Bomberman Game

Students:

Anthony Daniel, Bautista León
Katheryn Ximena, Peralta Haro
Renzo Renato, Quispe Amao
Universidad Nacional de Ingeniería, Facultad de Ciencias,
e-mail: abautistal@uni.pe, katheryn.peralta.h@uni.pe, rrquispea@uni.pe

Curso:

CC4P1 Programación Concurrente y Distribuida
Laboratorio 02

Abstract

El objetivo de este trabajo es familiriazarnos con el uso de Hilos *(Thread)* sincronizados en la máquina virtual de java (JVM)y *sockets*. Para ellos se planteó el desarrollo del juego Bomberman. El juego se desarrolla en una matriz y utilizamos caracteres para los jugadores y bombas.

Keywords: Thread, JVM, socket.

Contents

1 Introducción

Todo dispositivo con un aplicación de escritorio, web o móvil conectado a internet utiliza un arquitectura de cliente-servidor. La implementación más básica de esta arquitectura es utilizando *sockets* y *thread*. Se desarrolló el juego multijugador online llamado Bomberman.

Bomberman es una franquicia de videojuegos estratégico-laberínticos originalmente desarrollada por Hudson Soft y actualmente por Konami. Fue lanzado en 19983 para PC.

2 Marco Teórico

2.1 Threads

La Máquina Virtual Java (JVM) es capaz de manejar multihilos, es decir, puede crear varios flujos de ejecución simultánea, administrando los detalles como asignación de tiempos de ejecución o prioridades de los hilos.
Java proporciona soporte para hilos a través de una interfaz y un conjunto de clases. La interfaz de Java y las clases que proporciona algunas funcionalidades sobre hilos son:

- *Thread*
- *Runnable* (interfaz)
- *ThreadDeath*
- *ThreadGroup*
- *Object*

Tanto las clases como la interfaz son parte del paquete básico de java (*java.lang*).

Clase *Thread*

Es la clase en Java responsable de producir hilos funcionales para otras clases. Para añadir la funcionalidad de hilo a una clase solo se hereda de ésta.
La clase *Thread* posee el método *run*, el cual define la acción de un hilo y , por lo tanto, se conoce como el cuerpo del hilo. La clase *Thread* también define los métodos *start* y *stop*, los cuales permiten inciar y detener la ejecución del hilo.
Por lo tanto, para añadir la funcionalidad deseada a cada hilo creado es necesario redefinir el método *run*. Este método es invocado cuando se inicia el hilo. Un hilo se inicia mediante una llamada al método *start* de la clase *Thread*.

Interfaz *Runnable*

La interfaz *Runnable* permite producir hilos funcionales para otras clases. Para añadir la funcionalidad de hilo a una clase por medio de *Runnable*, solo es necesario implementar la interfaz.
La interfaz *Runnable* proporciona un método alternativo al uso de la clase *Thread*, para casos en los que no es posible hacer que la clase definida herede de la clase *Thread*.
Las clases que implementan la interfaz *Runnable* proporcionan un método *run* que es ejecutado por un objeto *Thread* creado. Ésta es una herramienta muy útil y, a menudo, es la única salida que se tiene para incorporar hilos dentro de las clases.

2.2 Sockets

Los sockets son un sistema de comunicación entre procesos de diferentes máquinas de una red. Más exactamente, un socket es un punto de comunicación por el cual un proceso puede emitir o recibir información.

Fueron popularizados por Berckley Software Distribution, de la universidad norteamericana de Berkley. Los sockets han de ser capaces de utilizar el protocolo de streams TCP (Transfer Contro Protocol) y el de datagramas UDP (User Datagram Protocol).

Utilizan una serie de primitivas para establecer el punto de comunicación, para conectarse a una máquina remota en un determinado puerto que esté disponible, para escuchar en él, para leer o escribir y publicar información en él, y finalmente para desconectarse.

