**RaspTank**

Report per il progetto di Sistemi Digitali

Gruppo Leonardo Lembo, Luca Lombardi, Alessio Troffei

**Link progetto GitHub:**

<https://github.com/XANA-Hub/ProgettoTT>

**Indice**

* **Introduzione**
  + Requisiti
  + Divisione del core del progetto ed estensioni
  + Divisione del lavoro
* **Lato Server**
  + Panoramica
  + Hardware
  + Scelte Progettuali
    - Tecnologia scelta
    - Schema degli attori
    - Protocollo di comunicazione
  + IA
* **Lato Client**
  + Panoramica
  + Unity
  + Applicazione
    - Menu principale
    - Schermata di controllo del robot
    - Schermata di battaglia
  + Dettagli tecnici
    - Entità
    - Manager
    - Salvataggio e caricamento dei dati
    - Descrizione comandi

**Introduzione**

Il progetto consiste in un robot dotato di quattro ruote, di cui due sono motrici, che può essere controllato da remoto tramite un’applicazione client installata su un dispositivo Android. Questa app fornisce un’interfaccia user-friendly e permette di inviare i comandi in tempo reale.

Oltre al controllo di base, il robot è dotato di una fotocamera atta a riconoscere i vari pezzi degli scacchi, una volta che ne viene riconosciuto uno, verrà mostrata una schermata di un gioco di battaglia a turni dove il giocatore dovrà sconfiggere il mostro che è stato scelto in base al pezzo degli scacchi riconosciuto.

Nel corso di questa relazione, esamineremo in dettaglio il design e lo sviluppo del robot e dell’app, comprese le scelte di hardware e software insieme alle sfide incontrate durante il processo di sviluppo.

**Requisiti**

* Robot controllato remotamente tramite dispositivo esterno presente localmente.
* Il robot deve avere capacità di acquisizione e trasmissione video.
* Capacità di riconoscimento di oggetti specifici su richiesta dell'utente. La risposta dovrà essere restituita all'utente.
* Riconosciuto il nemico deve essere avviato un minigioco.
* La pinza del robot mostrerà fisicamente la buona riuscita di un attacco.

**Divisione del core del progetto ed estensioni**

* **Progetto base:** Requisito 1 e 2.
* **Estensione:** Requisito 3, 4 e 5.

**Divisione del lavoro**

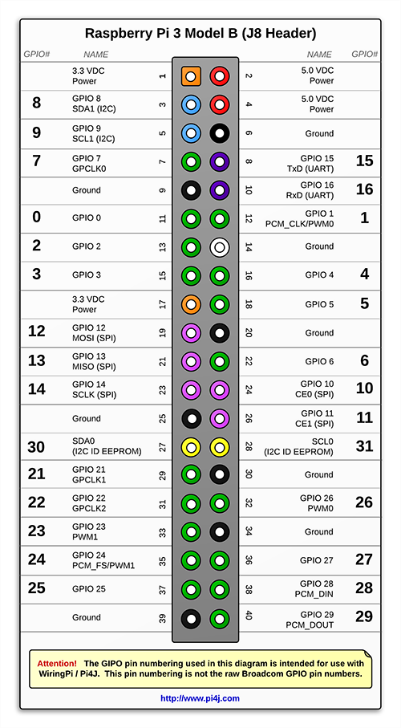
|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint** | **Goal** | **Tempo Stimato** | **Divisione del Lavoro** |
| **SPRINT 1** | Robot con funzionalità di base: movimento ruote e braccio meccanico | 2 man-days | Lembo, Lombardi |
|  | Console Remota | 1 man-day | Troffei |
| **SPRINT 2** | Invio Stream video lato server | 1 man-day | Lembo, Lombardi |
|  | Ricezione video lato client | 1 man-day | Troffei |
| **SPRINT 3** | Riconoscitore AI | 1 man-week | Lembo, Lombardi |
|  | Minigioco | 3 man-days | Troffei |

**Lato Server**

**Hardware**

L’Hardware utilizzato è il seguente:

* [Free](https://github.com/Freenove/Freenove_Tank_Robot_Kit_for_Raspberry_Pi#freenove-tank-robot-kit-for-raspberry-pi)nove Tank Robot Kit for Raspberry Pi.
* Raspberry Pi 3B.
* 2x motoriduttori.
* 2x servo motori che compongono un braccio posto al di sopra dello chassis.
* Raspberry Camera rev v1.3.
* Driver motore.



