Vitual Tennis+

Esame di **Realtà Virtuale**

Studente **Tartaglia Luca**

Corso di Studi di **Informatica**

Università degli Studi di Napoli Parthenope

Sommario

Introduzione

Il progetto sviluppato è un gioco di realtà virtuale per giocatore singolo che simula una partita di tennis. La caratteristica principale del gioco è l'interconnessione con un'app companion installata su smartphone, che migliora l'immersione del giocatore. Grazie all'integrazione tra app e gioco, l'utente può vivere un'esperienza più coinvolgente, utilizzando il dispositivo mobile per influenzare i colpi e le dinamiche di gioco.

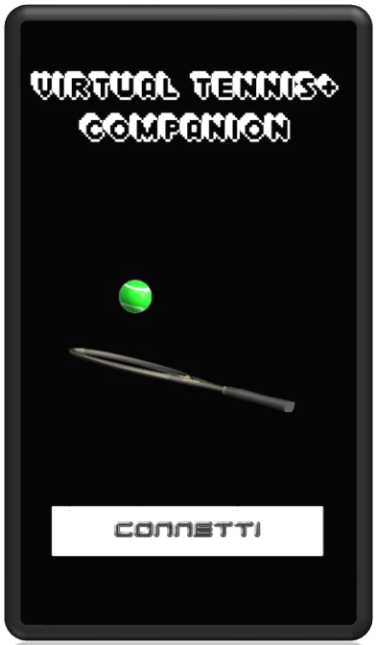
Il gioco prevede due modalità principali: **Arcade** e **Torneo**. Nella modalità Arcade, il giocatore può scegliere il livello di difficoltà dell'avversario (facile, medio, difficile) tramite un menu intuitivo, navigando con i comandi dell'app companion o con frecce direzionali e tasto invio. Nella modalità Torneo, il giocatore affronta avversari di difficoltà crescente, ottenendo un trofeo per ogni vittoria. Alla prima sconfitta, si torna al menu principale, perdendo i progressi accumulati.

L'app companion è fondamentale perché fornisce un ulteriore strumento di controllo e interazione. Si può giocare sia tramite tastiera che app, con quest'ultima che offre un'interazione più immersiva e naturale. Per esempio, l'app distingue tra colpi "forti" o "deboli" in base alla velocità del gesto, rendendo l'esperienza più realistica.

Il sistema di gioco è semplice, suddiviso in due tipi di colpi principali: colpi di risposta, che devono essere sincronizzati e precisi, e colpi di battuta, distinti in "forti" o "deboli" in base alla rapidità del gesto dell'utente.

Dal punto di vista tecnico, il progetto utilizza un'architettura client-server, in cui il gioco funge da server e l'app companion da client. La connessione avviene in due fasi: l'app invia un broadcast UDP sulla porta 5001 per individuare il server e stabilisce una connessione TCP/IP al ricevimento del messaggio di risposta. Il sistema è basato sulle librerie Mono integrate in Unity, e la comunicazione avviene continuamente, con un meccanismo di riconnessione automatica in caso di disconnessione per garantire la fluidità dell'esperienza di gioco.

Il progetto prevede la creazione di diversi componenti, ognuno con funzioni specifiche per l'app companion e il gioco principale. Ogni componente sarà illustrato attraverso schermate e descrizioni dei file che ne regolano il funzionamento.



Prime schermate

Le **prime schermate** principali del progetto includono il menu iniziale, la scelta del giocatore e la scelta dello scenario di gioco. Queste schermate costituiscono l'interfaccia di base attraverso cui l'utente interagisce con il gioco e seleziona le opzioni di configurazione prima di iniziare una partita. Il progetto utilizza classi dedicate, come CaricaPersonaggio e CaricaScena, rispettivamente per caricare i modelli dei personaggi e degli ambienti di gioco. Sono stati implementati due singleton: Musica e RacchettaManager, che gestiscono la colonna sonora e la configurazione dei game object relativi alle racchette e agli altri asset di gioco.

