Battle city cliente-servidor en java

Alumnos::
Ayrton Coronado - acoronadoh@uni.pe
Flores Huamaní - lfloresh@uni.pe
Barrientos Porras - herlees.barrientos.p@uni.pe
Universidad Nacional de Ingeniería, Facultad de Ciencias,

Curso:
CC462 Sistemas concurrentes y distribuidos
Laboratorio 2

Resumen

El presente trabajo es la implementación del juego Battle City en lenguaje java bajo la arquitectura cliente-servidor, con el uso de sockets para lograr una comunicación entre diferentes clientes y el servidor, genera nuevos threads para cada proceso independiente; para lograr la sincronización entre clientes, el servidor realiza un broadcast de la información que recibe .

Palabras clave: Battle city, sockets, cliente, servidor, threads, java.

Contenidos

1 Introducción

Toda aplicación,archivo,página web,etc que queramos acceder mediante internet aplica una arquitectura de cliente-servidor, la implementación más básica de esta arquitectura es mediante el uso de sockets y threads . Usamos java ya que su manejo de threads está muy desarrollado y posee una clase importante para su manejo la cual es Threads y la interfaz Runnable.

Desarrollamos la implementación del juego Battle City entre 2 jugadores, comunicando ambos clientes mediante el servidor a través de sockets en diferentes hilos.

2 Marco Teórico

2.1 Creación de Threads.

Hay dos formas de crear un nuevo hilo de ejecución. Una es declarar una clase como una subclase de Thread. Esta subclase debería anular el método run de la clase Thread. Entonces se puede asignar e iniciar una instancia de la subclase. Por ejemplo, un hilo que calcula números primos mayores que un valor establecido podría escribirse de la siguiente manera:

```
class PrimeThread extends Thread {
    long minPrime;
    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }
    public void run() {
        // compute primes larger than minPrime
        . . .
    }
}
```

El siguiente código crearía un hilo y lo comenzaría a ejecutar:

```
PrimeThread p = nuevo PrimeThread (143);
    p.start ();
```

La otra forma de crear un hilo es declarar una clase que implemente la interfaz Runnable. Esa clase luego implementa el método run. Luego se puede asignar una instancia de la clase, pasarla como argumento cuando creamos Thread e iniciarlo. El mismo ejemplo en este otro estilo tiene el siguiente aspecto:

```
class PrimeRun implements Runnable {
```

```
    long minPrime;
    PrimeRun(long minPrime) {
      this.minPrime = minPrime;
    }
    public void run() {
      // compute primes larger than minPrime
      . . .
    }
  }
```

El siguiente código crea un thread y lo inicia:

```
PrimeRun p = new PrimeRun(143);
  new Thread(p).start();
```

2.2 Cliente-Servidor mediante sockets

Para crear una conexión por sockets entre cliente y servidor usando threads :

En la clase servidor

```
Class Server implements Runnable {
        ServerSocket serverSocket;
        Socket client;
        Public static final int SERVERPORT = 4444;
        . . .
        public void run(){
                try{
                        serverSocket = new ServerSocket(SERVERPORT);
                        client = serverSocket.accept();
                        . . .
                }catch(Exception e){
                        System.out.println("Error: "+ e.getMessage());
                }
        }
}
```

En la clase cliente

```
Class Cliente implements Runnable{
        public string SERVERIP;
        public static final int SERVERPORT = 4444;
        Socket socket;
        . . .
        public void run(){
                try{
                InetAddress serverAddr = InetAddress.getByName(SERVERIP);
                socket = new Socket(serverAddr, SERVERPORT);
                . . .
                }catch(Exception e){
                        System.out.println("Error: "+ e.getMessage());
                }
}
```

3 Metodología

3.1 Manejo de threads

Para la implementación del juego se requiere trabajar estrictamente con threads ya que se generan procesos independientes para el correcto funcionamiento.
Por ejemplo:

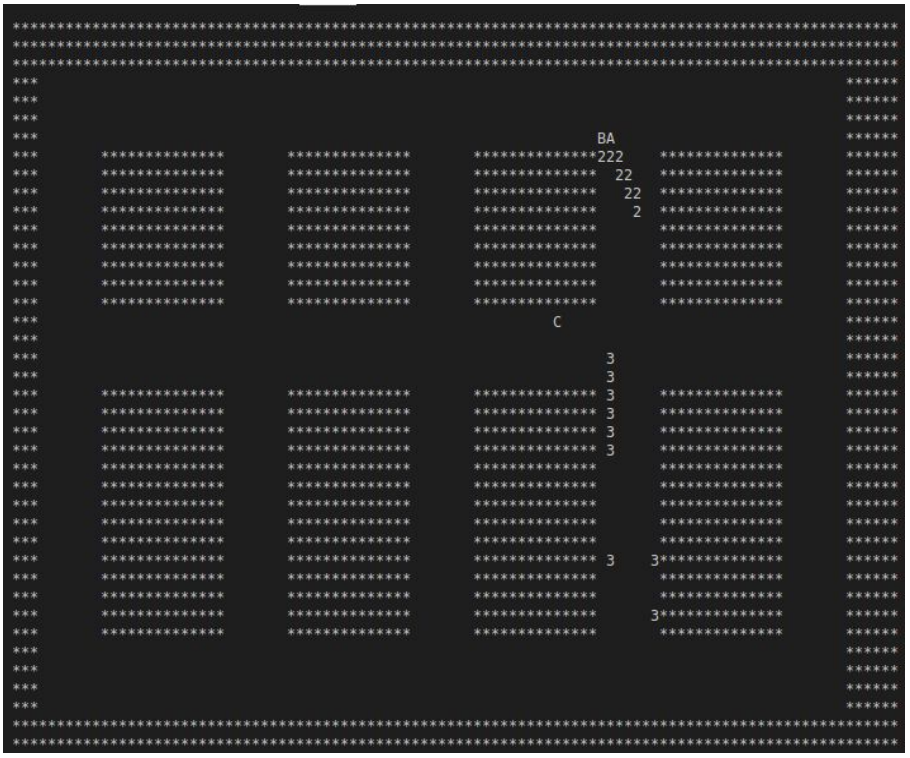
. Proceso principal para iniciar el servidor y que reciba y envíe información.
. Procesos principales para iniciar cada cliente, que reciban y envíen información.
. Proceso para aceptar nuevas conexiones por socket en el servidor.
. Proceso para cada conexión por socket creada por cada cliente.
. Proceso para cada objeto nuevo creado con comportamiento independiente : minas,etc

3.2 Comunicación entre clientes

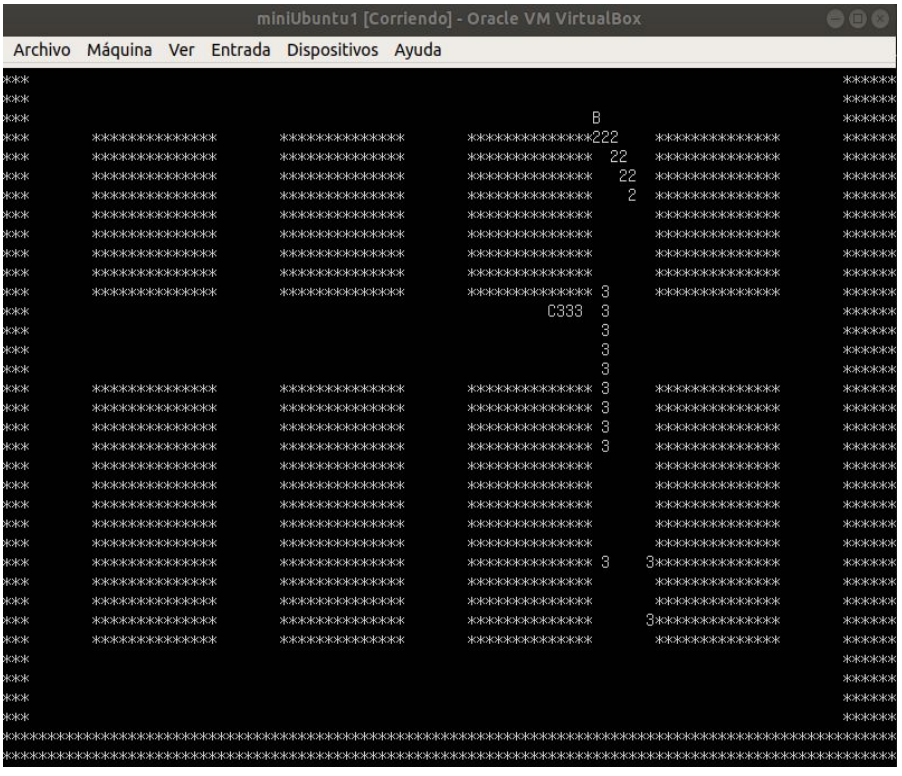
Para lograr que diferentes clientes se comuniquen y sincronicen sus acciones, el servidor realiza un broadcast de la información que recibe de cada cliente.

