Progetto di Sistemi Distribuiti e Pervasivi

Laboratorio di Sistemi Distribuiti e Pervasivi

Aprile 2017

1 Descrizione del progetto

Lo scopo del progetto è quello di realizzare un gioco on-line multiplayer (MMOG, Massive Multiplayer Online Game) semplificato. Il gioco consiste nello spostare un personaggio virtuale all'interno di un'area di gioco, con lo scopo di ottenere punti eliminando un determinato numero di giocatori avversari. Lo spostamento dei giocatori nell'area e tutta la logica di gioco devono essere gestiti in modo totalmente distribuito tramite una apposita rete *peer to peer* formata dai giocatori facenti parte della partita. Un server *REST* ha il compito di gestire le partite in corso. Quando un giocatore si connette al server può decidere se creare una nuova partita o se aggiungersi ad una partita già esistente. Le prossime sezioni di questo documento descriveranno in dettaglio le regole del gioco e le componenti del sistema da implementare.

2 Regole del gioco

2.1 Area del gioco

L'area del gioco, come mostrato in Figura 2.1, è una griglia quadrata di lato N, dove ogni cella rappresenta una possibile posizione di un giocatore nell'area di gioco. Ogni giocatore può quindi spostarsi all'interno della griglia (una cella per volta) in 4 direzioni: su, giù, destra e sinistra. Inoltre, la griglia è logicamente divisa in quattro parti uguali, che chiameremo *aree*: *area verde*, *area rossa*, *area blu* e *area gialla*.

2.2 Creazione della partita

Un giocatore che vuole creare una partita deve specificarne il nome (che deve essere univoco), la lunghezza N del lato della griglia quadrata, e il numero p di punti che è necessario raggiungere per vincere la partita (verrà spiegato più avanti come è possibile acquisirli). Inizialmente, il giocatore che ha creato la partita verrà posizionato in una posizione casuale della griglia e sarà l'unico giocatore presente. Successivamente, i giocatori interessati ad

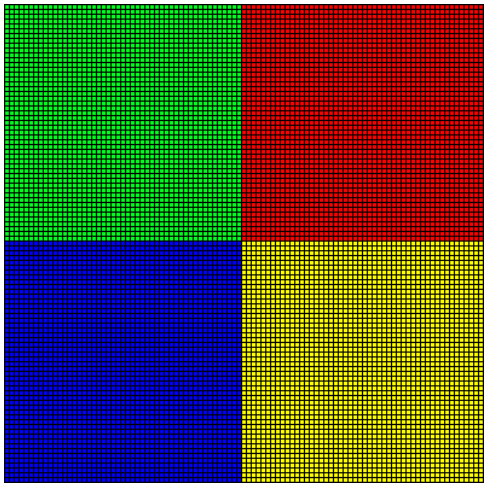

Figura 1: Esempio di area di gioco 100x100

aggiungersi nella griglia verranno aggiunti in essa in una posizione casuale dove però non sono già presenti altri giocatori. Tutti i giocatori iniziano a zero punti.

2.3 Acquisizione punti e vittoria

L'obiettivo principale di ogni giocatore è quello di raggiungere un numero di punti p prima di tutti gli altri giocatori. Una volta che un giocatore ottiene p punti, viene proclamato vincitore e la partita viene conclusa. Può anche succedere che una partita termini senza vincitori (tutti i giocatori vengono eliminati). Ci sono due modi per acquisire punti, che vengono descritti in seguito.

2.3.1 Mangiare l'avversario

Il modo più semplice di acquisire un punto è quello di spostarsi in una posizione della griglia dove è già presente un altro giocatore. Più precisamente, quando un giocatore a si sposta in una posizione in cui è presente un giocatore b (in seguito diremo che "a mangia b"), il giocatore a guadagna un punto, mentre il giocatore b viene eliminato dal gioco.

