Juego de la serpiente multijugador

Eduardo David Martínez Neri

***Resumen*—Se desarrolló una aplicación en Java del juego de la serpiente, con modalidad multijugador. La cual, mediante multidifusión [1] [2], y orden causal [3], permite generar nuevas coordenadas de los items (manzanas) sincronizados en todos los jugadores y estos compiten por llegar primero al item. El jugador que llegue primero al item informa a los demás que ya lo ganó y envía las coordenadas del nuevo item. Cuando un jugador por algún motivo pierde, informa al resto que ha perdido y sale de la partida y el último jugador conectado gana.**

***Keywords*—multidifusión, juego, causal, orden.**

I. Introducción

La implementación en Java realizó mediante el socket multicast que se encuentra en el paquete **java.net.MulticastSocket**, la cual ya es una abstracción de la comunicación UDP (User Datagram Protocol) con multidifusión, y solo se tienen que indicar ciertas propiedades como se indica en la sección a continuación. Y sobre los métodos creados de envío y recepción de mensajes por multidifusión se impementó el algoritmo de Fidge para mantener orden causal en los mensajes [3].

II. Implementación

II-A. Multidifusión

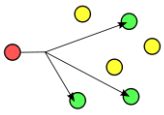

Figura 1. Multidifusión

Antes de poder enviar y recibir mensajes por multidifusión, deben establecerse una serie de parámetros. Es necesario establecer lo que se denomina **grupo multicast**, este grupo tiene la función de que cuando un mensaje se distribuye por la red, no alcanza a todos los dispositivos como sucede con broadcast, sino sólo a los integrantes de este grupo.

Ese grupo multicast tiene asociado una dirección de Internet. La versión actual del protocolo de Internet (Internet Protocol o IP), conocida como IPv4, reserva las direcciones de tipo D para la multidifusión. Las direcciones IP tienen 32 bits, y las de tipo D son aquellas en las cuales los 4 bits más significativos son '1110' (224.0.0.0 a 239.255.255.255)[4]. En este caso se utilizó la IP 225.4.5.7, con el puerto 5000.

En ambos casos, envío y recepción la creación del socket Multidifusión se genera de la siguiente manera:

```
private String group = "225.4.5.7";
private int port = 5000;
MulticastSocket s = new MulticastSocket(port)
s.joinGroup(InetAddress.getByName(group));
```

Una implementación completa de este código en Java, se puede encontrar en la documentación de Oracle https://docs.oracle.com/javase/tutorial/networking/datagrams/broadcasting.html.

II-B. Orden causal

Para mantener orden causal en los mensajes, se utilizó un algoritmo basado en relojes vectoriales de Fidge. Orden causal nos asegura un orden parcial de los mensajes, pero basta para identificar al jugador que llegó primero a la manzana y con esto aumentar correctamente la puntuación del jugador al que corresponde.

El algoritmo utilizado para orden causal asumiento que V_i son relojes vectoriales, es el siguiente:

Inicialización

$V_i^g := O(j = 1, 2..., N)$

Al enviar un mensaje.

$V_i^g[i] := V_i^g[i] + 1$
$Multidifusión(m, < V_i^g >)$

Al recibír un mensaje $(m, < V_j^g >)$ de p_j, con g = group(m).

Colocar $(m, < V_j^g >)$ en una cola de espera;
Esperar hasta que $V_j^g[j] = V_i^g[j] + 1$ y $V_j^g[k] \leq V_i^g[k](k \neq j)$;
Sacar m de la cola y entregar;
$V_i^g[j] := V_i^g[j] + 1$;

II-C. Mensajes

Un ejemplo de los mensajes transmitidos por multidifusión son los siguientes, los cuales se decodifican del lado del cliente.

.

Iniciar el juego Message CO: ¡452!Inicia el juego (6,31){452 = 1}

Cuando un usuario transmite la nueva coordenada
Message CO: ¡608!Coordenadas (13,20){608 = 3}

Estos mensajes se ejecutan en eventos de la aplicación, ver Fig. 2 y Fig. 3, respectivamente.

III. Conclusión

En diferentes partes del programa se encontraron dificultades de implementación, la más grande fue al intentar crecer el reloj vectorial dinámicamente, para no limitar el número de jugadores, y otra dificultad fue la implementación del algoritmo anterior con paso de mensajes UDP, y ligar esto con la interfaz gráfica. Se realizaron pruebas con dos y con tres jugadores, y se logró mantener la comunicación y consistencia de relojes vectoriales en todos los procesos, con lo que se validó el funcionamiento del juego, la comunicación por multicast y el orden causal. También se realizaron pruebas en diferentes arquitecturas Windows, Linux y Mac OS y diferentes procesadores y funcionó correctamente.