Schema dei pin del Raspberry 3B

**GPIO utilzzati**

Per comunicare con le varie componentistiche, il Raspberry Pi 3B mette a disposizione 40 GPIO (General Purpose Input Output) pins. Il driver motore utilizzato mette ne mette a disposizioni diversi, tra i quali:

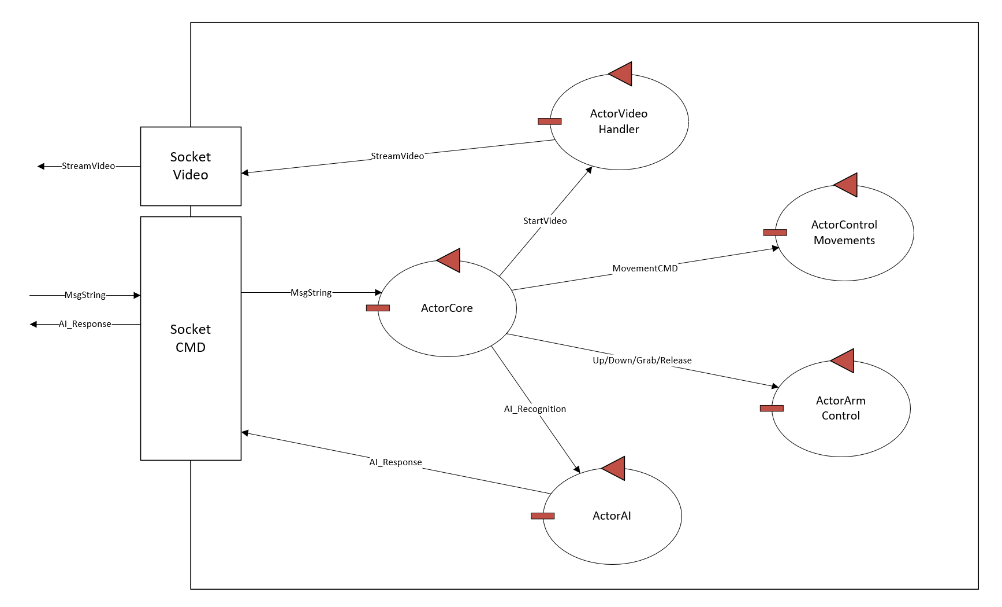
* 4 per i due motoriduttori:
  + GPIO 23 per la componente positiva del motore sinistro.
  + GPIO 24 per la componente negativa del motore sinistro.
  + GPIO 5 per la componente positiva del motore destro.
  + GPIO 6 per la componente negativa del motore destro.
* 2 per i due servo motori:
  + GPIO 7 per controllare il braccio.
  + GPIO 8 per controllare la pinza meccanica.

**Scelte Progettuali**

Uno degli aspetti fondamentali del nostro progetto è stato il processo decisionale relativo all'infrastruttura hardware e software da utilizzare. Sono stati considerati vari aspetti per garantire un'implementazione leggera e funzionale del sistema. In questo contesto, due delle scelte più cruciali sono state la selezione dell’architettura software e la compatibilità con la fotocamera.

Abbiamo optato per Raspberry Pi OS (Legacy, 64 bit) come base del nostro ambiente software. Questa decisione è stata influenzata da diverse considerazioni, in particolar modo la retrocompatibilità con applicazioni e librerie che potrebbero risultare fondamentali per il funzionamento del robot e per l'integrazione con la fotocamera.

Per quanto riguarda la struttura software, il robot è composto da più componenti distinti (ruote, braccio, fotocamera, riconoscitore AI, gestore della socket) che operano parallelamente. Abbiamo mantenuto questa divisione anche nella progettazione del codice, definendo una serie di componenti autonomi che comunicano e collaborano. Per facilitare il tutto usiamo il modello dell’attore[[1]](#footnote-2) per poter coordinare con facilità l’intera architettura di progetto. Unica eccezione data dalla socket che viene gestita da un componente sottostante per nascondere la logica di connessione al resto del sistema.