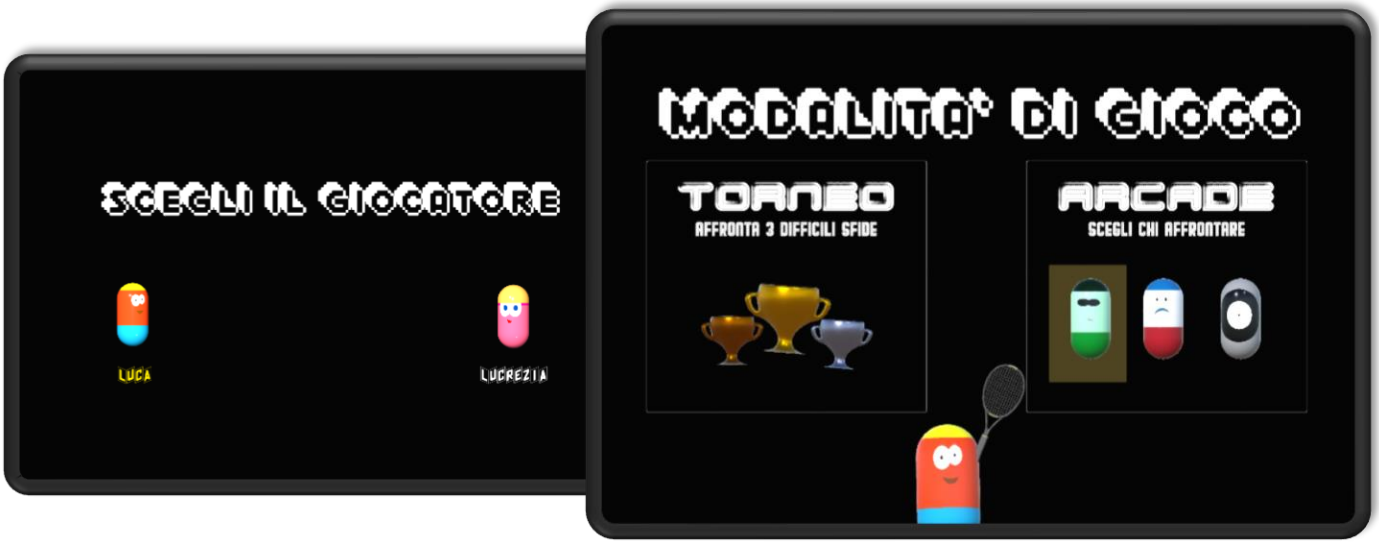
Schermata iniziale

La schermata iniziale ha lo scopo di guidare l'utente nel collegamento con l'app companion e avvia RacchettaManager e Musica, caricati come singleton in un GameObject vuoto. RacchettaManager gestisce l'inizializzazione delle racchette e le dinamiche di gioco legate ad esse, mentre Musica si occupa della colonna sonora. Inoltre, questa schermata mostra animazioni in base allo stato del collegamento con l'app companion. I tre scenari principali sono:

1. **Collegamento non riuscito**: dovuto a errori di calibrazione o connessione.
2. **Ignorare il collegamento**: premendo invio si può saltare il collegamento.
3. **Collegamento riuscito**: il gioco prosegue automaticamente allo scenario successivo.

Scelta del personaggio e dello scenario di gioco

Dopo la schermata iniziale, l'utente può scegliere tra i personaggi giocabili Luca o Lucrezia e selezionare una delle modalità di gioco disponibili. La selezione può avvenire tramite tastiera o comandi dall'app companion. I comandi dall'app vengono gestiti tramite la funzione DataReceived, che classifica i dati ricevuti per determinare l'azione appropriata, come la scelta del personaggio o dello scenario. Le selezioni vengono poi salvate nelle **PlayerPrefs di Unity** tramite SetString, per garantire che i parametri e i prefab siano caricati correttamente. Ad esempio, se l'utente seleziona il personaggio "Luca" e la modalità "Facile" dello scenario "Arcade", queste informazioni vengono richiamate all'avvio del gioco per caricare i prefab e le configurazioni corrette.