4 Results and Discussion

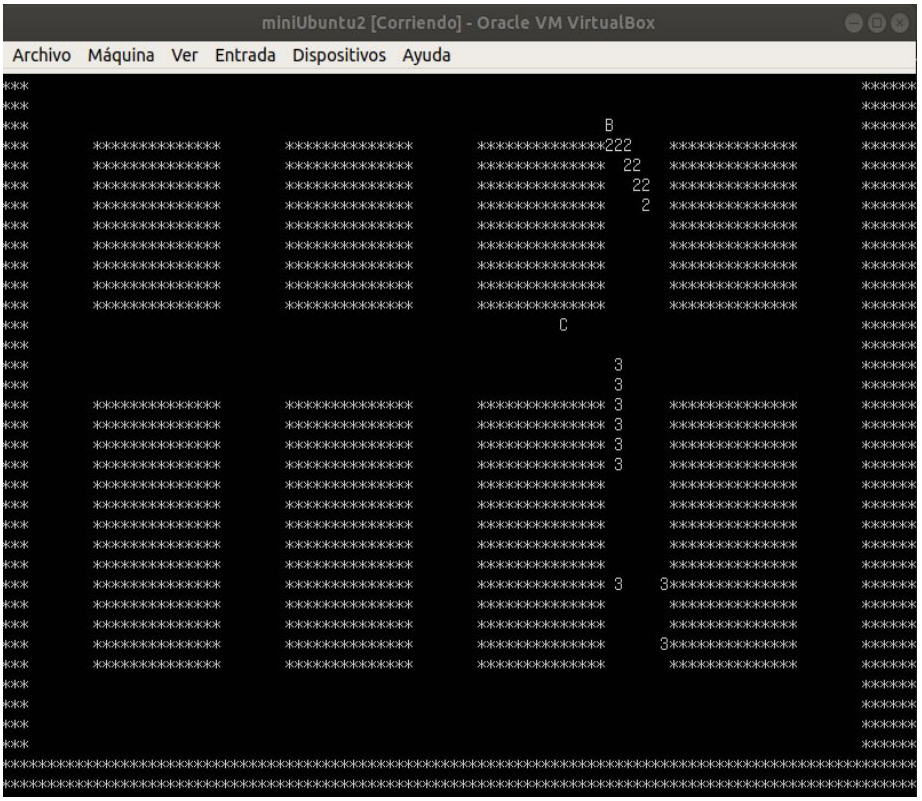
Results: Jugador A



Jugador B



Jugador C



5 Conclusiones

- Se logró el la implementación del juego Battle City en la arquitectura cliente-servidor mediante sockets y threads.

- El uso de threads resulta necesario, ya que permite que múltiples objetos independientes se ejecuten paralelamente.

- El uso de sockets permitió que varios jugadores puedan jugar entre ellos a pesar de ejecutarse en diferentes máquinas.

6 Anexo Código

- https://github.com/lfloresh/CC462Concurrentes

7 Anexo Documentación

- https://docs.oracle.com/en/java/javase/14/
- https://es.wikipedia.org/wiki/Battle_City