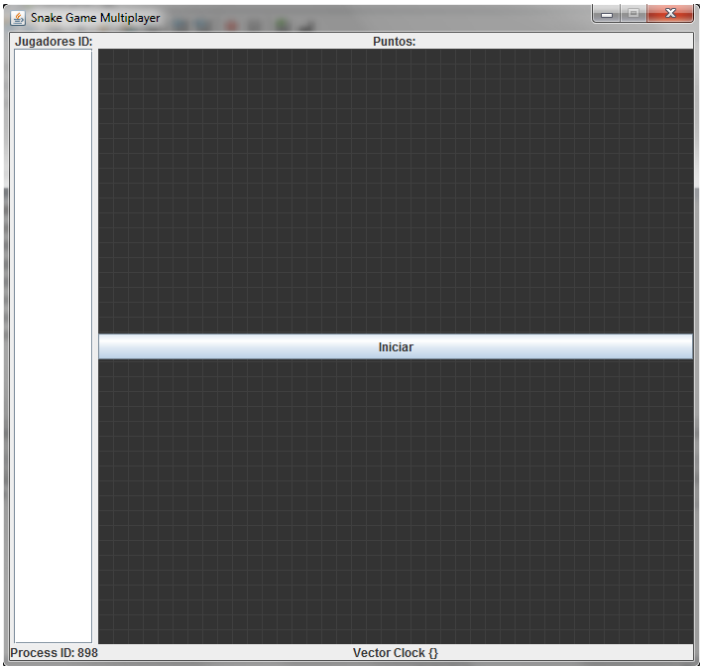

Figura 2. Interfaz gráfica del juego, al dar click en el botón inicio, los usuarios dentro del grupo multicast, reciben el mensaje iniciar juego.

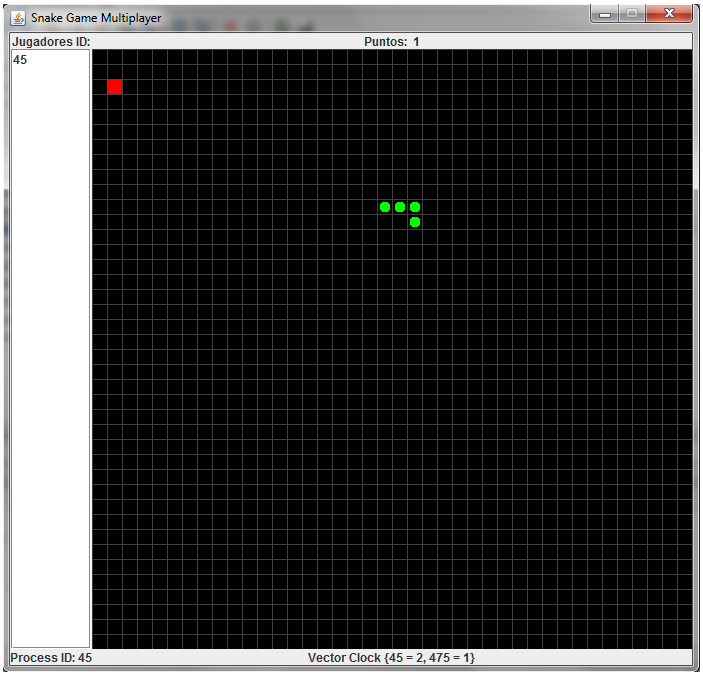

Figura 3. Interfaz gráfica del juego, se observa el reloj vectorial en la parte de abajo, el cual va incrementando en cada proceso al consumir primero un item. En todos los procesos este vector debe ser igual.

REFERENCIAS

[1] Birman, K. and Joseph, Th., *Reliable communication in the presence of failures*, ACM Transactions on Computer Systems, Vol. 5, No. 1, pp. 47-76., 1987.

[2] Birman, K., Schiper, A., and Stephenson, P., *Lightweight causal and atomic group multicast*, ACM Transactions on Computer Systems, Vol. 9, No. 3, pp. 272-314., 1991.

[3] Colin J. Fidge, Timestamps in Message-Passing Systems That Preserve the Partial Ordering, In K. Raymond (Ed.). Proc. of the 11th Australian Computer Science Conference (ACSC'88). pp. 56–66., February 1988

[4] Wikipedia, Multidifusión — Wikipedia, La enciclopedia libre, https://es.wikipedia.org/w/index.php?title=Multidifusi2016