Funzioni e classi di supporto

CaricaPersonaggio

CaricaPersonaggio è una classe statica progettata per gestire il caricamento dinamico dei personaggi giocabili e degli avversari nel gioco. La sua funzione principale è sostituire il **gameobject "giocatore"** con un altro prefab specificato. Questo processo avviene tramite un parametro chiamato nomeGiocatore, che fa riferimento a una risorsa presente nella cartella **Resources**.

Quando viene invocata, **CaricaPersonaggio** elimina il modello del personaggio esistente e carica dinamicamente il nuovo prefab richiesto, assicurando che ogni partita utilizzi il personaggio corretto in base alle selezioni fatte dall'utente nel menu iniziale.

CaricaScena

La classe **CaricaScena** gestisce il caricamento asincrono delle scene e si occupa anche di migliorare l'esperienza utente con piccole animazioni di transizione. In particolare, viene inizializzata da un **GameObject** posto nella scena del menu principale e viene attivata quando l'utente sceglie di avviare una partita o cambiare scenario.

La sua funzione principale è gestire il passaggio tra una scena e l'altra. Prima di effettuare il caricamento effettivo della scena successiva, **CaricaScena** esegue un'animazione che sposta la telecamera in avanti per un periodo di tempo specificato tramite il parametro durata, impostato a un valore standard di 3 secondi. Questa animazione è controllata tramite il parametro cameraTransform, che rappresenta il Transform della camera da animare. Una volta completata l'animazione, la scena specificata dal parametro nomeScena viene caricata in modo asincrono.

Musica

Il **singleton Musica** si occupa della gestione della colonna sonora del gioco. Utilizza il componente **AudioSource** della scena del menu principale per riprodurre la musica di sottofondo. Questo singleton garantisce che la musica sia continua attraverso le varie schermate e scenari del gioco. L'utente può mettere in mute la musica premendo il tasto **M** sulla tastiera durante il gioco. Inoltre, l'app companion offre un'ulteriore modalità per controllare la musica, permettendo di disattivarla durante la partita stessa (tuttavia, questa funzionalità è limitata solo alle fasi di gioco, mentre il controllo via tastiera è sempre disponibile).

RacchettaManager

Il **singleton RacchettaManager** gestisce la connessione e la comunicazione con l'app companion. Questo singleton è attivo sin dalla scena iniziale del gioco, permettendo all'app companion di connettersi non appena disponibile e di stabilire una comunicazione continua con il gioco. Una delle funzionalità principali di **RacchettaManager** è la sua capacità di ripristinare automaticamente un collegamento interrotto, garantendo così che l'interazione tra l'app e il gioco non venga compromessa.