2.3.2 Bombe

Durante la partita, ogni giocatore virtuale ha con sè un sensore (simulato) che genera continuamente una serie di misurazioni. Queste misurazioni vengono analizzate in background dall'applicazione di gioco per rilevare osservazioni anomale, cha da ora chiameremo *outliers*. Ogni volta che si verifica un *outlier* nelle misurazioni, al giocatore viene messa a disposizione una

bomba utilizzabile solo verso una specifica area a. Ci saranno quindi *bombe rosse*, *bombe gialle*, *bombe verdi* o *bombe blu* che colpiranno l'area del colore corrispondente. La colorazione della bomba si basa sul valore dell'outlier (maggiori dettagli verranno spiegati più avanti). Le bombe vengono messe a disposizione al giocatore in una struttura dati di tipo first-in/first-out. In alternativa al classico movimento, il giocatore può quindi scegliere (sempre da standard input) in qualsiasi momento di lanciare la prima bomba a lui disponibile. Tutti i giocatori nella partita vengono quindi avvisati che una bomba nell'area a è stata lanciata. Dopo 5 secondi, calcolati a partire dal momento in cui a tutti i giocatori è stato reso noto il lancio della bomba, l'applicazione di gioco del giocatore che ha lanciato la bomba manda in automatico a tutti gli altri giocatori un messaggio di esplosione della stessa. Quando la bomba esplode, tutti i giocatori nell'area a (compreso, eventualmente, il lanciatore della bomba) vengono eliminati. Il giocatore che ha lanciato la bomba ottiene tanti punti quanti giocatori eliminati, con un massimo di 3 punti (eccezione fatta se il giocatore viene eliminato dalla sua stessa bomba, in quel caso non guadagnerà nessun punto e verrà eliminato).

3 Specifiche implementative: *peer*

Ogni giocatore esegue una propria applicazione detta *peer*. Un server, che verrà spiegato in dettaglio nella Sezione 4, si occupa di gestire la creazione/rimozione delle partite e di inserire/rimuovere i peer nelle varie partite. Quando l'applicazione *peer* viene fatta partire, questa si connette automaticamente al server di gioco. Tramite il server è quindi possibile entrare in una partita già esistente o crearne una nuova. Una volta dentro la partita, ogni singolo *peer* non ha la visione completa della griglia di gioco, ma solo locale. La griglia, infatti, viene gestita in modo distribuito tra tutti i *peer* partecipanti alla partita. Ogni *peer* memorizza quindi solo la propria posizione, e comunica ogni suo spostamento o lancio di bomba in broadcast a tutti gli altri *peer*. Per la comunicazione in broadcast è richiesto che il *peer* comunichi in parallelo con ogni altro giocatore nella partita con una comunicazione diretta.

3.1 Interazione con l'utente

Quando l'applicazione *peer* viene fatta partire, viene prima di tutto chiesto all'utente l'indirizzo del server di gioco al quale connettersi (IP e numero di porta). Dopodichè viene richiesto di inserire il suo nome utente. Una volta che la connessione è avvenuta con successo, viene mostrata una scelta di operazioni possibili: visualizzare l'elenco delle partite in corso, visualizzare i dettagli di una particolare partita, creare una nuova partita o aggiungersi ad una partita già esistente. Nel caso in cui l'utente voglia creare una nuova

partita, dovrà anche specificarne il nome. All'interno della partita, l'applicazione di gioco resta in ascolto su standard input di ogni mossa del giocatore. Un utente può inserire una mossa alla volta. Le mosse a disposizione sono lo spostamento sulla griglia o il lancio della bomba (se disponibile).

3.2 Spostamento

Nel caso di spostamento, il *peer* deve occuparsi di comunicare le sue nuove coordinate a tutti gli altri *peer*. Quando un peer a riceve un messaggio di spostamento da parte di un peer b, a confronta la propria posizione con quella indicata da b. Se le posizioni coincidono, a segnala a b che ha totalizzato un punto, comunica al server di essere stato eliminato e a termina la propria esecuzione.

