
BATTLE CITY CLIENTE-SERVIDOR

A PREPRINT

Ipanaque Casquina Ingrid
Facultad de Ciencias
Universidad Nacional de
Ingeniería
iipanaquec@uni.pe

Castillo Flores Junior
Facultad de Ciencias
Universidad Nacional de
Ingeniería
junior.castillo.f@uni.pe

July 6, 2020

ABSTRACT

La expresión cliente servidor se utiliza en el ámbito de la informática, en dicho contexto, se llama cliente al dispositivo que requiere ciertos servicios a un servidor. La idea de servidor, por su parte, alude al equipo que brinda servicios a las computadoras (ordenadores) que se hallan conectadas con él mediante una red. El concepto de cliente servidor, o cliente-servidor, refiere por lo tanto a un modelo de comunicación que vincula a varios dispositivos informáticos a través de una red. El cliente, en este marco, realiza peticiones de servicios al servidor, que se encarga de satisfacer dichos requerimientos. Con esta arquitectura, las tareas se distribuyen entre los servidores (que proveen los servicios) y los clientes (que demandan dichos servicios). Dicho de otro modo: el cliente le pide un recurso al servidor, que brinda una respuesta.

Keywords cliente-servidor · sockets · hilos

1 INTRODUCCIÓN

Battle city fue un juego de tanques popular en los años 90, producido por la empresa Namco, este juego consiste en controlar un tanque sobre un escenario plagado de tanques enemigos. Su misión consiste en evitar que destruyan su base militar. Se completa el nivel cuando haya destruido todos los tanques enemigos. El juego termina si el enemigo destruye la base o el jugador gasta todas sus vidas.

En este trabajo, intentaremos replicar el videojuego utilizando el paradigma cliente-servidor, en donde el servidor se encargara de realizar modificaciones en el mapa, a medida que los clientes envíen una solicitud de cambio de posición, o de disparar. Para esto se utilizará los programas de conexión por sockets vistos en clase, además de crear una red en una máquina virtual usando minimal ubuntu.

2 OBJETIVOS

- Crear el juego battle city con cliente-servidor en java, de manera concurrente.
- Implementar una subred con minimal ubuntu, en donde se pueda hacer las pruebas del, mediante la conexión por sockets.
- Implementar varios niveles en el juego.

3 DESCRIPCIÓN DEL PROYECTO

El videojuego se ha programado en java creando varias clases para el cliente y su conexión, y toda la jugabilidad se realizó en el servidor. Contamos con una matriz que almacena las posiciones de los jugadores, de los tanques enemigos, de las paredes, y de las balas.

cada jugador es independiente en la conexión y modificación de datos al servidor, esto se logra utilizando

em metodo synchronized de java para la recepcion de mensajes.

El juego termina si un tanque enemigo choca con el jugador, o si este recibe una bala de algun enemigo.

4 CLASES UTILIZADAS PARA LOS OBJETOS EN LA MATRIZ

Definimos varias clases para el manejo de los objetos dentro del mapa.

```
package battlecity;

public class Enemigos {
    boolean vida = false;
    int posX;
    int posY;
    int dimX = Global.dimX;
    int dimY = Global.dimY;
    boolean disparo = false;
    Bala bala;
    String direccion = "arriba";
    private int desplazamiento = 1;

    public Enemigos(int posX, int posY) {
        this.posX = posX;
        this.posY = posY;
        this.vida = true;
        bala = new Bala(posX, posY, 0, 0, false);

        //this.dimX = dimX;
        //this.dimY = dimY;
    }
}
```

Clase para el objeto Enemigo

```
package battlecity;

public class Jugador {

    private int posX = 3;
    private int posY = 4;
    private int id;
    private String direccion = "derecha";
    private int desplazamiento = 1;
    boolean vida = true;
    Bala bala;
    boolean disparo = false;
    int dimX = Global.dimX;
    int dimY = Global.dimY;

    public Jugador(int id) {
        this.id = id;
        this.disparo = false;
        //this.posX = posX;
        //this.posY = posY;
        //this.dimX = dimX;
        //this.dimY = dimY;
    }
}
```

Clase para el objeto Jugador

```

package battlecity;

public class Bala {
    int posX;
    int posY;
    int vX;
    int vY;
    boolean isEnemy;

    public Bala(int x , int y ,int vx,int vy,boolean isEnemy) {
        this.posX = x;
        this.posY = y;
        this.vX = vx;
        this.vY = vy;
        this.isEnemy = isEnemy;
    }
}

```

Clase para el objeto Bala

5 INSTRUCCIONES PARA LOS MOVIMIENTOS

Los jugadores se mueven con su identificador de jugador seguido de:

w : arriba

s : abajo

a : izquierda

d : derecha

t : dispara

6 HILO DE CONTROL DEL MAPA EN EL SERVIDOR

Para controlar los eventos en el mapa dentro del servidor, creamos un hilo independiente del hilo que recepciona los mensajes con los clientes.