Elenco degli attori implementati:

* **ActorCore**

(gestisce tutto il sistema)

* **ActorControlArm**

(gestisce il braccio robotico)

* **ActorControlMovements**

(gestisce le ruote)

* **ActorVideoHandler**

(gestisce la telecamera)

* **ActorAi**

(gestisce l’AI)

Abbiamo implementato su Python il modello dell’attore attraverso la libreria PYKKA. Questo ci ha permesso di introdurre semplici regole per controllare la condivisione dello stato e la cooperazione tra le unità di esecuzione anche concorrenti.

La definizione degli attori è stata condivisa tramite un file di configurazione comune tra i vari moduli chiamato “ActorConfig”, contenente i riferimenti ai singoli attori istanziati e alla camera, utilizzata sia per il flusso video che per la cattura delle immagini per l’IA.

Come encoder video la scelta è ricaduta sul “JpegEncoder” data la facile implementazione e le buone prestazioni. Tutto implementato tramite la libreria picamera2[[2]](#footnote-3).

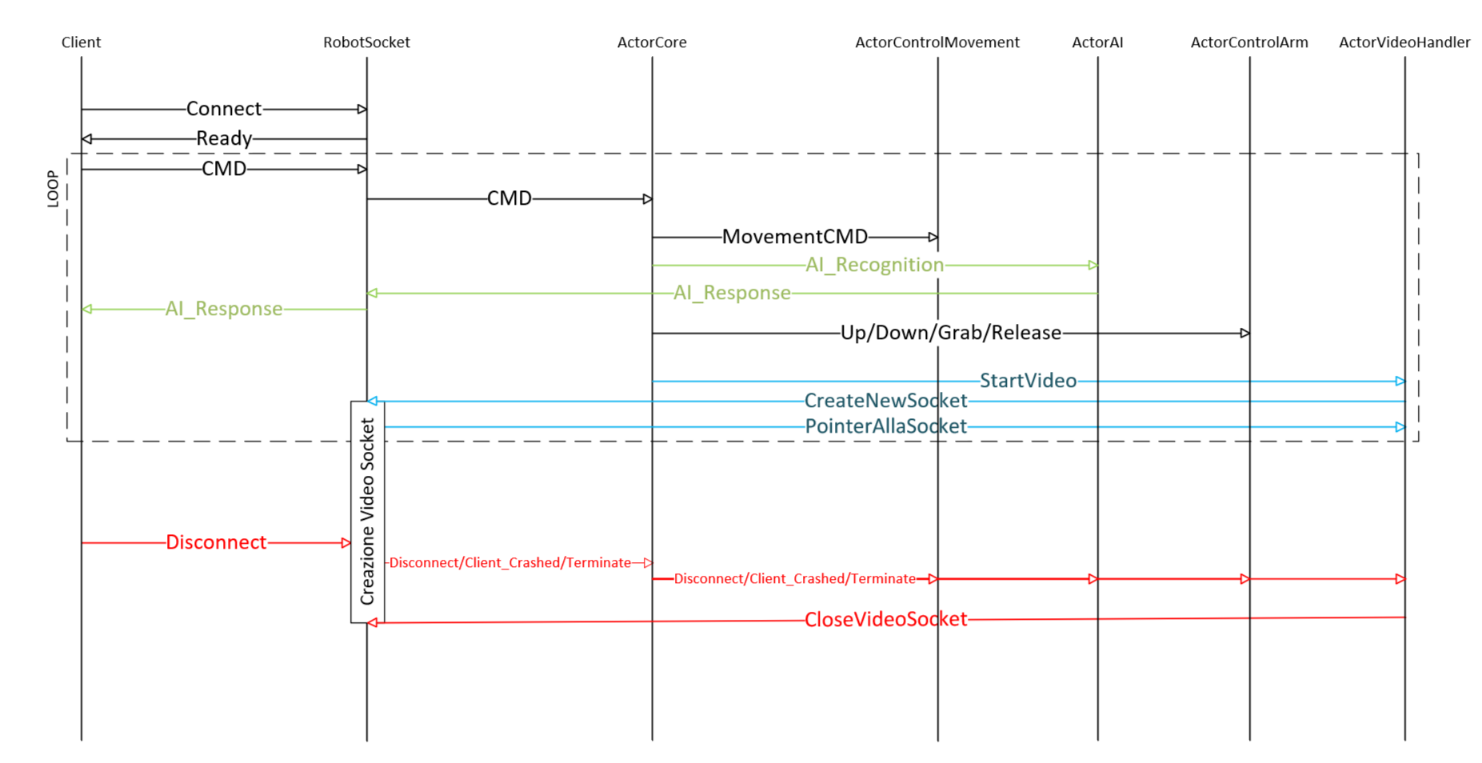
I motoriduttori e i servo motori sono stati controllati tramite segnali PWM (Pulse Width Modulation) implementati utilizzando la libreria di sistema pigpio [[3]](#footnote-4). Abbiamo inoltre implementata la possibilità di regolare la velocità dei motori tramite un’opportuna chiamata di funzione. A tal proposito sono stati analizzati esempi presenti nelle librerie di esempio del kit utilizzato[[4]](#footnote-5)