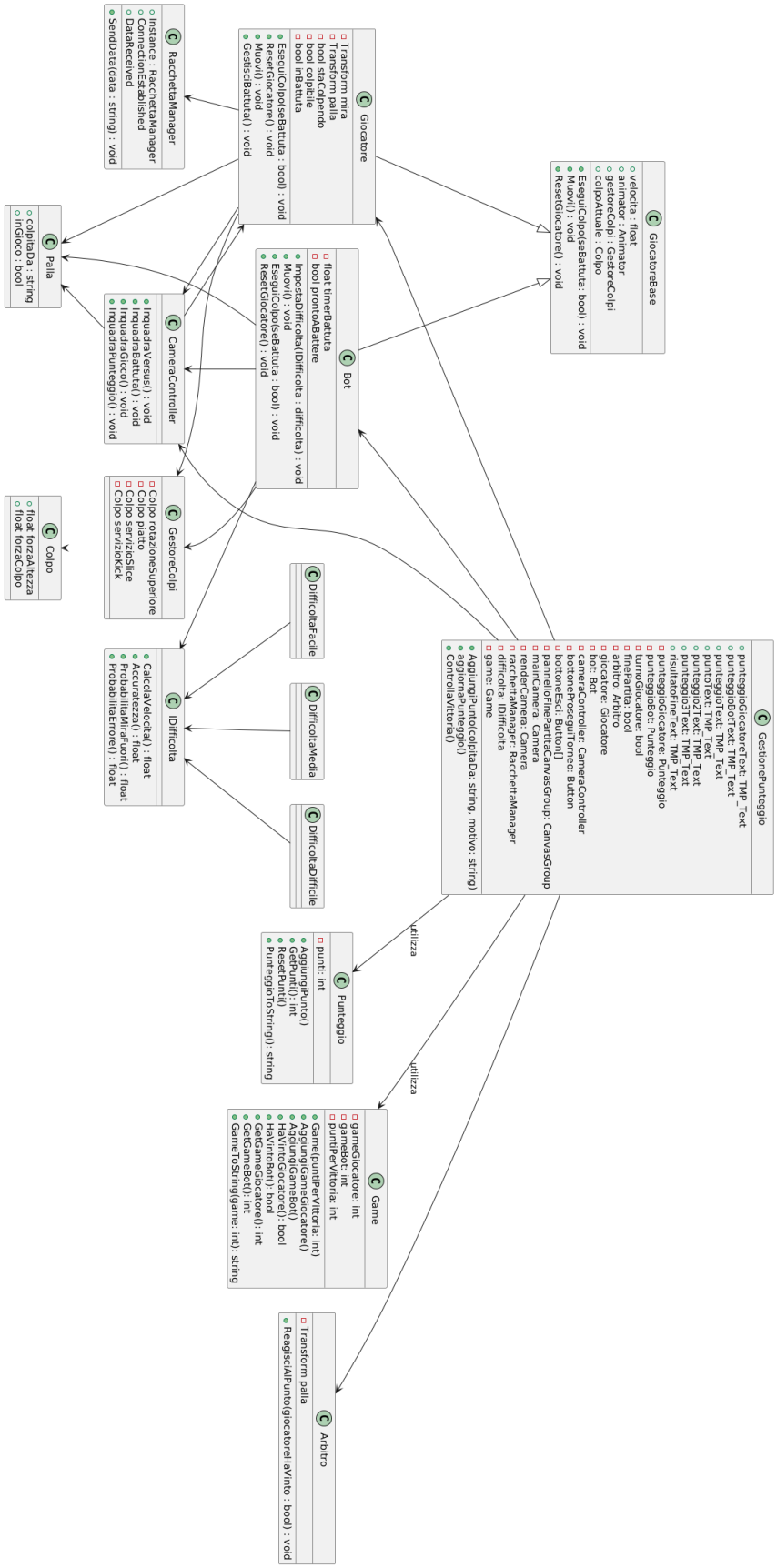
Quando l'app companion invia un comando, questo viene gestito come un'azione che viene passata a un **thread separato**. **RacchettaManager** utilizza una **coda di azioni sincronizzata** per processare i comandi, sfruttando le operazioni di lock e unlock per evitare conflitti o esecuzioni non corrette.

Update, essendo chiamata ad ogni frame del gioco, controlla se ci sono comandi in attesa di essere processati e, in caso affermativo, li esegue con l'uso della funzione Invoke.

Infine, la funzione DataReceived è responsabile di ricevere e decodificare i dati dall'app companion, trasformandoli in stringhe che vengono poi processate da uno degli script attivi nel gioco (Menu, SceltaGiocatore, SceltaPartita, Giocatore).

Il gioco

Il gioco in sé per sé si compone di diversi script, ciascuno dei quali gestisce il giocatore, l'avversario, i colpi, il punteggio (i game, le partite), la logica dell'arbitro, la difficoltà e la camera.



La classe principale che gestisce la logica del gioco, inclusa l'inquadratura della camera, il punteggio, la gestione delle pause, e la vittoria o sconfitta, è **GestionePunteggio**.

Gli input da tastiera e il movimento del giocatore sono invece gestiti dalla classe **Giocatore**, mentre la classe **Bot** regola la difficoltà e risponde alle battute del giocatore.

La classe **Palla** svolge il ruolo di arbitro, assegnando i punti in base alle collisioni. **Arbitro**, al contrario, è utilizzata per scopi estetici, ruotando l'arbitro verso la palla e gestendo le animazioni relative ai punti.

GestoreColpi è associato ai gameobject di giocatore e bot per regolare i colpi direttamente dal motore di gioco.

Infine, **CameraController** controlla le inquadrature, ricevendo input da **GestionePunteggio** e gestendo la visuale in prima persona segnalata dallo script **Giocatore**.