En este hilo instanciamos nuestro objeto **battlecity** en el cual se ubican los jugadores y mientras haya jugadores conectados (el método vida comprueba esto), movemos los enemigos, dibulamos el mapa, e verificamos los choques entre las balas de los enemigos con los jugadores, asi mismo los choques entre ellos, y tambien los choques que matan a los enemigos.

Todos estos metodos estan creados dentro de la clase **BattleCity**, y es ahi donde hacemos los cambios para cada cliente, desde el array de **jugadores**

```

Thread battlecity = new Thread() {
    @Override
    public void run() {
        bc = new BattleCity();
        jugadores[jugadorId] = new Jugador(jugadorId);
        bc.ubicarJugador(jugadores[jugadorId]);

        while(Vida()){

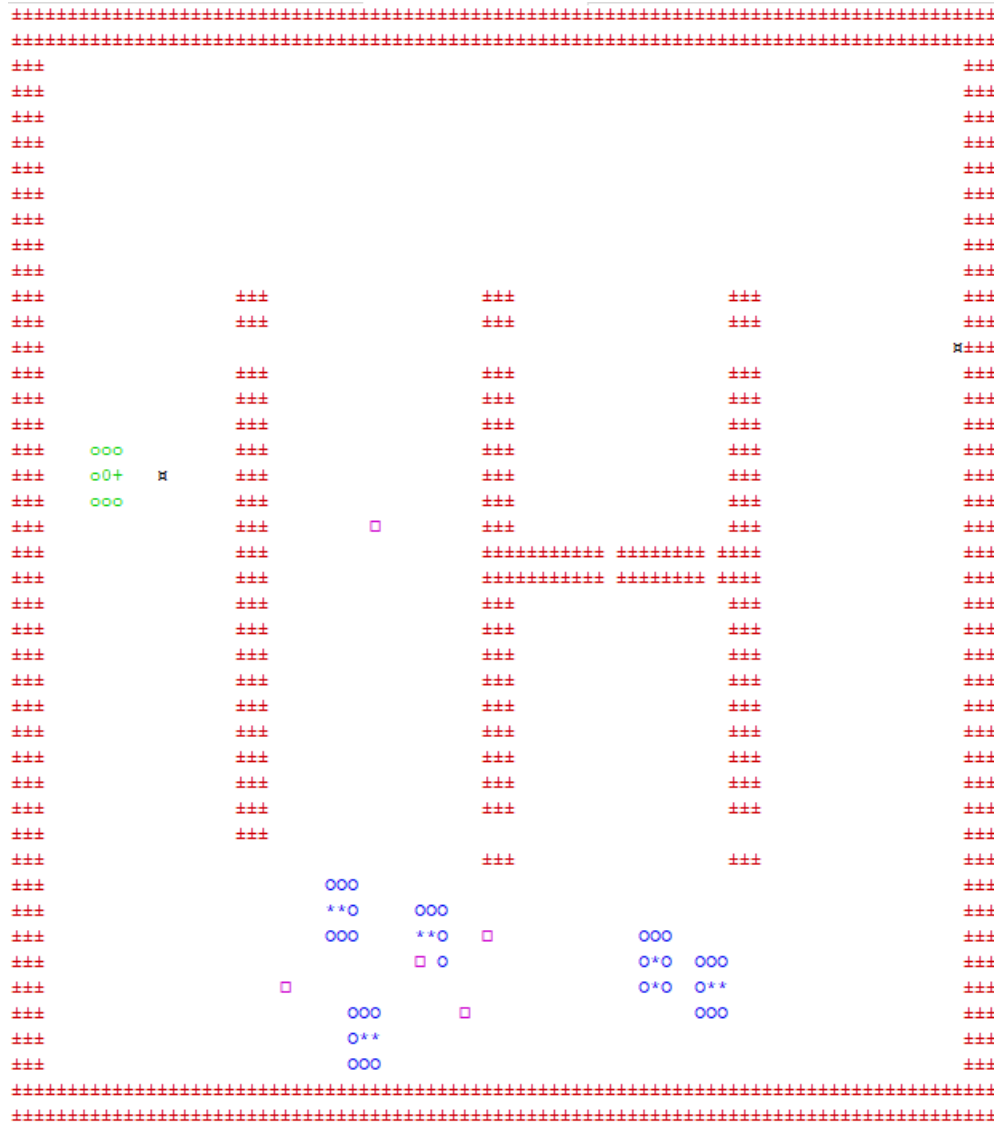
            bc.moverEnemigos();
            vidaJugadores(); //cambio
            mapa = bc.mapa();
            for (int i= 0; i <= jugadorId; i++) {
                bc.choqueEnemigos(jugadores[i]);
                bc.moverBalasJugadores(jugadores[i]);
                for(int j = 0; j< Global.cantEnemigos;j++){
                    bc.choqueDisparoEnemigo(bc.enemigos[j].bala, jugadores[i]);
                }
            }

            try{
                Thread.sleep(800);
            }
            catch(InterruptedException e){
                // this part is executed when an exception (in this example Inter
            }
            ServidorEnvia(mapa);
        }
    }
};
//battlecity.sleep(12000);
battlecity.start();