3.3 Bomba

3.3.1 Accelerometro

Ogni giocatore virtuale ha con sè un accelerometro dal quale vengono continuamente rilevate misurazioni. Per semplicità considereremo solo i dati di accelerazione (in milli g, dove g = $9.81\frac{m}{s^2}$) provenienti dall'asse X. Le misurazioni dell'accelerometro sono quindi caratterizzate da:

- Valore letto
- Timestamp in millisecondi

La generazione di queste misurazioni viene svolta da un opportuno simulatore, che viene fornito già pronto per semplificare lo svolgimento del progetto. Il simulatore assegna come timestamp alle misurazioni il numero di secondi passati dalla mezzanotte.
Al link *http://ewserver.di.unimi.it/sdp/simulation_src_2017.zip* è quindi possibile trovare il codice necessario. Questo codice andrà aggiunto come package al progetto e NON deve essere modificato. Ogni *peer* dovrà quindi occuparsi di far partire il simulatore non appena viene inserito in una partita. Il simulatore è un thread che consiste in un loop infinito che simula periodicamente (con frequenza predefinita) le misurazioni, inserendole in una opportuna struttura dati. Di questa struttura dati viene fornita solo l'interfaccia (Buffer), la quale espone due metodi:

- $void\ addNewMeasurement(Measurement\ m)$
- $List<Measurement>\ readAllAndClean()$

È dunque necessario creare una classe che implementi l'interfaccia. Ogni *peer* avrà un suo buffer. Il thread di simulazione usa il metodo $addNewMeasurement$

per riempire la struttura dati. Il metodo *readAllAndClean* deve invece essere usato per ottenere tutte le misurazioni contenute nella struttura dati. Al termine di una lettura, *readAllAndClean* deve occuparsi di svuotare il buffer in modo da fare spazio a nuove misurazioni.

3.3.2 Rilevamento outlier

Ogni secondo, il *peer* chiama il metodo *readAllAndClean* sul buffer per prendere tutte le misurazioni prodotte dal sensore. Ottenute queste misurazioni, si vuole calcolarne il valore medio m. Usando il valore medio m_i ottenuto al secondo i, si vuole quindi calcolare l'*Exponential Moving Average* (EMA). L'*EMA* è una media calcolata in tempo reale che attribuisce più peso alle misurazioni recenti. In particolare, la formula per calcolarla al secondo i-esimo è la seguente:

$$EMA_i = EMA_{i-1} + \alpha(m_i - EMA_{i-1})$$

dove $0 < \alpha < 1$ è un parametro che verrà scelto in fase di sviluppo. Un *outlier* viene quindi identificato quando si verifica che:

$$EMA_i - EMA_{i-1} > th$$

dove th è una soglia sempre scelta in fase di sviluppo. Infine, per scegliere a quale area è diretta la bomba, viene calcolato il resto intero della divisione per 4, ovvero:

$$\lceil EMA_i \rceil \mod 4$$

Se il modulo è 0 la bomba sarà direzionata verso l'*area verde*, se 1 verso l'*area rossa*, se 2 verso l'*area blu* e se 3 verso l'*area gialla*.

3.3.3 Lancio bomba

Una volta rilevato un outlier, il giocatore ha a disposizione una bomba verso una particolare area i. Se il giocatore decide di usarla, il *peer* manda a tutti gli altri *peer* la comunicazione che una bomba è stata posizionata nell'area i. Dopo 5 secondi, il *peer* manda un'ulteriore messaggio a tutti gli altri *peer* per comunicare che la bomba è esplosa. Quando un peer a si trova in un'area i e riceve un messaggio da b di bomba esplosa in i, comunica a b che ha totalizzato un punto, comunica al server di essere stato eliminato e a termina la propria esecuzione. Per ogni bomba lanciata, b può collezionare un massimo di 3 punti. Se però al momento dell'esplosione anche b si trovava in i, non guadagna nessun punto, comunica al server di essere stato eliminato e termina la sua esecuzione.

3.3.4 Vittoria

Quando un *peer* raggiunge il numero di punti p da ottenere per la vittoria, comunica in broadcast a tutti gli altri giocatori di avere vinto. Dopodichè comunica al server che lui è il vincitore e che la partita è da considerare conclusa.

