Relazione del Progetto: Starship

Nome: Vincenzo
Cognome: Franchetti
Matricola: 0124002616

Sommario

Descrizione del Progetto

Il progetto Starship è stato sviluppato per dimostrare l'applicazione di concetti di programmazione di rete in un contesto di simulazione interattiva. Il gioco richiede che l'utente, che controlla una navicella spaziale, eviti detriti spaziali (meteoriti) generati da una tempesta. La navicella, che agisce come client, riceve costantemente le posizioni aggiornate dei meteoriti da un server, il quale li genera e li invia tramite pacchetti UDP.

L'obiettivo è implementare una soluzione che utilizzi il protocollo UDP (User Datagram Protocol) per gestire la comunicazione tra client e server, garantendo una trasmissione dati veloce e senza controllo sull'integrità del pacchetto, in quanto si privilegia la rapidità e la reattività, caratteristiche fondamentali in una simulazione di gioco in tempo reale. Durante lo sviluppo del progetto, l'attenzione si è focalizzata sulla gestione delle connessioni di rete, l'uso delle socket e il bilanciamento tra la leggerezza dei pacchetti e la gestione della perdita di dati.

Descrizione dell'Architettura

Architettura Client-Server

L'architettura del progetto si basa sul modello **client-server**, dove il server gestisce la generazione dei meteoriti e il client, controllato dall'utente, si sposta su una griglia evitando le collisioni.

1. **Client**: La navicella spaziale viene controllata dall'utente tramite comandi che aggiornano la sua posizione in tempo reale. Il client riceve periodica-

mente le coordinate dei meteoriti generate dal server e, se la posizione di un meteorite coincide con quella della navicella, viene notificato il rischio di collisione. L'interfaccia utente, implementata in **Pygame**, mostra la navicella e i meteoriti sulla griglia di gioco.

2. **Server**: Il server gestisce la tempesta di meteoriti, generando nuove posizioni per i detriti ogni due secondi e inviandole ai client connessi. La comunicazione tra server e client avviene tramite il protocollo **UDP**, che invia pacchetti senza garantire la consegna o l'ordine, ma con una latenza molto bassa.

Protocollo UDP

Il **UDP** è un protocollo di livello di trasporto leggero, scelto per la sua efficienza in termini di velocità e semplicità. Nel progetto Starship, il server invia continuamente pacchetti contenenti le coordinate dei meteoriti al client tramite **UDP**, aggiornando la griglia di gioco in tempo reale.

Il protocollo **UDP** è essenziale nel contesto di applicazioni in tempo reale come questa, in cui la perdita di un pacchetto non compromette il funzionamento del sistema e la rapidità di risposta è cruciale.

Dettagli Implementativi

Utilizzo delle Socket e Protocollo UDP

La comunicazione tra il client e il server è realizzata tramite **socket**, che permettono ai due nodi di rete di inviare e ricevere pacchetti di dati. In **Python**, l'implementazione delle socket è gestita tramite il modulo **socket**, che consente di creare connessioni basate su **UDP**.

- **Creazione delle Socket**: Il server è configurato per ascoltare sulla porta 9999, pronto a ricevere le richieste del client. Ecco un esempio della configurazione della socket nel server:

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind(('localhost', 9999))  # Configurazione del server
```

Il client crea una socket per inviare comandi e ricevere le coordinate dei meteoriti dal server:

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

- **Scambio di Pacchetti**: Una volta configurata la socket, il client invia i comandi al server e riceve pacchetti **UDP** contenenti le posizioni dei meteoriti:

```
client_socket.sendto(b'start', (HOST, PORT))  # Inizio della sessione
dati, _ = client_socket.recvfrom(1024)
posizione_meteorite = tuple(map(int, dati.decode().split(',')))
```

Nel server, i meteoriti vengono generati casualmente ogni due secondi, e le loro posizioni vengono inviate ai client connessi tramite la funzione **sendto**:

```
server_socket.sendto(f'{posizione[0]},{posizione[1]}'.encode(), addr)
```

Gestione del Movimento della Navicella e delle Collisioni

Il client gestisce il movimento della navicella spaziale su una griglia, aggiornata in tempo reale grazie alle informazioni ricevute dal server. Se la navicella si trova sulla stessa posizione di un meteorite, viene rilevata una collisione:

```
if posizione_navicella == posizione_meteorite:
    print("Collisione! Game Over.")
```

In caso di collisione, il client avvisa l'utente e termina il gioco.

Generazione dei Meteoriti

Nel server, la generazione dei meteoriti è gestita da un thread separato, che ogni due secondi genera nuove coordinate per i meteoriti e le invia ai client connessi:

```
def genera_meteoriti():
    while True:
        x = random.randint(0, 800)
        y = random.randint(0, 600)
        dati_meteorite = f"{x},{y}"
        for client in client_connessi:
            server_socket.sendto(dati_meteorite.encode(), client)
        time.sleep(2)
```

Casi d'Uso

1. **Connessione del Client al Server**: Quando il client si connette al server, invia un comando **start**. Il server lo registra e inizia a inviare pacchetti contenenti le posizioni dei meteoriti.
2. **Generazione dei Meteoriti**: Ogni due secondi, il server genera nuovi meteoriti e invia le loro coordinate ai client. Se un meteorite collide con la navicella, il client riceve una notifica di **Game Over**.
3. **Movimento del Client**: L'utente può muovere la navicella tramite comandi inviati al server. Le posizioni vengono costantemente aggiornate e la griglia di gioco viene sincronizzata tra client e server.

Schema Architetturale

Il client e il server comunicano costantemente utilizzando il protocollo **UDP**. Il server gestisce la generazione dei meteoriti, mentre il client riceve le loro coordinate e reagisce di conseguenza.

```
CLIENT (navicella)
     |
     |
SERVER (meteoriti)
```

Conclusione

Il progetto "**Starship**" ha dimostrato l'importanza della comunicazione in tempo reale in un contesto di rete, evidenziando l'efficienza del protocollo **UDP** nel gestire flussi di dati veloci e senza interruzioni. Le **socket** e la gestione delle connessioni hanno permesso di creare un sistema di comunicazione bidirezionale tra il server, che gestisce le dinamiche della tempesta di meteoriti, e il client, che ne riceve le informazioni in tempo reale. L'architettura **client-server**, combinata con il protocollo **UDP**, ha garantito una latenza minima, essenziale per applicazioni come i giochi interattivi.