```

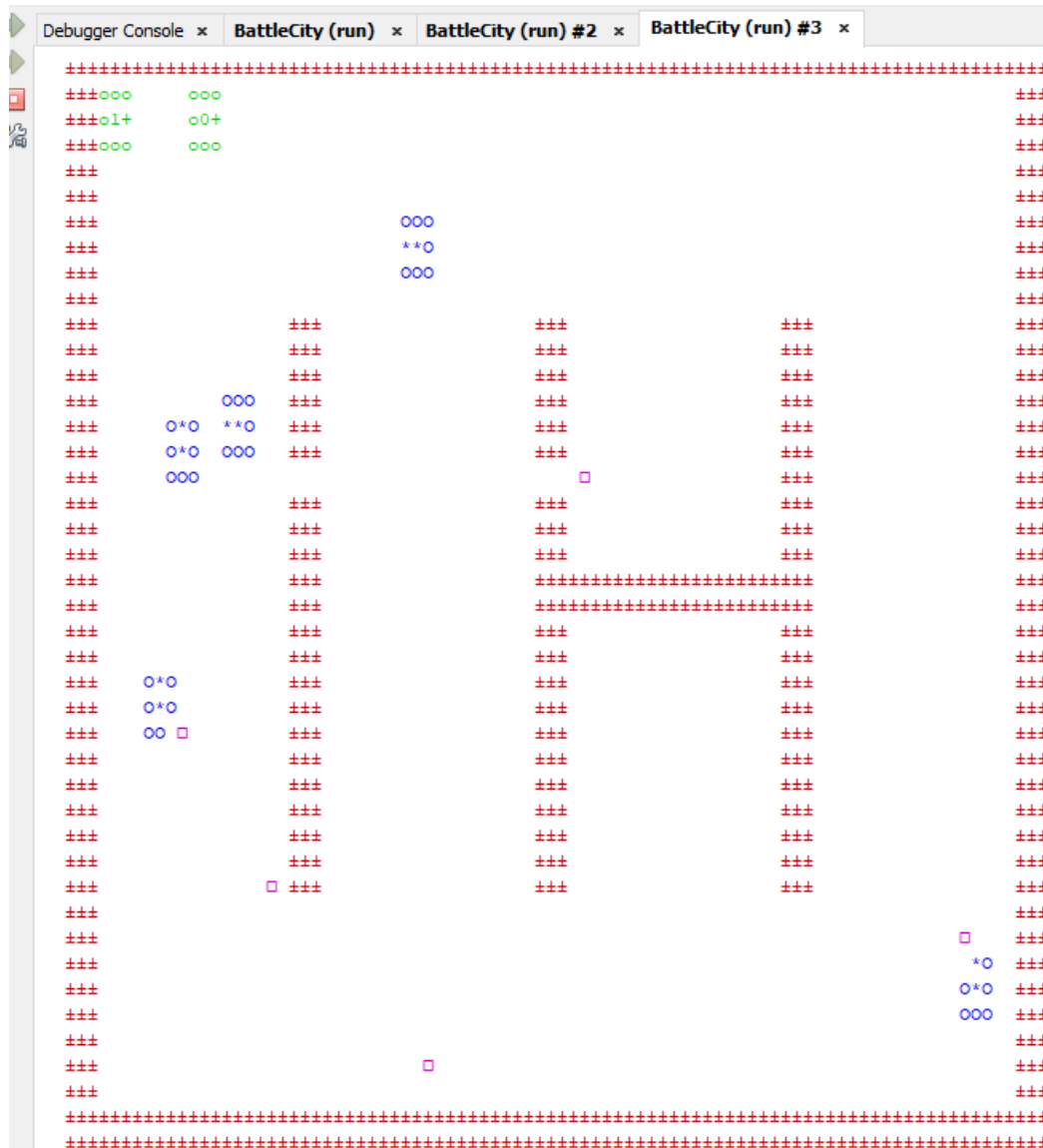
Mapa con los jugadores, enemigos, balas y muros

El siguiente mapa nos muestra nuestro tanque del cliente 0 (su identificador, esta en el centro del tanque), disparando una bala, mientras que los enemigos se encuentran moviendose y disparando.