4 Specifiche implementative: *server*

Un server REST si occupa di gestire la rete dei peer. In particolare, deve fornire ai peer dei metodi per:

- avere l'elenco delle partite in corso
- avere i dettagli di una partita (comprendenti anche indirizzo IP e porta dei giocatori già connessi)
- creare una nuova partita
- aggiungere un utente ad una partita in corso
- cancellare un utente da una partita
- eliminare una partita

Ogni partita è caratterizzata da un nome univoco e dalla lista dei giocatori attivi. Quando un utente viene aggiunto ad una partita, gli vengono anche comunicati i dettagli della stessa. Una partita può avere anche un solo giocatore. All'interno di una stessa partita, due giocatori non possono avere lo stesso username. Quando il server si accorge che in una partita non sono più rimasti giocatori e non ci sono vincitori, questa viene automaticamente terminata e rimossa.

5 Sincronizzazione

La rete dei peer è strutturata ad anello. La comunicazione tra i peer avviene tramite socket. Quando entra in una partita, ogni peer riceve dal server l'elenco di tutti i giocatori connessi e lo memorizza. Per garantire le comunicazioni in broadcast, questa lista deve essere aggiornata quando nuovi peer vengono aggiunti o quando dei giocatori vengono eliminati. La sincronizzazione distribuita viene garantita mediante algoritmo di mutua esclusione su rete ad anello. Ogni comunicazione (spostamento, lancio bomba, ...) può essere effettuata solo quando si è in possesso del token. Quindi, una volta ricevuta una mossa da standard input, prima di ricevere in input la mossa successiva, l'applicazione peer aspetta di avere il token e di inviare in broadcast la comunicazione. Se il *peer* riceve il token e non ha mosse da fare,

rilascia immediatamente il token. Altrimenti, il token viene rilasciato una volta finita la comunicazione. In ogni caso, quando il *peer* ha il token può effettuare solo una mossa alla volta. Ogni peer passa il token al peer che segue nell'elenco (l'ultimo peer lo passa al primo). Quando un peer viene eliminato, comunica la propria disconnessione al server.

6 Semplificazioni e limitazioni

Si ricorda che lo scopo del progetto è dimostrare la capacità di progettare e realizzare un'applicazione distribuita e pervasiva. Pertanto gli aspetti non riguardanti il protocollo di comunicazione, la concorrenza e la gestione dei dati di sensori sono considerati secondari. Inoltre è possibile assumere che :

- nessun nodo si comporti in maniera maliziosa,
- nessun nodo termini in maniera incontrollata

Si gestiscano invece i possibili errori di inserimento dati da parte dell'utente. Inoltre, il codice deve essere robusto: tutte le possibili eccezioni devono essere gestite correttamente.

Sebbene le librerie di Java forniscano molteplici classi per la gestione di situazioni di concorrenza, per fini didattici gli studenti sono invitati a fare **esclusivamente uso di metodi e di classi spiegati durante il corso di laboratorio**. Pertanto, eventuali strutture dati di sincronizzazione necessarie (come ad esempio lock, semafori o buffer condivisi) dovranno essere implementate da zero e saranno discusse durante la presentazione del progetto.

Nonostante alcune problematiche di sincronizzazione possano essere risolte tramite l'implementazione di server iterativi, per fini didattici si richiede di utilizzare server multithread. Inoltre, per le comunicazioni via socket, è richiesto di usare formati standard (ad esempio JSON, XML) per lo scambio di dati. Qualora fossero previste comunicazioni in broadcast, queste devono essere effettuate in parallelo e non sequenzialmente.

7 Presentazione del progetto

Il progetto è da svolgere individualmente. Durante la valutazione del progetto verrà richiesto di mostrare alcune parti del programma, verrà verificata la padronanza del codice presentato, verrà verificato il corretto funzionamento del programma e verranno inoltre poste alcune domande di carattere teorico inerenti gli argomenti trattati nel corso (parte di laboratorio). E' necessario presentare il progetto sul proprio computer.

Il codice sorgente dovrà essere consegnato al docente prima della discussione del progetto. Per la consegna, è sufficiente archiviare il codice in un file zip, rinominato con il proprio numero di matricola (es. 760936.zip) ed effettuare l'upload dello stesso tramite il sito http://upload.di.unimi.it . Sarà possibile effettuare la consegna a partire da una settimana prima della data di ogni appello. La consegna deve essere tassativamente effettuata entro le 23:59 del secondo giorno precedente quello della discussione (es. esame il 13 mattina, consegna entro le 23.59 dell'11).