Alcune classi nel progetto sono state generalizzate attraverso l'uso di pattern come lo **Strategy** e l'impiego di **classi astratte**.

L'interfaccia **"Idifficolta"** è implementata dalle classi **"DifficoltaFacile"**, **"DifficoltaMedia"** e **"DifficoltaDifficile"**, che restituiscono i valori relativi alla probabilità di errore del bot, alla sua accuratezza nei movimenti e alla sua capacità di mirare correttamente. Questi parametri influenzano la probabilità che il bot sbagli il colpo, manchi la palla o la tiri fuori dal campo. L'interfaccia viene inizializzata nella classe **GestionePunteggio** dopo aver letto la variabile **"bot-attuale"** dalle **PlayerPrefs**, e successivamente i valori sono comunicati alla classe **Bot** tramite la funzione **ImpostaDifficolta** all'inizio della partita, momento in cui viene mostrato anche il nome dell'avversario.

Le difficoltà impostate prevedono per la modalità **facile**: una velocità di 30, una probabilità di errore del 30%, una probabilità di mirare fuori del 30% e un'accuratezza nei movimenti del 100%. Per la modalità **media**: una velocità di 35, un 20% di probabilità di errore e di tirare fuori, e un'accuratezza del 95%. Infine, la modalità **difficile** assegna al bot una velocità di 40, una probabilità di errore del 10%, una probabilità di tirare fuori del 10% e un'accuratezza di movimento del 90%.

Le classi **Punteggio** e **Game** sono implementate da **GestionePunteggio** per determinare i game necessari per la vittoria e per salvare i punteggi di bot e giocatore. Poiché **Giocatore** e **Bot** condividono alcuni parametri, come il **gestoreColpi**, l'**animator**, e variabili per la velocità, il colpo, il movimento e il reset (utilizzato per la battuta), si è scelto di usare una classe astratta, **GiocatoreBase**, da cui derivano entrambe.

Il **CameraController** gestisce le inquadrature del gioco, richiamate da **GestionePunteggio**, **Giocatore** e **Bot**. Durante lo switch tra prima e terza persona e durante i colpi, vengono utilizzati i metodi **lerp** e **slerp** nell'**Update**, per interpolare in modo fluido i valori di transizione tra le inquadrature. **Lerp** viene impiegato per le interpolazioni lineari, mentre **Slerp** per le rotazioni della camera.

La classe Giocatore

La classe **Giocatore** svolge tre funzioni principali. Prima di tutto, gestisce automaticamente il movimento e la mira del giocatore. In particolare, l'utente si concentra solo nel muovere la racchetta, mentre la direzione del giocatore viene gestita dal sistema. In secondo luogo, la classe gestisce gli input sia dalla tastiera che dall'app companion, sfruttando la comunicazione tcp/ip implementata dalla classe **RacchettaManager**. Infine, si occupa della gestione dei colpi.

I colpi possono essere controllati tramite i tasti **F**, **E**, **R** e **T**, che rispettivamente eseguono una rotazione superiore (colpo forte), un piatto (colpo debole), una battuta kick (servizio forte) e una battuta slice (servizio debole). Inoltre, il tasto **ESC** permette di mettere in pausa il gioco, un'opzione disponibile

anche tramite l'app companion. Il tasto **V**, invece, consente di passare dalla visuale in prima persona a quella in terza persona, una funzione replicabile anche attraverso l'app companion.

Il movimento del giocatore è gestito orizzontalmente e avviene solo se l'utente non sta colpendo la palla. La precisione è quindi fondamentale. La funzione **Muovi** calcola la posizione futura della palla utilizzando la funzione **CalcolaPosizioneFuturaPalla** e tiene conto dei limiti laterali del campo. La posizione futura della palla viene calcolata attraverso una formula che considera il tempo stimato di atterraggio della palla e la sua velocità attuale:

*Vector3 posizioneFutura = palla.position + velocitaPalla * tempo;*

Il movimento del giocatore è quindi confrontato con questa posizione futura. Se la palla è vicina, il giocatore si ferma; inoltre, il movimento è limitato dai **boxcollider** sui lati del campo per evitare che il giocatore esca fuori dai limiti, anche quando la palla esce dal campo. Questa gestione dei limiti è effettuata tramite la funzione **Mathf.Clamp**, che restituisce un valore compreso tra i limiti massimi e minimi consentiti.