**Protocollo di Comunicazione:**

La comunicazione tra Client e Robot avviene attraverso i seguenti messaggi:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Sender** | **Receiver** | **Type of Signal** | **Body** |
| Movement | Client | Robot | Dispatch | String |
| Robot\_Arm | Client | Robot | Dispatch | String |
| AI\_Recognition | Client | Robot | Req/Resp | String |
| StartVideo | Client | Robot | Dispatch | String |
| Disconnect | Client | Robot | Dispatch | None |
| Connect | Client | Robot | Req/Resp | None |

La comunicazione tra i due avviene seguendo il seguente modello:



Protocollo di scambio dei messaggi lato Robot

***req/resp:*** *indica un messaggio DI cui mi aspetto una risposta, solitamente bloccante ma non necessariamente.*

***dispatch:*** *INDICA un messaggio per un componente specifico che conosco e Di CUi NON MI INTERESSA la risposta.*

*NOTA: TIPI DI SEGNALI*

Descrizione completa dei comandi: [qui](#Descrizione_Comandi).

I**A**

Dato il basso potere computazionale a nostra disposizione, la scelta del framework è ricaduta su TensorFlow Lite. Quest’ultimo ha una **notevole efficienza computazionale** e dei **tempi di risposta molto bassi** che ci hanno permesso di ottenere delle buone prestazioni anche utilizzando le risorse limitate del Raspberry.

L’allenamento del modello è stato fatto con Tensorflow utilizzando il dataset Chessman[[5]](#footnote-6) e successivamente convertito in TensorFlow Lite grazie alla libreria Tflite\_runtime[[6]](#footnote-7):

tflite\_model.tflite = tf.lite.TFLiteConverter.from\_keras\_model(tf\_model.h5)

L’allenamento del modello in TensorFlow avviene mediante il metodo compile() che richiede 3 parametri:

* **Loss function:** funzione che calcola la differenza tra l’output predetto e quello reale;
* **La funzione di ottimizzazione:** con lo scopo di minimizzare la loss function;
* **La metrica di valutazione:** per mostrare le performance del training;

model.compile(optimizer='Adam', loss='categorical\_crossentropy', metrics=['accuracy'])

Come loss function è stata scelta la cross entropy, solitamente utilizzata per problemi di classificazione come il nostro, mentre come ottimizzatore è stato utilizzato Adam (Adaptive Moment Estimation), generalmente considerato uno dei migliori e computazionalmente efficiente.

La rete neurale è stata invece utilizzata out-of-the-box, composta da un modello sequenziale a 14 layers. Inoltre, per l’allenamento abbiamo sfruttato tecniche di data augmentation, per aumentare la varietà di immagini usate durante la fase di fitting. Le trasformazioni applicate spaziano dalla semplice normalizzazione dei dati, a operazioni rotazionali e di zoom sul dataset di partenza.