3 Metodología

3.1 Threads

Hay dos formas de crear hilos: crear un objeto de la clase *Thread* o bien extendiendo la interfaz *Runnable*. Todos los hilos son instancias de la clase *Thread* por lo tanto lo natural es crearlos extendiendo dicha clase. El método *run* es el punto de entrada a la ejecución del hilo, debe sobreescribirse en las subclases.
La ejecución del hilo al terminar la ejecución secuencial de las instrucciones en el método *run*. Para inciar un nuevo hilo definido en miHilo, se debe crear una instancia de esta clase y llamar al método start(), el cual invoca indirectamente a run.

Creando hilos extendiendo de la clase Thread

```
public class ThreadChild extends Thread {
        public ThreadChild(String str) {
                super(str);
        }
        public void run() {
                for (int i = 0; i < 10 ; i++)
                        System.out.println(i + " " + getName());
                System.out.println("Termina thread " + getName());
        }
        public static void main (String [] args) {
                new ThreadChild("thread hijo 1").start();
                new ThreadChild("thread hijo 2").start();
                System.out.println("Termina thread main");
        }
}
```

Creando hilos implementando la interfaz Runnable

```
public class ThreadChildRunnable implements Runnable {
        public void run() {
                for (int i = 0; i < 5 ; i++)
                        System.out.println(i + " " +
                                Thread.currentThread().getName());
                System.out.println("Termina thread " +
                        Thread.currentThread().getName());
        }
        public static void main (String [] args) {
                new Thread (new ThreadChildRunnable(), "thread 1").start();
                new Thread (new ThreadChildRunnable(), "thread 2").start();
                System.out.println("Termina thread main");
        }
}
```

3.2 Sockets

Para comprender el funcionamiento de los sockets se mostrará una pequeña implementación de un dialogo entre un programa servidor y sus clientes.

Programa Servidor:

```
import java.io.* ;
import java.net.* ;
class Servidor {
  static final int PUERTO=5000;
  public Servidor( ) {
    try {
      ServerSocket skServidor = new ServerSocket( PUERTO );
      System.out.println("Escucho el puerto " + PUERTO );
      for ( int numCli = 0; numCli < 3; numCli++; ) {
        Socket skCliente = skServidor.accept(); // Crea objeto
        System.out.println("Sirvo al cliente " + numCli);
        OutputStream aux = skCliente.getOutputStream();
        DataOutputStream flujo= new DataOutputStream( aux );
        flujo.writeUTF( "Hola cliente " + numCli );
        skCliente.close();
      }
      System.out.println("Demasiados clientes por hoy");
    } catch( Exception e ) {
      System.out.println( e.getMessage() );
    }
  }
  public static void main( String[] arg ) {
    new Servidor();
  }
}
```

Programa Cliente:

```
import java.io.*;
import java.net.*;
class Cliente {
```

```
  static final String HOST = "localhost";
  static final int PUERTO=5000;
  public Cliente( ) {
    try{
      Socket skCliente = new Socket( HOST , Puerto );
      InputStream aux = skCliente.getInputStream();
      DataInputStream flujo = new DataInputStream( aux );
      System.out.println( flujo.readUTF() );
      skCliente.close();
    } catch( Exception e ) {
      System.out.println( e.getMessage() );
    }
  }
  public static void main( String[] arg ) {
    new Cliente();
  }
}
```

4 Desarrollo de la Investigación

Para el desarrollo del juego Bomberman se creó un programa para el cliente y otro para el servidor con sus respectivas clases.

- Cliente
  - Bomba
  - Bomberman
  - Mapa
  - Jugador
  - TCPJugador
- Servidor
  - Servidor
  - TCPServidor
  - TCPservidorHilo