Per quanto riguarda la battuta, i **boxcollider** sono responsabili di attribuire alla palla le proprietà di rimbalzo, direzione e velocità, in base alla forza del colpo. Questi collider rimangono inattivi fino al momento del colpo, e la posizione per la battuta viene gestita da due gameObject tra cui il giocatore si alterna.

La direzione della risposta della palla è controllata dal gameObject **mira**, che si posiziona a sinistra o a destra nel campo del bot, in base alla direzione della risposta del giocatore. Inoltre, è presente un componente **audiosource** che riproduce i suoni relativi ai colpi andati a buon fine o mancati.

La classe Bot

La classe **Bot**, similmente al **Giocatore**, gestisce il movimento automatico in base alla direzione e alla velocità della palla. Tuttavia, per il bot è stato implementato un sistema di gestione delle decisioni per i colpi e gli errori, evitando che prenda sempre la palla. A differenza del giocatore, che ha una mira più semplice, il bot può scegliere tra tre possibili punti di mira (mire), selezionandone uno in modo casuale, e decide anche se effettuare un colpo forte o debole, sempre randomicamente.

La precisione del bot nei colpi è influenzata da tre parametri principali.

1. **Accuratezza**: determina quanto bene il bot segue la palla. Un valore più alto indica una maggiore precisione. La mira viene calcolata nel metodo **Muovi**, dove la posizione della mira viene moltiplicata per il valore di accuratezza:

   posizioneMira.x = palla.position.x * accuratezza;

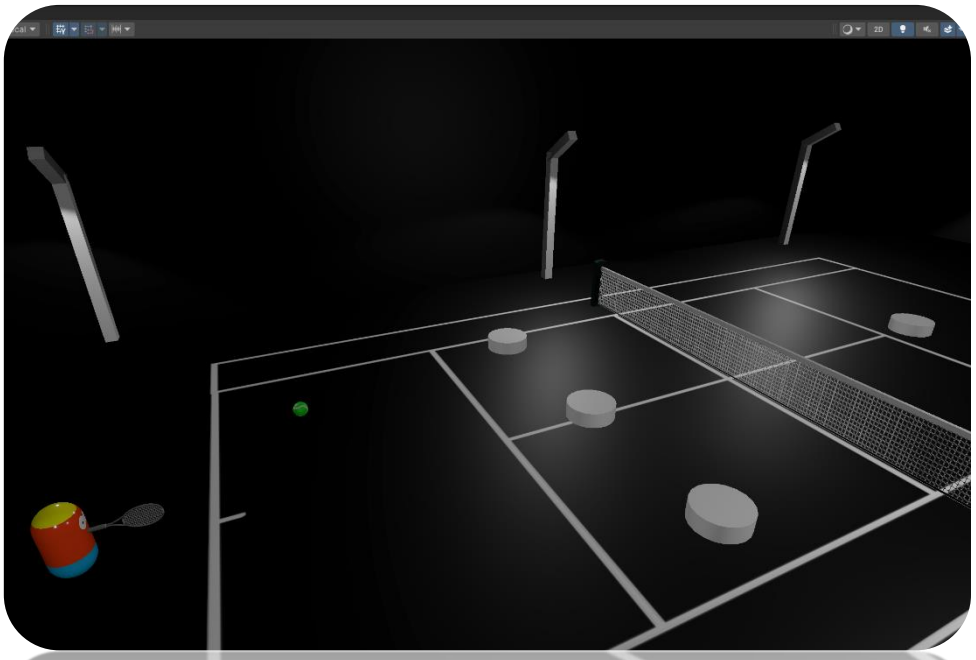
2. **Probabilità di mirare fuori**: rappresenta la possibilità che il bot miri fuori dal campo o sulla rete. Questo viene gestito nel metodo **ScegliMira**, dove un numero casuale viene confrontato con la probabilità di sbagliare mira. Se è inferiore a tale probabilità, la mira viene impostata fuori dai limiti del campo:

   ```
   if (Random.value < probabilitaMiraFuori)
     return new Vector3(limiteDestroCampo.position.x + 1f, mire[randomico].position.y, mire[randomico].position.z);
   ```

3. **Probabilità di errore**: rappresenta la probabilità che il bot fallisca nel colpire la palla. Questo viene gestito nel metodo **OnTriggerEnter**, dove, al momento del tentativo di colpire la palla, viene generato un numero casuale. Se è inferiore alla probabilità di errore, il bot fallisce nel colpo:

   ```
   if (Random.value < probabilitaErrore)
   {
     Debug.Log("Il bot ha mancato la palla!");
     RiproduciSuono(false);
     return;
   }
   ```

La battuta, sia per il giocatore che per il bot, viene gestita al momento del **Reset**, che avviene quando un punto viene assegnato.



La classe Palla

La classe **Palla** gestisce il comportamento della palla durante il gioco, inclusi i movimenti, le collisioni e l'assegnazione dei punti. Il metodo **ImpostaPosizioneBattuta** viene utilizzato per preparare la palla alla battuta, disabilitando temporaneamente la gravità e resettando i rimbalzi. Le collisioni sono gestite principalmente tramite i metodi **OnCollisionEnter** e **OnTriggerEnter**.

- **OnCollisionEnter** si attiva quando la palla collide con oggetti fisici come il campo o i muri (che hanno box collider). Se la palla rimbalza più di una volta sul campo, viene chiamato il metodo **TroppiRimbalzi**, assegnando il punto all'avversario. Se viene rilevata una condizione di "fuoricampo", il punto viene dato all'avversario.
- **OnTriggerEnter** gestisce le collisioni con i trigger, come la rete o i limiti del campo, e assegna i punti in maniera simile.

Il punteggio è gestito dalla classe **GestionePunteggio**, che interrompe la partita per mostrare le reazioni dell'arbitro e dell'avversario e visualizza un testo con una frase motivazionale in base all'esito del punto.

L’app Companion

L'app **Companion** è una semplice applicazione a singola schermata pensata per gestire il movimento tramite i sensori del telefono e interagire con il gioco tramite una connessione TCP/IP.

Struttura dell'app

- **Tutorial**: Gestisce uno scenario animato che spiega come utilizzare l'app e calibrare i movimenti del dispositivo. L'utente deve posizionare il telefono di fronte a sé, con lo schermo rivolto verso destra, per permettere al giroscopio di rilevare correttamente gli swing (oscillazioni). In caso di disorientamento dei sensori, l'utente può ricalibrare premendo un pulsante dedicato.
- **GestioneSensori**: Questa classe è la più complessa e gestisce principalmente la connessione al server, la traduzione delle rotazioni del giroscopio in dati da inviare, e la visualizzazione di messaggi, trofei o personaggi (a seconda che il giocatore vinca o perda). Sono presenti anche coroutine per impostare timer asincroni che non influenzano il motore di gioco.

La classe utilizza un **thread director** con una coda per gestire le azioni derivanti dalla ricezione dei dati, controllata nel metodo **Update**. La coda consente di eseguire diverse azioni, come:

- Mostrare o nascondere pannelli di pulsanti (per la pausa, per la selezione, o per conferme).
- Aggiornare elementi dell'interfaccia, come cambiare il testo di un pulsante (ad esempio, quello per la visuale in prima persona).
- Gestire la calibrazione, che è obbligatoria al primo avvio.
- Riconoscere i gesti tramite i sensori del telefono.

Riconoscimento dei gesti

Nella classe GestioneSensori, il riconoscimento dei gesti avviene monitorando costantemente i dati del giroscopio del dispositivo per identificare uno swing. Il processo è suddiviso in due fasi principali: **Preparazione della Battuta** e **Riconoscimento dello Swing**.

<u>Preparazione della Battuta</u>

Questa fase rileva i movimenti preparatori del giocatore, ignorando quelli non significativi.

1. **Rilevamento del Movimento Laterale**:

   Viene calcolata la differenza di rotazione (deltaRotation) tra i dati attuali e quelli precedenti del giroscopio.