**from** tensorflow.keras.preprocessing.image **import** ImageDataGenerator  
train\_datagen = ImageDataGenerator(rescale = 1/255.0, rotation\_range = 30, zoom\_range = 0.4, horizontal\_flip=True, shear\_range=0.4)

**A graph of a graph

Description automatically generated with medium confidence**

Andamento delle metriche durante il training

**Lato Client**

Il lato client consiste in un’applicazione sviluppata con Unity, che **permette all’utente di controllare il robot** tramite un’apposita interfaccia e di un **minigioco di battaglia a turni** dove il giocatore potrà “combattere” contro un nemico scelto in base al pezzo degli scacchi riconosciuto dall’AI.

Si è scelto **Unity** come ambiente di sviluppo dell’app; si tratta di un software ampiamente riconosciuto e utilizzato nell’industria dello sviluppo di videogiochi, ma le sue capacità si estendono ben oltre il mero gioco, permettendo di sviluppare applicazioni complesse per un elevatissimo numero di piattaforme. Come sistema operativo per il deployment si è scelto **Android** in quanto si tratta del sistema mobile più diffuso al mondo.

**Unity**

Questi sono i principali aspetti che hanno guidato la scelta di Unity come software di sviluppo:

* **Versatilità e flessibilità:** Unity è in grado di gestire una vasta gamma di asset, tra cui 2D e 3D, audio, video e molto altro. Ha anche un Asset Store dedicato con una ampia varietà di risorse pronte all’uso.
* **Cross-Platform:** Unity offre la possibilità di pubblicare l'applicazione su diverse piattaforme, inclusi dispositivi iOS, Android, PC, console e molto altro. Questo permette di creare applicazioni che possono essere facilmente distribuite senza preoccuparsi di complessi processi di porting.
* **Facilità di gestione degli oggetti (detti GameObjects):** In Unity, il GameObject è l’unità di base per rappresentare oggetti all’interno dell’applicazione. Si tratta di un contenitore che può essere utilizzato per rappresentare qualsiasi tipo di oggetto o entità. È uno dei componenti fondamentali di Unity ed è estremamente versatile e flessibile in quanto è possibile “agganciarci” un’elevata quantità di componenti che ne andranno a personalizzare il comportamento.
* **Linguaggio di programmazione C#:** Unity utilizza il linguaggio C#, il quale è noto per la sua chiarezza e leggibilità, rendendo la scrittura del codice un processo accessibile anche per gli sviluppatori meno esperti. Grazie a ciò, si possono quindi creare interazioni complesse in modo semplice ed efficiente.

**Applicazione**

L’applicazione è sviluppata per Android, ed è composta principalmente da:

* Un’interfaccia che permette il controllo del robot e la visualizzazione dell’output della telecamera in tempo reale.
* Un minigioco di battaglia a turni che consente all’utente di “combattere” contro il pezzo degli scacchi riconosciuto dall’IA tramite la telecamera stessa.

È importante far notare che ogni applicazione creata con Unity è suddivisa in “**scene**”. Una scena è **una risorsa che contiene un insieme di GameObject;** queste possono essere usate per creare un menu principale, livelli individuali, schermate specifiche o qualsiasi altra cosa; si può vedere ogni scena come un livello isolato ed in ognuna saranno presenti i GameObject che si desidera avere in quest’ultima, come ad esempio la UI contenente i bottoni di gioco, il giocatore, gli eventuali nemici, ecc… Le scene permettono essenzialmente di progettare e costruire il gioco “a pezzi”, garantendo un’elevata flessibilità di sviluppo.

Nel nostro caso, l’applicazione è suddivisa in tre scene:

* Menu principale.
* Schermata di controllo del robot.
* Schermata con gioco a battaglia a turni.

**Menu principale**

Il menu principale rappresenta la prima schermata mostrata all’utente. Essa è costituita da un’interfaccia con diverse opzioni, ciascuna avente uno scopo ben specifico:

* Immagine che contiene testo, schermata, montagna, grafica

  Descrizione generata automaticamente**Start:** Verrà mostrata una schermata apposita con i comandi di controllo del robot e un riquadro in cui è possibile visualizzare in tempo reale l’output della fotocamera.

