Universidad Nacional de Ingeniería

CC4P1 Programación Concurrente y Distribuída

PC2: Bomberman con sockets

Profesor:
Yuri Nuñez
Departamento de Ciencias de la Computación

Grupo:
Ronaldo Lopez
Davis Alderete
Cristhian Sánchez Sauñe

Índice

1. Introducción

El proyecto consiste en diseñar, configurar y simular el popular videojuego Bomberman usando Sockets y Threads en Java. Antes de comenzar, pasaremos a repasar algunas definiciones.

1.1. Definiciones

- **Thread**. Es la menor de las estructuras lógicas de programación que se ejecuta de forma secuencial por parte del planificador del sistema operativo (nota: forma secuencial no quiere decir que no haya bucles, solo que es una línea secuencial de ejecución: una sentencia después de otra).

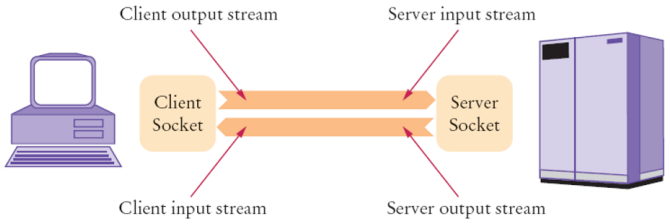

Figura 1: Sockets entre cliente y servidor

- **Sockets**. Es conocido como un tipo de software que actúa como un punto final que funciona estableciendo un enlace de comunicación de red bidireccional entre el extremo del servidor y el programa receptor del cliente.
- **Puerto TCP/IP**. En TCP/IP, el protocolo que usan los ordenadores para entenderse en Internet -y actualmente casi en cualquier otra red-, el puerto es una numeración lógica que se asigna a las conexiones, tanto en el origen como en el destino. No tiene ninguna significación física.

2. Personajes del juego

- **Jugadores**. Actualmente el juego dispone de hasta 4 jugadores de color amarillo, rojo, blanco y negro. Para poder controlar la dirección contamos con las teclas A-S-D-W. Cada personaje tiene movimientos personalizados cuando se mantiene quieto, cuando se mueve a la derecha o a la izquierda, o cuando muere.
- **Enemigos**. Actualmente el juego solo cuenta con un tipo de enemigo, el cual no cuenta con movimientos personalizados. Cuando un jugador se acerca a este enemigo, automáticamente muere.

Figura 2: Jugadores disponibles

Figura 3: Enemigo por defecto

- **Obstáculos**. El juego cuenta solo con un tipo de obstáculo, tal como se muestra en la figura. Este obstáculo desaparece tras ser afectado por una bomba.

Figura 4: Obstáculo por defecto

- **Bombas**. El juego cuenta solo con un tipo de bomba, el cual cuenta con animaciones personalizadas. Si el jugador esta en la zona de la explosión automáticamente morirá. Actualmente aún no se cuenta con la funcionalidad de poder matar a los enemigos con estas bombas.

Figura 5: Bomba disponible

3. Lógica y características del juego

El juego sigue un esquema cliente-servidor, en donde cada jugador obtiene los datos de los demás jugadores a través del servidor. El juego consta de varias clases, a continuación se explicarán las principales.

- Clase Player encargada de manejar los atributos y métodos de cada jugador. Usa un identificador **id** que permitirá informar al servidor el jugador que actualmente esta realizando los movimientos.

```
public class Player {
   int x, y;
   String status, color;
   JPanel panel;
   boolean alive;

   StatusChanger sc;

   Player(int id, JPanel panel) throws InterruptedException {
      this.x = Client.spawn[id].x;
      this.y = Client.spawn[id].y;
      this.color = Sprite.personColors[id];
      this.panel = panel;
      this.alive = Client.alive[id];

      (sc = new StatusChanger(this, "wait")).start();
   }

   public void draw(Graphics g) {
      if (alive)
         g.drawImage(Sprite.ht.get(color + "/" + status), x, y,
         Const.WIDTH_SPRITE_PLAYER, Const.HEIGHT_SPRITE_PLAYER, null);
   }
}
```

