TP 4: El TP de tu vida

Nicolas Mastropasqua

Programación sobre redes

October 5, 2022

1 Introducción

En 1970 el matemático John Horton Conway propuso el célebre "juego de la vida" que consiste en una serie de reglas que determinan la evolución del estado de células (autómatas) a partir de un estado inicial y del estado de sus vecinas. Se pude mostrar que el autómata es tan expresivo como una máquina de Turing [1].

1.1 Reglas del juego

Las células están organizadas en un grilla de tamaño finito y tendrán definido un vecino por cada dirección que este definida en los límites del tablero (por ejemplo, las células del borde superior no tendrán ningún vecino arriba). De esta manera, exceptuando los casos bordes, las células tienen nueve vecinos. La evolución de las células se da en pasos discretos que llamaremos turnos. En cada turno, las células actualizan su estado de acuerdo al estado anterior de sus vecinas siguiendo las siguientes reglas:

- Una célula muerta con exactamente 3 células vecinas vivas nace (es decir, al turno siguiente estará viva).
- Una célula viva con 2 o 3 células vecinas vivas sigue viva, en otro caso muere (por "soledad" o "super-población").

1.2 Patrones

Existen diversos patrones que pueden originarse en cualquier momento de la evolución de la comunidad de células. Algunos son fijos (por ejemplo un bloque de 4 células juntas siempre permanece igual), otros son oscilantes (por ejemplo, una barra vertical que en cada turno va rotando) o bien móviles ("naves") que se desplazan por la grilla. Se recomienda ver más ejemplos de patrones comunes acá [1]
Un resultado relevante es que el problema de saber si dado un patrón inicial cualquiera se va a alcanzar un patrón final determinado, es no decidible ya que es equivalente al halting problem.

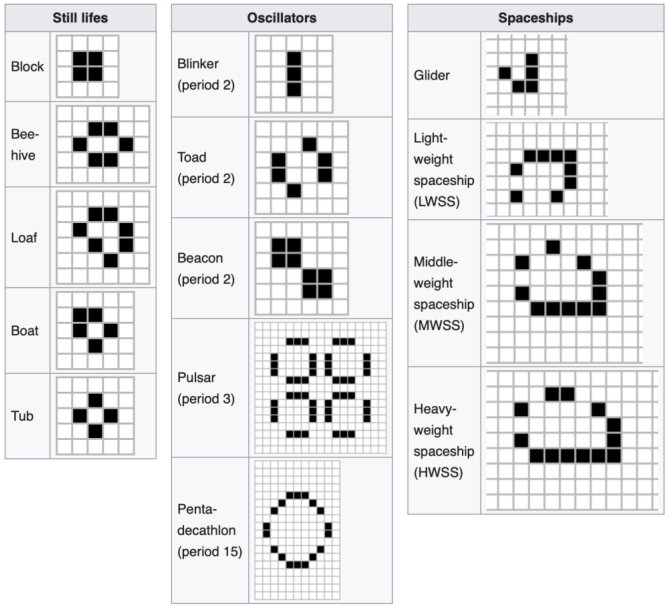

Figure 1: Ejemplos de patrones comunes

[1] https://en.wikipedia.org/wiki/Conway's_Game_of_Life

2 Enunciado

Se va a modelar el juego de la vida descrito anteriormente pensando a cada célula como un proceso que reside en un nodo de una red. Las células conocerán cuales son sus vecinas a través del servidor, quién además mandará una señal de clock de manera que en cada "tick" marque el comienzo de un turno.

Cuando las células actualicen su estado se lo comunicarán al servidor para que pueda mostrar la información global. Inicialmente el juego requerirá que al menos 9 células se hayan conectado pero el servidor deberá poder escuchar pedidos de nuevas células.

A continuación se describe el protocolo de cada uno de las entidades anteriores.

2.1 Clientes (células)

Antes de que un proceso célula pueda ser parte del juego debe:

1. Conectarse al servidor.
2. Registrarse en el servidor con información necesaria para que el resto de las células pueda comunicarse con ella.
3. Esperar a recibir del servidor el conjunto de vecinos.
4. Conectarse a sus vecinos.
5. Enviar un mensaje del tipo CLIENT_READY al servidor.
6. Comenzar el ciclo del juego.

A su vez dentro del ciclo del juego, las células recibirán del servidor mensajes del tipo TICK o NEW_CLIENTS.

En caso de un TICK deberán recibir de sus vecinos el estado previo a la actualización del turno por medio de mensajes de tipo CLIENT, computar el nuevo estado a partir de dicha información e informar al servidor el nuevo estado con un mensaje del tipo CLIENT_STATE.

