#

0.1.1.2 classe client C# per la comunicazione

In questa sezione è illustrato la classe che permette la comunicazione con il main server. Si noti come l'utilizzo della classe in unity richiede che essa sia figlia della classe MonoBehaviour, interna di unity.

```
1
2 public class SyncSocketClient : MonoBehaviour {
3     //struttura union per la conversione Bytes -> float
4     [StructLayout(LayoutKind.Explicit)]
5     public struct Ubones
6     {
7         [FieldOffset(0)] public float fBones;
8         [FieldOffset(0)] public byte strBones1;
9         [FieldOffset(1)] public byte strBones2;
10        [FieldOffset(2)] public byte strBones3;
11        [FieldOffset(3)] public byte strBones4;
12    }
13
14    //Oggetti socket
```

```
15      public IPHostEntry ipHostInfo;
16      public IPAddress ipAddress;
17      public IPEndPoint remoteEP;
18      public Socket sender;
19      private const int number_of_joints = 2;
20      private const int number_of_float = number_of_joints * 3;
21
22
23      // alloco la struttura di conversione
24      public Ubones[] bones = new Ubones[number_of_float]; //4 joints
25
26      // dichiaro il vettore di buffer
27      public byte[] bytes;
28
29      //settaggio parametri e inizzializzazione comunicazione
30      public SyncSocketClient(int Port){ ... }
31
32      //chiusura comunicazione
33      public void CloseConnection(){ ... }
34
35      // funzione per la ricezione e la conversione float
36      public void ReciveFloat(){ ... }
37
38    //funzione per la ricezione della stringa dal server
39      public string ReciveString(){ ... }
40
41      //trasmissione stringa
42      public void sendMessage(string msg){ ... }
43
44      //funzione di comunicazione, invia il codice di richiesta, riceve i byte li carica nell'
            ↪ union
45      public void TestComunication(){ ... }
46
47    //funzione per la reinizializzazione delle variabili bones
48      public void initBones(){ ... }
49      }
50  }
```

Listing 2: class client C#

0.1.2 classe di controllo del manichino

In questa sezione viene illustrata la classe interna al motore grafico realizzata per il caricamento dei dati ricevuti dal server nella struttura di dell'oggetto

0.1.2.1 librerie della classe controllo

```
1 using System.Collections.Generic;
2 using System.IO;
3 using System.Text;
4 using UnityEngine;
```

Listing 3: Librerie classe controllo C#

0.1.2.2 corpo della classe controllo

Anche questa classe è stata resa figlia di MonoBehaviour, il settaggio delle variabili viene effettuat dal metodo Start() richiamato da Unity all'avvio del programma, in questo caso però l'utilizzo della classe è richiamato attivamente da Unity attraverso il metodo Update() che viene richiamato una volta per ogni aggiornamento dello schermo.

```
1 public class ProvaControllo : MonoBehaviour {
2   //dichiarazione della classe socketClient sincrono
3   //e dell'oggetto di controllo del manichino
4     private SyncSocketClient socket;
5     public GameObject[] bones;
6
7     private List<string> Tags = new List<string>();
```

```
8
9       /*{ 4, 6, 3, 5, 8};*/ // indici degli oggetti da aggiornare nella lista GameObject
10      public int[] neededObjects = new int[] {3, 5};
11
12      //ref kinect{ 4, 5, 8, 9}; // indici degli oggetti da aggiornare nella struct Ubones della
            ↪ com socket
13      public int[] neededBonesF = new int[] {0, 1};
14
15    //vettori da tre per il passaggio degli angoli
16      public Vector3 BraccioDx;
17      public Vector3 avanBraccioDx;
18      public Vector3 BraccioSx;
19      public Vector3 avanBraccioSx;
20
21      // Metodo di inizializazzione della classe, richiamato all'avvio
22      void Start () {
23
24        //attivazione della comunicazione
25          socket = new SyncSocketClient(9999);
26
27          //inizializazzione della struttura di conversione byte/float
28          socket.initBones();
29
30      //creazione della lista dei tag
31          Tags.Add("Root"); //0
32          Tags.Add("Bacino"); //1
33          Tags.Add("schiena"); //2
34          Tags.Add("BraccioDx"); //3
35          Tags.Add("BraccioSx"); //4
36          Tags.Add("AvanbraccioDx"); //5
37          Tags.Add("AvanbraccioSx"); //6
38          Tags.Add("testa"); //7
39          Tags.Add("schiena"); //8
40
41      //inizializazzione dell'array di oggetti del gioco
42          bones = new GameObject[10];
43
44          //link degli oggetti creati alle ossa del manichino
45          int i = 0;
46          foreach (string Tag in Tags) {
47              bones[i] = GameObject.FindGameObjectWithTag(Tag);
48              i++;
49          }
50      }
51
52    // Update is called once per frame
53    void Update () {
54
55      //richiesta al server centrsle dei nuovi dati e aggiornamento
56      //della struttura interna alla classe socket
57          socket.TestComunication();
58
59      //caricamento dei dati ricevuti sulla struttura unity del manichino
60          for (int i = 0; i < 2; i++)
61          {
62              bones[neededObjects[i]].transform.rotation = Quaternion.Euler(socket.bones[
                    ↪ neededBonesF[i]*3].fBones, socket.bones[(neededBonesF[i] * 3) + 1].fBones,
                    ↪ socket.bones[(neededBonesF[i] * 3) + 2].fBones);
63          }
64  }
```

Listing 4: class client C#