- Clase Sender encargada de enviar los mensajes de cada jugador al servidor por medio de *Sockets*. Los mensajes generalmente contienen una palabra clave acompañada de una tecla presionada por el jugador.

```
public class Sender extends KeyAdapter {
   int lastKeyCodePressed;

   public void keyPressed(KeyEvent e) {
```

```
        if (e.getKeyCode() == KeyEvent.VK_SPACE)
            Client.out.println("pressedSpace "
                    + Game.you.x + " " + Game.you.y);
        else if (isNewKeyCode(e.getKeyCode()))
            Client.out.println("keyCodePressed " + e.getKeyCode());
    }

    public void keyReleased(KeyEvent e) {
        Client.out.println("keyCodeReleased " + e.getKeyCode());
        lastKeyCodePressed = -1;
         // la siguiente clave siempre será nueva
    }

    boolean isNewKeyCode(int keyCode) {
        boolean ok = (keyCode != lastKeyCodePressed) ? true : false;
        lastKeyCodePressed = keyCode;
        return ok;
    }
}
```

- Clase Window que sera la encarga de pintar los elementos en pantalla, y volver a repintar cada vez que el cliente mande algún cambio al servidor.

```
class Window extends JFrame {
    private static final long serialVersionUID = 1L;

    Window() {
        Sprite.loadImages();
        Sprite.setMaxLoopStatus();

        add(new Game(Const.COL * Const.SIZE_SPRITE_MAP,
                Const.LIN * Const.SIZE_SPRITE_MAP));
        setTitle("bomberman");
        pack();
        setVisible(true);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        addKeyListener(new Sender());
    }
}
```

Nota: Para ver con más detalle el resto del código puede revisar nuestro repositorio.

Figura 6: Interfaz del juego

4. Performance

A continuación se muestran las medidas de la performance del juego desde 1 hasta 4 jugadores. El procesador con el que se hacen estas medidas es un Intel i7 8750H de 12 hilos.

Con 1 jugador activo, se pueden observar pequeñas subidas temporales de algunos hilos.

Con 2 jugadores activos, el cambio ya se hace notorio, pero aún no se observa demasiado ruido en la gráfica.

Para 3 jugadores activos, los hilos están superando el 10 % de funcionamiento individual.