Mapa con los jugadores, enemigos, balas y muros

Cuando conectamos un segundo cliente podemos ver que su identificador (idCliente), aparece en el centro del tanque.



Nuevo cliente agregado al mapa, ambos se mueven de manera independiente.

7 PRUEBAS CON MINIMAL UBUNTU

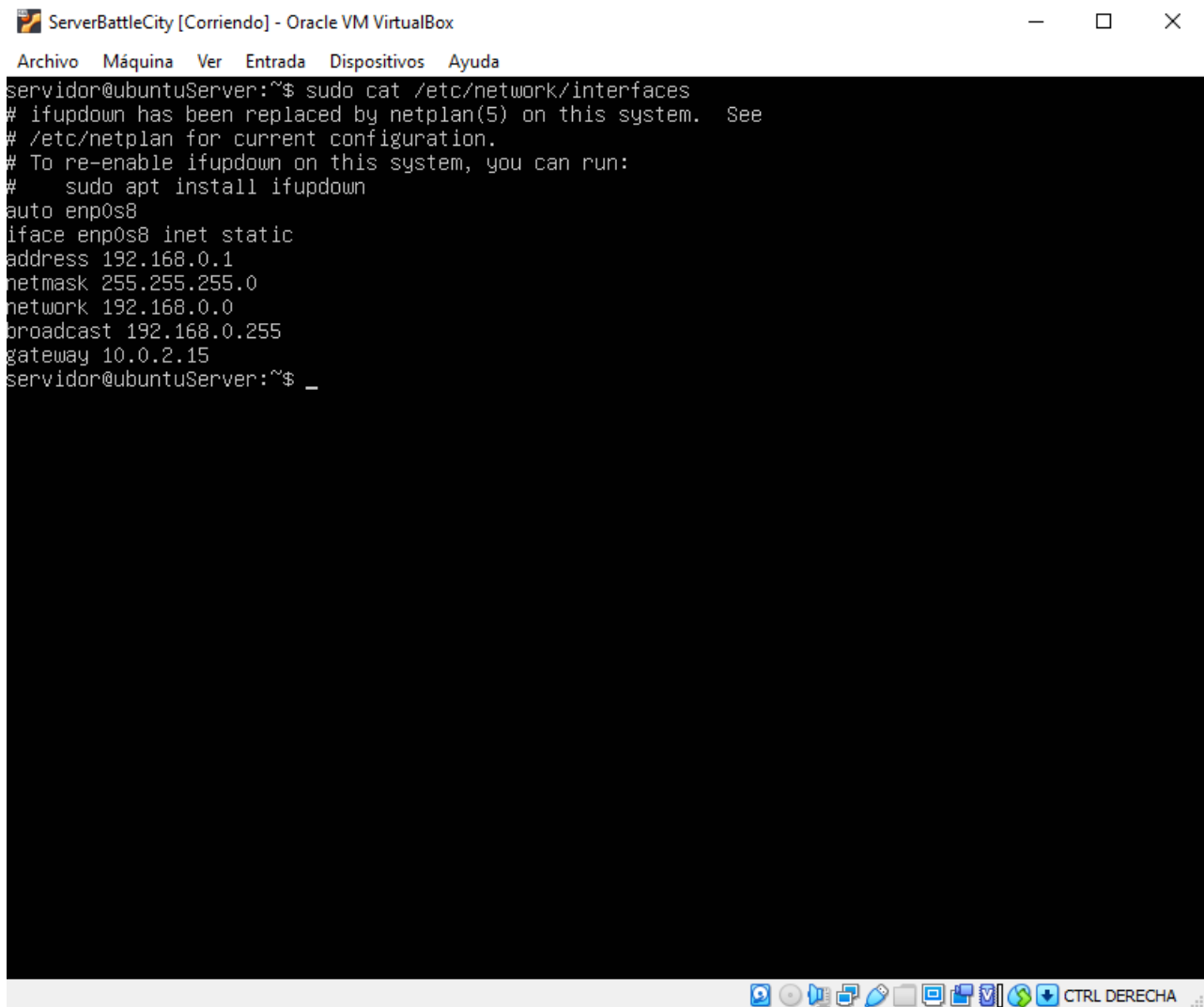
Hemos instalado en VirtualBox tres maquinas virtuales, con minimal ubuntu, en las cuales hemos configurado una subred, donde el Servidor tiene una interfaz de red de salida para la subred con:

ADDRESS : 192.168.0.1
SUBMASK : 255.255.255.0
NETWORK : 192.168.0.0

Los clientes pertenecen a esa misma subred y se conectan por esa interfaz, sus direcciones varían en los bits de HOST.