Si invitano inoltre gli studenti ad utilizzare, durante la presentazione, le istruzioni `Thread.sleep()` al fine di mostrare la correttezza della sincronizzazione del proprio programma. Si consiglia vivamente di analizzare con attenzione tutte le problematiche di sincronizzazione e di rappresentare lo schema di comunicazione fra le componenti. Questo schema deve rappresentare il formato dei messaggi e la sequenza delle comunicazioni che avvengono tra le componenti in occasione delle varie operazioni che possono essere svolte. Tale schema sarà di grande aiuto, in fase di presentazione del progetto, per verificare la correttezza della parte di sincronizzazione distribuita.

Nel caso in cui il docente si accorga che una parte significativa del progetto consegnato non è stata sviluppata dallo studente titolare dello stesso, lo studente in causa: a) dovrà superare nuovamente tutte le prove che compongono l'esame del corso. In altre parole, se l'esame è composto di più parti (teoria e laboratorio), lo studente dovrà sostenere nuovamente tutte le prove in forma di esame orale (se la prova di teoria fosse già stata superata, questa verrà annullata e lo studente in merito dovrà risostenerla). b) non potrà sostenere nessuna delle prove per i 2 appelli successivi. Esempio: supponiamo che gli appelli siano a Febbraio, Giugno, Luglio e Settembre, e che lo studente venga riconosciuto a copiare all'appello di Febbraio. Lo studente non potrà presentarsi agli appelli di Giugno e Luglio, ma potrà sostenere nuovamente le prove dall'appello di Settembre in poi. Il docente si riserva la possibilità di assegnare allo studente in causa lo svolgimento di una parte integrativa o di un nuovo progetto.

8 Parti facoltative

8.1 Prima parte

Per ragioni di analisi, è necessario memorizzare informazioni sui giocatori e sullo storico delle partite che sono state disputate. Di ogni partita (conclusa) si vuole memorizzare: il nome della partita, il numero di giocatori che hanno partecipato (con il relativo punteggio), il vincitore e la durata. È quindi necessario realizzare un client che permetta ad un particolare tipo di utente, chiamato *analista*, di interrogare il server sulle statistiche: a) di tutte le partite giocate, b) di una specifica partita (dando come input il nome della partita), c) della classifica generale dei giocatori (sommando i punteggi con-

siderando le varie partite disputate). È quindi necessario estendere il server REST in modo tale da supportare questo tipo di interrogazioni. L'analista può anche decidere di registrarsi nel sistema (specificando username e il proprio indirizzo) per ricevere notifiche in real-time sullo svolgimento delle partite. Se un'analista con lo stesso username è già presente, la registrazione deve fallire. Altrimenti, il server risponde confermando che la registrazione ha avuto successo. Una volta registrato, l'analista riceverà delle notifiche in real-time dal server (da stampare a schermo) ogni volta che una partita è iniziata o conclusa (e con quale vincitore). Implementare la soluzione evitando che il client analista chieda continuamente in polling al server se sono presenti aggiornamenti.

8.2 Seconda parte

La tecnica di gestione della sincronizzazione che si richiede di implementare nella versione base del progetto soffre di problemi di scalabilità con l'aumentare del numero di giocatori. Ad esempio, spostamenti di utenti diversi in celle diverse non dovrebbero essere sequenzializzati, in quanto possono essere svolti in parallelo. Si proponga quindi una soluzione alternativa maggiormente scalabile e con un più alto grado di concorrenza rispetto alla rete ad anello.

9 Aggiornamenti

Qualora fosse necessario, il testo dell'attuale progetto verrà aggiornato al fine di renderne più semplice l'interpretazione. Le nuove versioni del progetto verranno pubblicate sul sito del corso. Si consiglia agli studenti di controllare regolarmente il sito.

Al fine di incentivare la presentazione del progetto nei primi appelli disponibili, lo svolgimento della parte facoltativa 1 diventa obbligatoria a partire dall'appello di Settembre (incluso), mentre entrambe le parti facoltative (1 e 2) saranno obbligatorie negli appelli successivi.