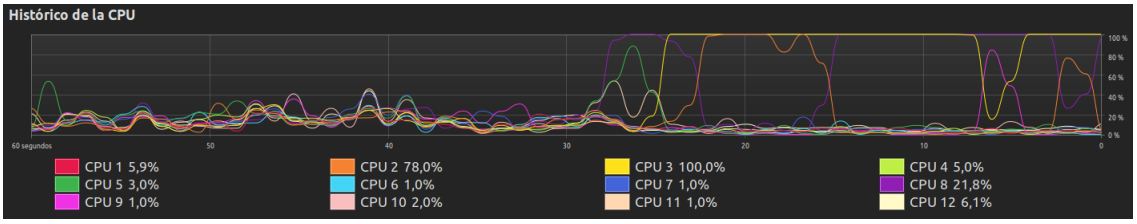

Figura 7: Con 1 jugador

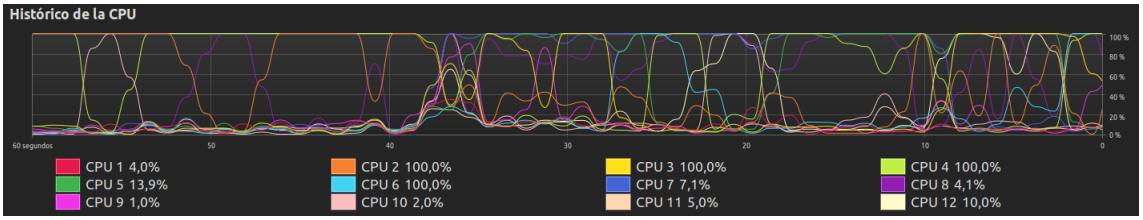

Figura 8: Con 2 jugadores

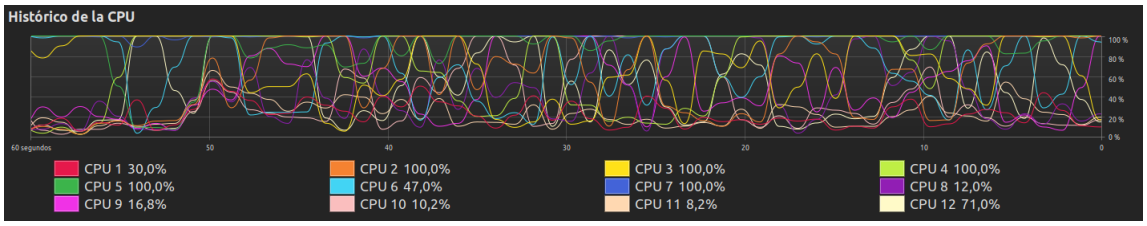

Figura 9: Con 3 jugadores

Para 4 jugadores activos, los hilos están superando el 20 % de funcionamiento individual.

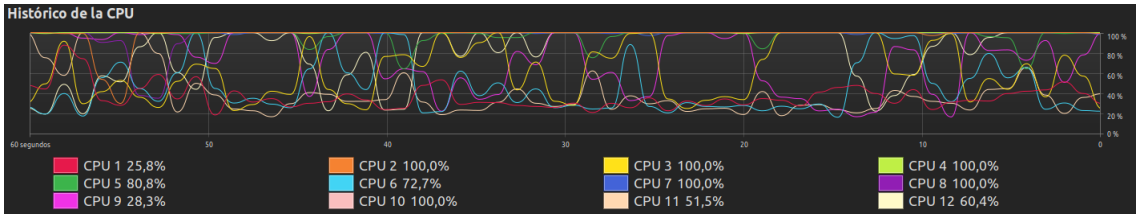

Figura 10: Con 4 jugadores

Finalmente, después de cerrar el juego se puede observar un descenso en las operaciones de los hilos, que se traduce en un uso de a lo mucho 18 % de la CPU.

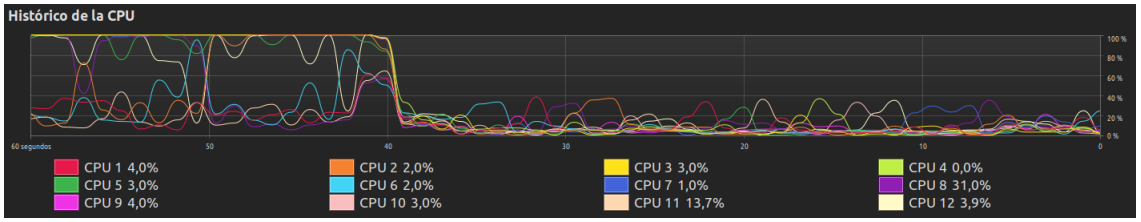

Figura 11: Inmediatamente después de cerrar el juego

Cabe recalcar que en todo momento no se ha observado ningún tirón en el juego, todo ha marchado fluidamente. Además la CPU no ha sufrido ningún cambio elevado de temperatura.

5. Conclusiones

Se presentó el popular juego de Bomberman usando sockets e hilos para poder hacer medir la performance en tiempo real, de donde se concluye que el uso de sockets es una muy buena alternativa para problemas en tiempo real, ya que la distribución de carga entre los diferentes hilos es manejable y entendible. Sin embargo, hay algunos aspectos que no se pudieron solucionar del todo, tales como:

- En la construcción de la lógica de los enemigos (para que puedan moverse solos) se presentaron dificultades al momento de repintar la pantalla, ya que se presentaron errores al momento de manejar los hilos.
- Como solución alternativa al problema anterior, se busco replicar la lógica del jugador y modificarla para ser un enemigo, sin embargo el código se hizo muy extenso y complicado.

Pensamos que en una futura versión de este proyecto, podríamos optimizar mucha parte del código y agregar las funcionalidades que quedaron pendientes. También podríamos migrar de lenguaje a uno más especializado como Golang o JavaScript, principalmente por las librerías actuales y una mayor documentación disponible, comparado con Java.