Menu principale dell’applicazione

* **Options:** Qui è possibile modificare le impostazioni relative al volume dell’audio di gioco e configurare le opzioni di connessione quali indirizzo IP e la porta sulla quale verrà tentata la connessione remota al robot.
* **Quit:** Verrà terminata l’app.

**Schermata di controllo del robot**

Questa schermata permette all’utente di controllare i movimenti del robot e altri parametri (come la connessione/disconnessione). Di seguito è fornita una panoramica dettagliata delle principali funzionalità:

1. **Comandi per i movimenti del robot**: Nel set di comandi situato a sinistra dello schermo, sono presenti i bottoni per controllare i movimenti del robot. Questi includono azioni come avanti, indietro, sinistra e destra.
2. **Comandi del braccio meccanico**: I comandi sulla parte destra dello schermo sono dedicati al controllo del braccio meccanico del robot. Tramite questi è possibile eseguire operazioni quali movimento in alto o in basso e apertura/chiusura della pinza.
3. **Bottoni per la gestione della connessione**: Nella parte superiore, si trovano i pulsanti per gestire la connessione con il robot. È possibile disconnettersi in qualsiasi momento o riconnettersi nel caso in cui i tentativi di connessione abbiano raggiunto il limite massimo consentito.
4. **Barra vita/esperienza giocatore**: Nella parte superiore a sinistra, è possibile trovare un riquadro con i dettagli relativi alle statistiche attuali del giocatore, quali punti vita, esperienza, e livello.
5. **Schermata di visualizzazione della fotocamera del robot**: Nel centro dello schermo, si trova un componente dedicato alla visualizzazione in tempo reale della fotocamera montata sul robot (in questo caso il colore è completamente nero per far comprendere dove l’output sarà visualizzato).

Immagine che contiene schermata, testo, elettronica, software

Descrizione generata automaticamente

Schermata di controllo del robot

**Schermata di battaglia**

Nell’applicazione si è scelto di programmare un minigioco basato sulla battaglia a turni. Quest’ultimo viene attivato quando l’IA identifica con successo un particolare pezzo degli scacchi. Infatti, in base alla pedina riconosciuta, verrà presentato un nemico diverso contro il quale combattere. Il giocatore potrà scegliere tra quattro opzioni principali, ciascuna delle quali influenzerà il corso dello scontro:

* **Attacco:** Il giocatore attacca il nemico infliggendo dei danni. L’efficacia dell'attacco può variare in base a molti aspetti quali colpo critico, attacco del giocatore, difesa del nemico, ecc…
* **Difesa:** Questa opzione permette al giocatore di ridurre i danni subiti dal prossimo attacco sferrato dal nemico. Anche qui ci sono diversi fattori che rendono il tutto più interessante, come l’attacco del nemico e la difesa del giocatore.
* **Cura:** Il può recuperare punti salute. Questa mossa è però rischiosa in quanto ha un elevato tasso di fallimento.
* **Fuga:** In caso di difficoltà, il giocatore ha la possibilità di tentare la fuga dalla battaglia. Anche qui, l’esito della mossa è dettato da caratteristiche quali i punti velocità del nemico e quelli del giocatore.

Anche il nemico potrà scegliere una delle quattro azioni sopra citate, è stata infatti programmata una semplice IA che si occuperà di andare a scegliere le mosse fatte dai nemici durante lo scontro. I nemici hanno inoltre caratteristiche uniche e diverse tra loro, incoraggiando i giocatori a esplorare diverse combinazioni di azioni.