   Se il giocatore è in fase di battuta (isServing == true), si verifica se il movimento è principalmente laterale, osservando se:

   - La variazione sull'asse Y (deltaRotation.y) supera una soglia predefinita (servePreparationThreshold).
   - La variazione sull'asse X (deltaRotation.x) è inferiore a una soglia (serveSwingThreshold).

   Se entrambe le condizioni sono soddisfatte, il movimento viene considerato come preparatorio e ignorato temporaneamente.

2. **Rilevamento del Movimento di Battuta**:

   Se la variazione lungo l'asse X supera la soglia di battuta (serveSwingThreshold), il sistema riconosce il movimento di battuta e imposta isServing = false per essere pronto a rilevare lo swing.

Riconoscimento dello Swing

Questa fase misura l'ampiezza e la velocità del movimento.

1. **Inizio dello Swing**:

   Viene calcolata la velocità angolare dividendo la magnitudine della differenza di rotazione per il tempo trascorso.

   Se la velocità angolare supera una soglia minima (angularVelocityThreshold) e non è già in corso uno swing, il sistema salva la rotazione iniziale (startSwingRotation) e imposta isSwingInProgress = true.

2. **Monitoraggio dello Swing**:

   Durante lo swing, il sistema calcola la distanza angolare percorsa, confrontando i dati attuali con la rotazione iniziale.

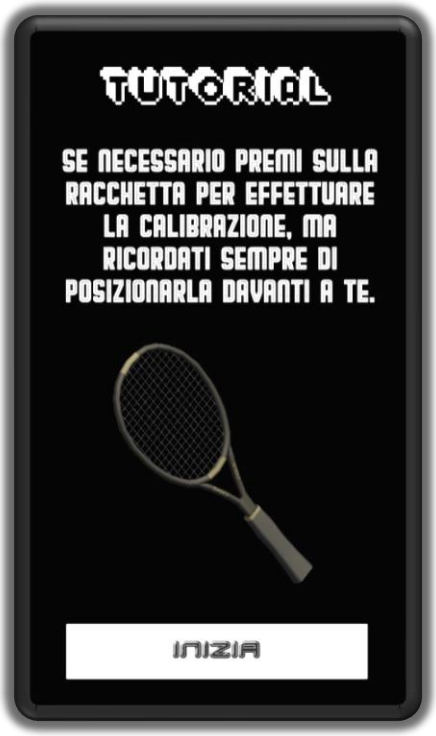
   Se la distanza supera un valore minimo, il sistema completa il riconoscimento dello swing inviando dati al server o aggiornando lo stato del gioco.

La classe utilizza una coroutine (SendPingEvery5Seconds) per verificare regolarmente la connessione al server inviando un ping. Se il ping riesce, il gioco continua, altrimenti viene chiamata la funzione DisconnectFromServer, che interrompe il collegamento e impedisce ulteriori interazioni fino al ripristino della connessione.

Oltre al riconoscimento dei gesti, GestioneSensori gestisce anche:

- **Animazioni**: Riproduzione di effetti grafici o suoni quando viene rilevato uno swing o in caso di vittoria/sconfitta.
- **Illuminazione della scena**: Creazione di effetti di luce per enfatizzare eventi come l'assegnazione di un trofeo o la celebrazione di una vittoria.

Galleria immagini



Riferimenti

1. **THE DFAULTS by fergicide (itch.io)**
   Si tratta di un'animazione di capsule Unity predefinite, sviluppata da "fergicide" su itch.io.
   Link

2. **I2TextAnimation**
   Utilizzata per l'animazione del testo in Unity utilizzando il pacchetto I2 Text Animation.
   Link

3. **Fontget**
   Piattaforma per scaricare font gratuiti per vari utilizzi.
   Link

4. **Trophy Cups/Chalices FREE (Unity Asset Store)**
   Un pacchetto gratuito di trofei 3D utilizzabile nei progetti Unity.
   Link

5. **Soundsnap - Tennis Sound Effects**
   I suoni di rimbalzo della palla e selezione nei menù provengono da qui.
   Link

6. **Bensound**
   Musica royalty-free disponibile per video e progetti creativi.
   Link