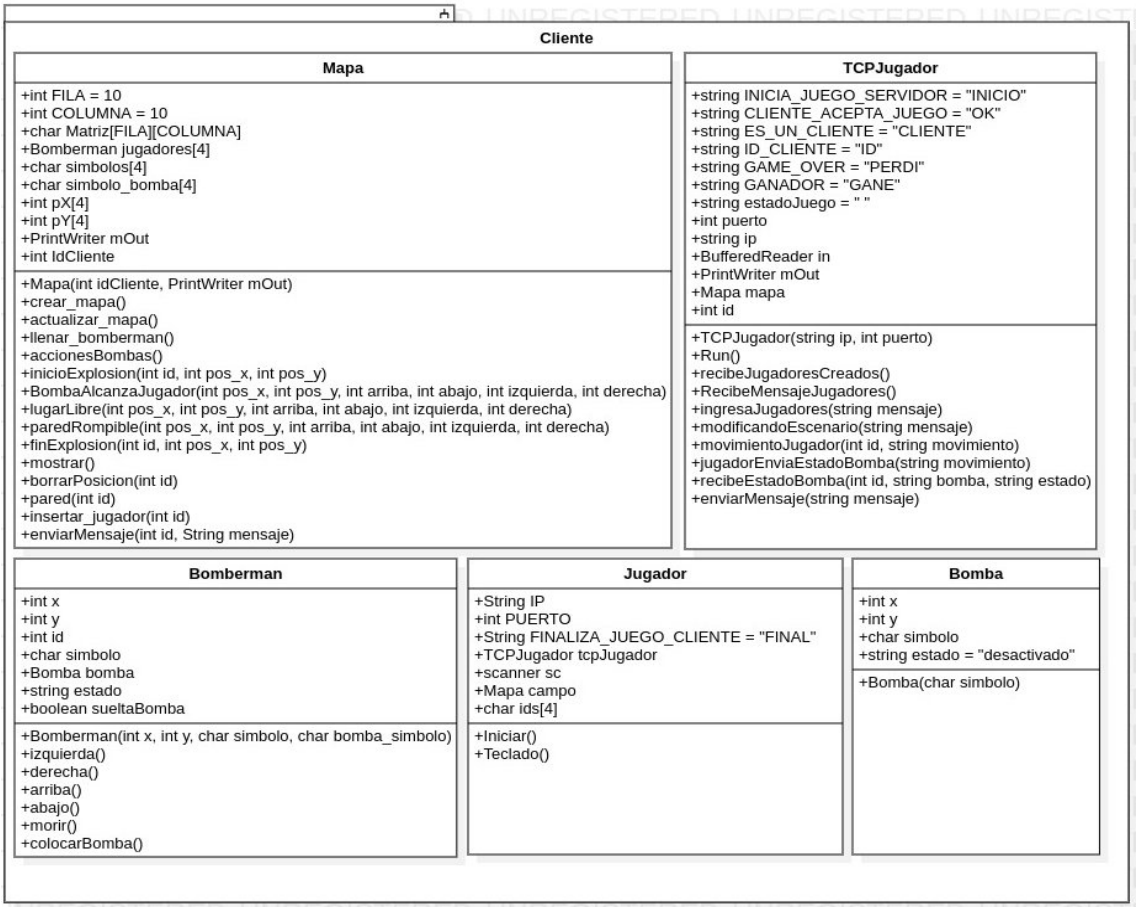

Figure 1: Diagrama de clases del programa cliente

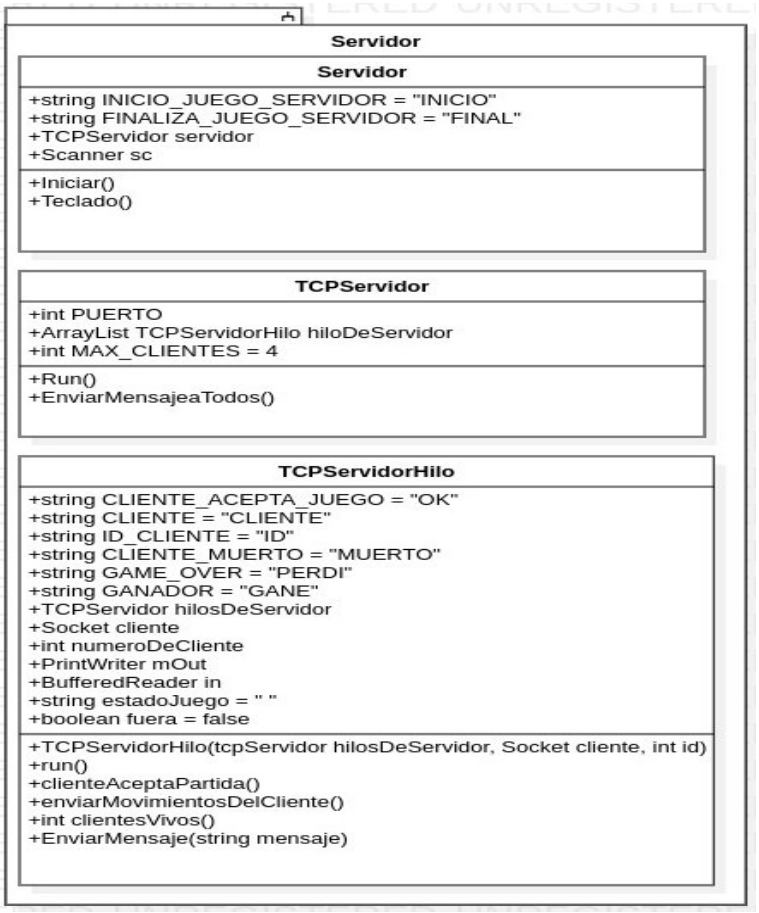

Figure 2: Diagrama de clases del programa servidor

5 Resultados

El juego se desarrolla en una matriz en la cual se desarrolla en una matriz. Los jugadores son graficados con letras "A","B","C","D". Cada bomberman se mueven con las letras "a"(izquierda),"s"(abajo),"d"(derecha),"w"(arriba) y se puede colocar la bomba con la letra "e"(bomba). Cuando se coloca la bomba esta empieza a cargar y se grafica con un número $1, 2, 3, 4$ dependiendo de que jugador sea. Una vez terminado el tiempo de carga esta explota y el daño es graficado con "%" . Para iniciar el juego el servidor envia la palabra "INICIO".