Immagine che contiene testo, Gioco per PC, Software per videogiochi, Videogioco di strategia

Descrizione generata automaticamente

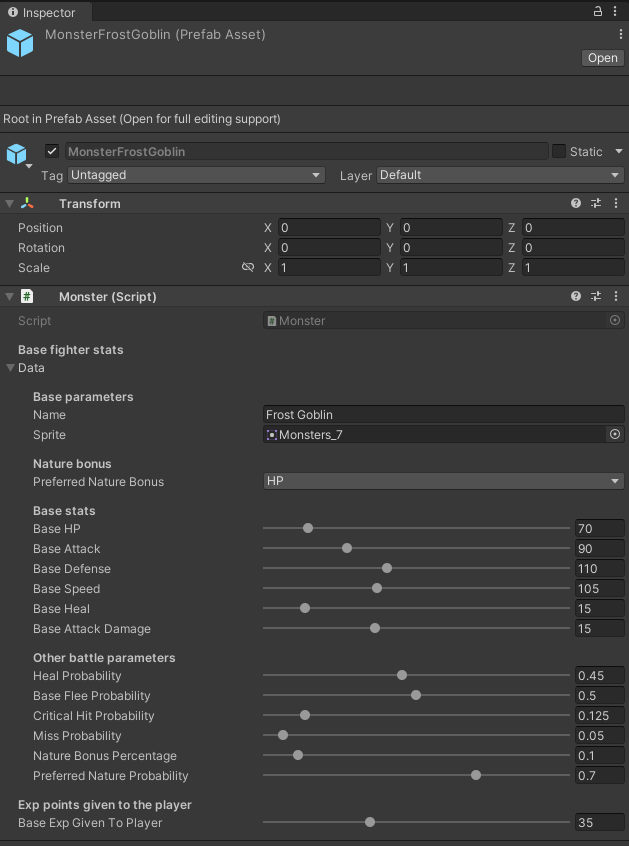
Schermata del minigioco

**Dettagli tecnici**

Di seguito saranno riportati i dettagli tecnici relativi ai vari componenti fondamentali che costituiscono l’applicazione, tra i quali abbiamo:

* **Entità**
* **Manager**
* **Salvataggio e caricamento dei dati**
* **Descrizione segnali**

**Entità**

Tutte le entità (sia giocatore che nemici) del progetto sono “prefab”. In Unity, un prefab, è un oggetto memorizzato persistentemente all’interno del progetto, contenente tutti i suoi componenti e proprietà. Funge da template da cui è possibile creare nuove istanze di GameObject nella scena in cui si sta lavorando; questo risulta migliore rispetto al copia-incolla degli oggetti perché mantiene sincronizzate automaticamente tutte le loro copie; le modifiche apportate a un prefab si riflettono automaticamente in tutte le istanze, semplificando i cambiamenti su larga scala nel progetto.

Esempio di prefab di un nemico del gioco

Con questo approccio, si ha una modifica delle statistiche di combattimento delle entità molto semplificata. Ognuna, infatti, possiede caratteristiche di base come attacco, difesa, velocità, punti di cura e punti di danno dell’attacco. Quest’ultime saranno poi aumentate in base al livello e a un bonus casuale chiamato “bonus natura” scelto all’inizio, rendendole così tutte diverse tra loro. Inoltre, ogni combattente avrà altre caratteristiche, tra cui probabilità di cura, probabilità di fuga, probabilità di colpo critico e probabilità di fallimento dell’attacco le quali consentono una personalizzazione ancora più profonda.

**Manager**

L’applicazione è suddivisa in vari componenti fondamentali detti **“manager”**. Si tratta di GameObject estremamente importanti che utilizzano uno specifico pattern, detto “**singleton**”, il quale impone che **dovrà sempre esserci un’unica istanza di quest’ultimi durante tutta l’esecuzione dell’applicazione senza duplicazioni**. Inoltre, questa istanza non dovrà mai distrutta quando avviene un cambio di scena, infatti, quando ciò accade, Unity va a distruggere ogni oggetto presente nella scena precedente. Il singleton pattern risulta ideale per questo scopo, perché si può controllare che venga creata sempre una singola istanza (statica) della classe e che venga restituita ai richiedenti l’istanza attualmente in uso, senza creare dei duplicati.