En el caso de NEW_CLIENTS deberán incorporar los vecinos especificados en el mensaje. Notar que inicialmente las células siguen un esquema de cliente-servidor tradicional para conseguir la información acerca de la topología de la red para luego conectarse en un esquema peer-to-peer con el resto de sus vecinos.

OPCIONAL: Una variante es pensar que las células van a formar un grafo conectándose con sus vecinas según la topología que les indique el servidor. Dicha estructura no necesariamente tiene que ser matricial.

2.2 Servidor

El servidor será quién coordine el establecimiento de las conexiones entre las células de acuerdo a la estructura de grilla descrita previamente. Cuando haya 9 células conectadas, dará comienzo al juego enviándoles la información sobre sus vecinos con un mensaje del tipo NEW_CLIENTS. Tener en cuenta que siempre pueden llegar nuevas células, y que en ese caso el servidor las incorporará solo cuando haya suficientes para poder agrandar la matriz en ambas dimensiones. Por ejemplo, si se tiene una matriz de 3x3 la próxima incorporación se realizara cuando lleguen 7 células dispuestas en forma de "L" apoyada en la esquina izquierda superior o esquina derecha inferior. Es importante que al anexar nuevas células se actualicen los vecinos de las existentes. En cualquier caso, luego de notificar la incorporación de nuevos vecinos el servidor debe esperar la confirmación de los mensajes de tipo CLIENT_READY

Además brindará un servicio de timer que publicará mensajes del tipo TICK cada cierto intervalo de tiempo. Esto ocurrirá cuando lleguen al menos 9 células. Es importante que las células que pudieron haberse agregado entre ticks sean tenidas en cuenta.

A su vez el servidor debe permitir visualizar en cada tick el estado de todas las células.

2.3 Mensajes

Los clientes y el servidor deben interactuar con mensajes que tienen un tipo. Para eso se modelará un objeto request que tenga un campo tipo y mensaje (este último en algunos quedará como "padding" y en otros contendrá, por ejemplo, la lista de vecinos de un cliente).

Opcionalmente, los mensajes del tipo client pueden ser modelados como un objeto client request con campos tipo = CLIENT y un char que indicará el estado (vivo/muerto) de la célula.

2.4 Implementación

Se pide diseñar e implementar en C/C++ la variante propuesta del juego de la vida. Para comunicar procesos se debe utilizar UNIX Sockets teniendo en cuenta que cada proceso puede correr en un host distinto.

En caso de necesitar implementar algún patron de sincronización, se sugiere utilizar POSIX semaphores o bien la clase mutex provista en C++. El programa no debe tener race conditions, data races ni deadlocks.

Para guiar el desarrollo, es aconsejable seguir el esqueleto de los archivos provistos y crear tests que vayan garantizando el correcto funcionamiento de las funciones básicas. En el archivo utils.cpp se deberán implementar funciones comunes a ambas entidades, y en particular puede resultar conveniente encapsular lo más posible la interacción con la API de sockets. Por ejemplo definir una función *get_request* que dado un socket descriptor espere a recibir un objeto request y lo devuelva.

Opcionalmente, se pueden definir clases para las entidades cliente y servidor. En cualquier caso, deben proveer una forma de lanzar N clientes y un servidor desde un único ejecutable.

3 Entrega

La entrega del tp deberá ser subida al *Classroom* en la tarea correspondiente. El formato de entrega será un archivo zip con el apellido de los integrantes del grupo, que **no puede exceder las dos personas**. Se deberá incluir un link al repositorio cuyo README consignen algunas observaciones que crean necesarias sobre su implementación (en particular, aquellas referidas a la forma de correrla adecuadamente).

La fecha limite propuesta para la entrega final es el **31 de Octubre** hasta las 23:59:59. Cualquier entrega fuera de tiempo no será tenida en cuenta.

Los siguientes ejes serán evaluados:

- **Implementación**: Correctitud. Legibilidad del código y claridad. Uso de recursos vistos en clase.
- **Defensa oral**: Luego de la entrega final, y en fechas a confirmar, los grupos que hayan obtenido como nota final ocho o mas, podrán acceder a la defensa oral individual para "promocionar " este tema en el parcial. Durante la misma se discutirán, con cierta generalidad, distintos aspectos del trabajo tanto de implementación como experimentación.

References

[1] Berlekamp, E. R.; Conway, John Horton; Guy, R. K. (2001–2004). Winning Ways for your Mathematical Plays (2nd ed.). A K Peters Ltd.