Cliente: ADDRESS : 192.168.0.i

donde i puede tomar valores entre 2 hasta 254.



The screenshot shows a terminal window titled "ServerBattleCity [Corriendo] - Oracle VM VirtualBox". The terminal displays the output of the command `sudo cat /etc/network/interfaces`. The output shows a configuration for the `enp0s8` interface with static IP `192.168.0.1`, netmask `255.255.255.0`, network `192.168.0.0`, broadcast `192.168.0.255`, and gateway `10.0.2.15`. The terminal prompt is `servidor@ubuntuServer:~$`. The window has a menu bar with "Archivo", "Máquina", "Ver", "Entrada", "Dispositivos", and "Ayuda". The bottom of the window shows a taskbar with various icons and the text "CTRL DERECHA".

```
servidor@ubuntuServer:~$ sudo cat /etc/network/interfaces
# ifupdown has been replaced by netplan(5) on this system.  See
# /etc/netplan for current configuration.
# To re-enable ifupdown on this system, you can run:
#   sudo apt install ifupdown
auto enp0s8
iface enp0s8 inet static
address 192.168.0.1
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 10.0.2.15
servidor@ubuntuServer:~$ _
```

Configuración de la red en el servidor

The screenshot shows a terminal window titled "ClienteUno (GNS3 Linked Base for clones) [Corriendo] - Oracle VM VirtualBox". The window contains a nano 2.9.3 editor editing the file `/etc/network/interfaces`. The configuration for the `enp0s3` interface is as follows:

```
# ifupdown has been replaced by netplan(5) on this system. See
# /etc/netplan for current configuration.
# To re-enable ifupdown on this system, you can run:
#   sudo apt install ifupdown
auto enp0s3
iface enp0s3 inet static
address 192.168.0.2
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1
```

At the bottom of the terminal, there is a status bar with various keyboard shortcuts and a message "[11 líneas leídas]". The status bar includes shortcuts for Ver ayuda, Guardar, Buscar, Cortar Text, Justificar, Posición, Deshacer, Salir, Leer fich., Reemplazar, Pegar txt, Ortografía, Ir a línea, and Rehacer. On the far right, there is a "CTRL DERECHA" button.

Configuración en el cliente 1, esto puede variar en el último dígito de address.

Verificamos que se encuentren conectados en la misma subred haciendo un **ping**.

The image shows two side-by-side Oracle VM VirtualBox windows. The left window, titled 'ClienteUno (GNSS Linked Base for clones) [Corriendo] - Oracle VM VirtualBox', displays a terminal session where a user runs 'ping 192.168.0.1'. The output shows eight successful ping requests from 192.168.0.1 to 192.168.0.1, each with a TTL of 64 and varying response times between 0.268 ms and 0.327 ms. The right window, titled 'ServerBattleCity [Corriendo] - Oracle VM VirtualBox', displays a terminal session where a user runs 'ping 192.168.0.2'. The output shows eight successful ping requests from 192.168.0.2 to 192.168.0.2, each with a TTL of 64 and varying response times between 0.291 ms and 0.356 ms. Both windows have a menu bar with 'Archivo', 'Máquina', 'Ver', 'Entrada', 'Dispositivos', and 'Ayuda'. The status bar at the bottom of each window shows system icons and 'CTRL DERECHA'.

Ping entre cliente-servidor.

Observaciones:

Se tuvo que instalar los servicios desde cero, para el **networking-manager**, **net-tools**. [2]

Para compilar el programa en java se instaló el **JDK_version_8**. [3]

Por último para poder tener los archivos, se utilizó un repositorio de **GitHub**.

8 RESULTADOS

9 Bibliografía

1. colaboradores de Wikipedia. (2020, 17 marzo). Battle City. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Battle_City
2. R. (2019, 28 abril). Install NetworkManager in Ubuntu • Linux Hub. <https://linuxxx.info/install-networkmanager-in-ubuntu/>
3. apt-get install openjdk-7-jdk doesn't install javac. Why? (2012, 29 marzo). Ask Ubuntu. <https://askubuntu.com/questions/117189/apt-get-install-openjdk-7-jdk-doesnt-install-javac-why>