Tra i manager più importanti di questa applicazione troviamo:

* **ClientTCPManager:** Questo manager si occupa di andare a gestire la comunicazione con il robot, la quale avverrà tramite socket TCP. Permette infatti di inviare i vari comandi (descritti nella sezione apposita sotto) in maniera affidabile. Questo componente è fondamentale ed ha bisogno di rimanere attivo durante tutta l’esecuzione dell’applicazione.
* **VideoReceiverManager:** Questo manager utilizza invece una socket UDP per la ricezione del flusso video catturato dalla telecamera del robot.
* **BattleManager:** Questo componente si occupa di gestire il minigioco della battaglia a turni Questo componente è logicamente il più complesso poiché deve gestire tutte le interazioni tra il giocatore e i nemici, calcolare i danni inflitti e subiti, monitorare le statistiche e molto altro.

**Salvataggio dei dati**

Nell’applicazione c’è un estensivo uso della classe di Unity chiamata “**PlayerPrefs**” la quale **permette di effettuare dei salvataggi permanenti di dati**. Esistono diversi modi per mantenere i dati su disco, PlayerPrefs è sicuramente il più semplice, anche se delimitata ad un insieme ristretto di tipi di dati, quali float, int e string. C’è da notare che PlayerPrefs non salva i dati su file, infatti la posizione del salvataggio varia da dispositivo a dispositivo, ma la funzionalità a livello di codice è la stessa.

Ad esempio, per salvare un dato di tipo float all’interno delle PlayerPrefs, usiamo:

PlayerPrefs.SetFloat("MasterVolume", volume);

Per invece andarlo a riprendere in un secondo momento utilizziamo:

**float** gameVolume = PlayerPrefs.GetFloat("MasterVolume");

Nel nostro caso, questa classe verrà usata per salvare e caricare alcuni specifici dati di gioco tra cui:

* I dati relativi alle impostazioni di gioco (come indirizzo IP/porta, volume di gioco).
* Quelli relativi alle statistiche del giocatore (punti vita, esperienza, livello).
* Il nemico da incontrare in base al pezzo degli scacchi riconosciuto dall’IA.

Immagine che contiene testo, schermata, Software multimediale, software

Descrizione generata automaticamente

I dati salvati nella classe PlayerPrefs dell'applicazione

**Descrizione segnali**

Il formato dei comandi scambiati tra il robot e l’applicazione client è il seguente:

ID:<number>;TYPE:<command>;BODY:<options>

Dove:

* **ID:** Numero incrementale della richiesta ricevuta da parte del client.
* **TYPE:** Rappresenta il comando possibile tra i seguenti:
  + **Start:** Per avviare il flusso video verso un client.
  + **Movement:** Per controllare il movimento del robot.
  + **Robot\_Arm:** Per controllare il braccio meccanico.
  + **AI\_Recognition:** Per avviare le procedure di riconoscimento tramite AI.
  + **Disconnect:** Per avviare le procedure di disconnessione dal client.
* **BODY:** Contiene le varie opzioni dato il comando che possono essere:
  + **Start:**
    - **“<address>:<port>”** per poter indirizzare il flusso video all’host scelto.
  + **Movement:**
    - **“[Forward|Backward|Rotate\_Right|Rotate\_Left] [Start|Stop]”.**
  + **Robot\_Arm:**
    - **“[Up|Down|Grab|Release]”.**
  + **AI\_recognition:**
    - **“Identify\_Current”.**

Il formato della **AI\_Response** corrisponde ad una stringa tra le seguenti (scelta in base al pezzo degli scacchi riconosciuto):

* **Nothing**
* **Bishop**
* **Knight**
* **Pawn**
* **Queen**
* **Rook**

1. <https://www.brianstorti.com/the-actor-model> [↑](#footnote-ref-2)
2. <https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf> [↑](#footnote-ref-3)
3. <https://abyz.me.uk/rpi/pigpio/python.html> [↑](#footnote-ref-4)
4. 4 <https://github.com/Freenove/Freenove_Tank_Robot_Kit_for_Raspberry_Pi/tree/main/Code/Server> [↑](#footnote-ref-5)
5. <https://www.kaggle.com/datasets/niteshfre/chessman-image-dataset> [↑](#footnote-ref-6)
6. <https://www.tensorflow.org/lite/guide/python> [↑](#footnote-ref-7)