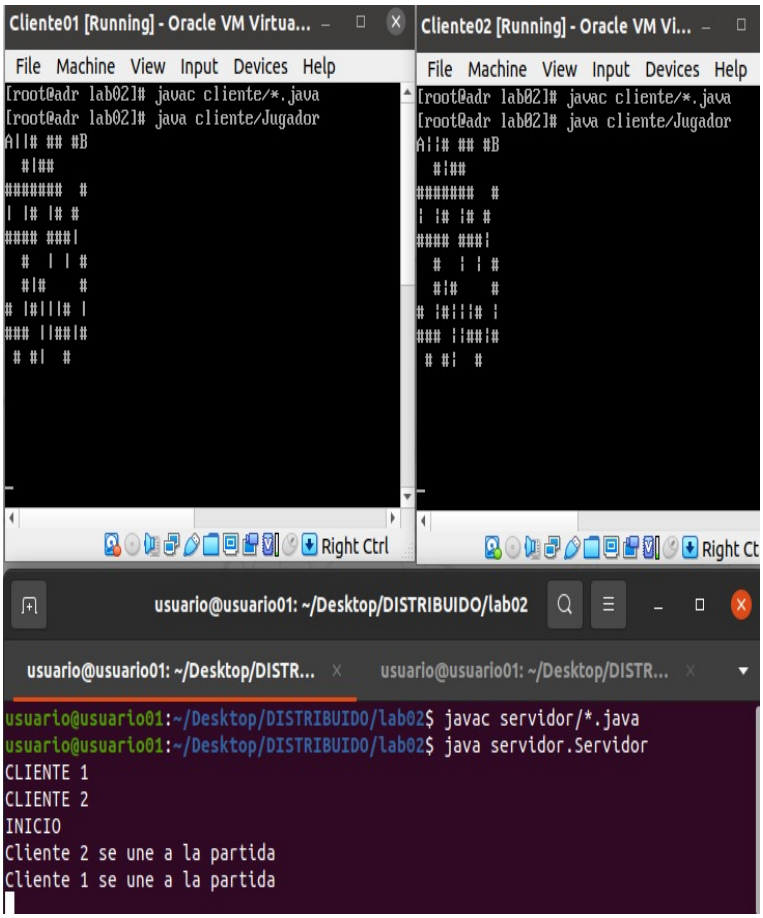

Figure 3: Inicio del juego

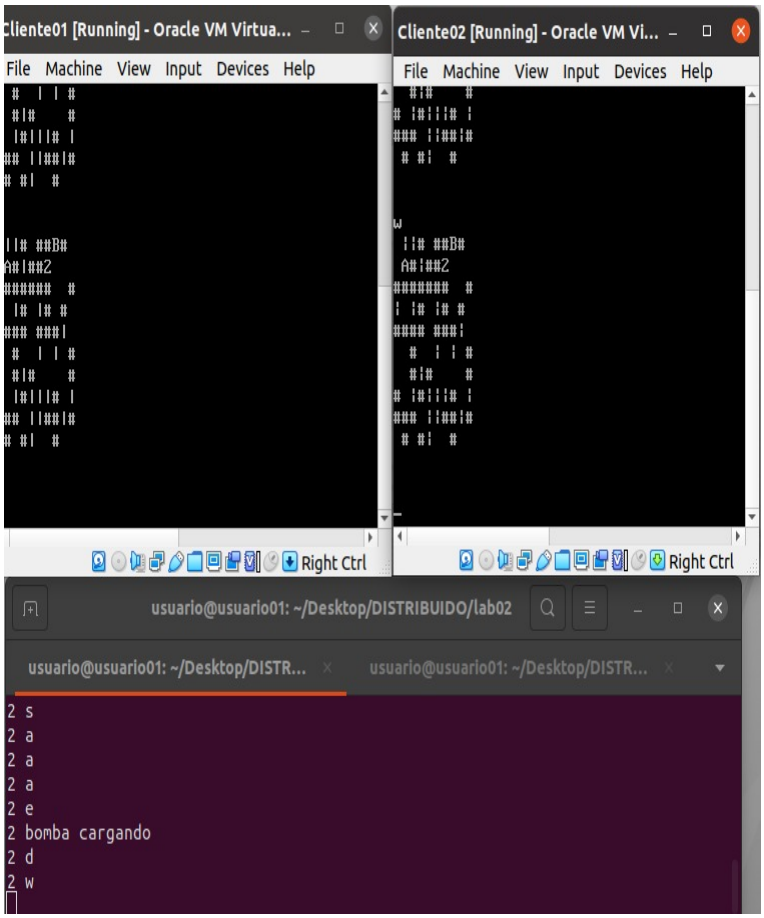

Figure 4: Movimiento con bomba

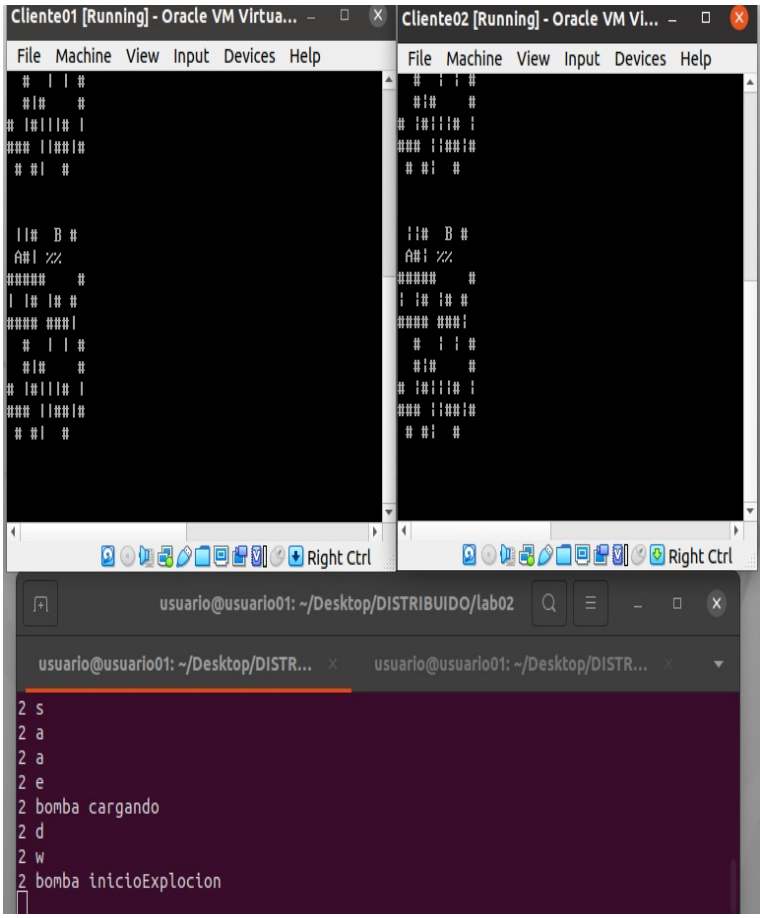

Figure 5: Explosión de la bomba

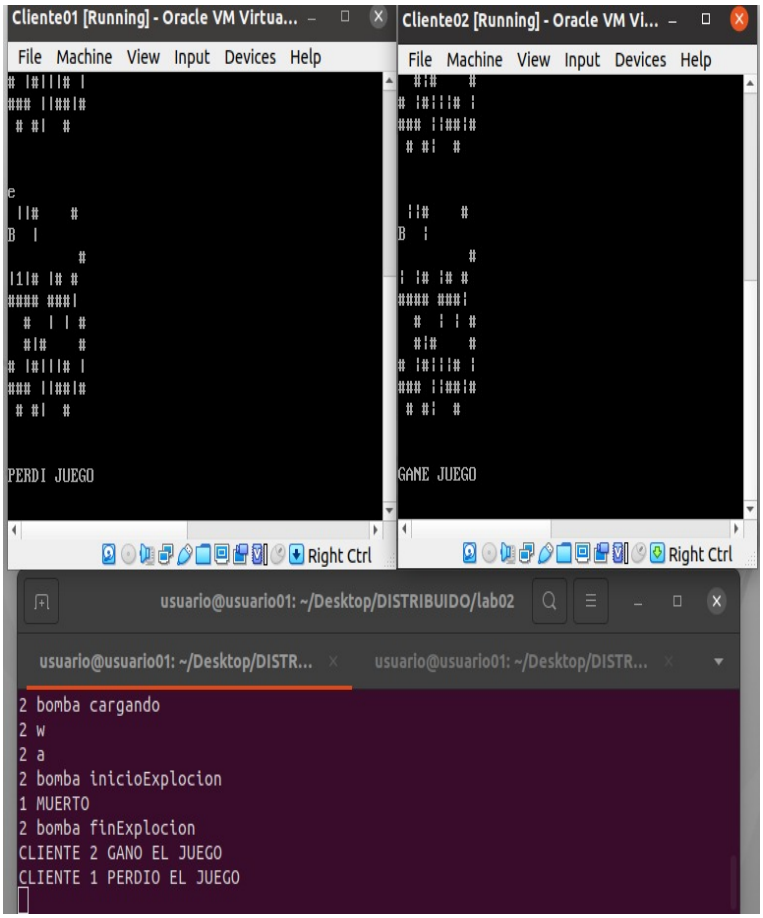

Figure 6: Final del juego

6 Conclusiones

- Se logró el desarrollo de Bomberman con una arquitectura cliente-servidor mediante socket en java.
- El uso de thread fue necesario para permitir que se ejecuten multiples tareas.

7 Anexo Código

`https://github.com/renzoqamao/CC4P1-ProgramacionConcurrente-Distribuida`

8 Anexo Documentación

- `https://es.wikipedia.org/wiki/Bomberman`
- `https://cruzado.info/tutojava/V_2.htm#:~:text=Los%20Sockets%20en%20Java&text=Los%20sockets%20son%20un%20sistema,puede%20emitir%20o%20recibir%20informaci%C3%B3n.&text=Con%20todas%20primitivas%20se%20puede%20crear%20un%20sistema%20de%20di%C3%A1logo%20muy%20completo.`
- `https://www.infor.uva.es/~fdiaz/sd/doc/hilos`