Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2021-2022

Práctica 1: El juego de la Vida

1. Objetivo

En esta práctica se implementan y usan tipos de datos definidos por el usuario en lenguaje C++.

2. Entrega

Se realizará en dos sesiones de laboratorio en las siguientes fechas:

Sesión tutorada: del 21 al 25 de febrero de 2022
Sesión de entrega: del 7 al 11 de marzo de 2022

3. Enunciado

Un autómata celular [1] es un modelo matemático y computacional para un sistema dinámico que evoluciona en pasos discretos. Es adecuado para modelar sistemas naturales que puedan ser descritos como una colección masiva de objetos simples que interactúan localmente.

No existe una definición formal aceptada de autómata celular. Sin embargo, se puede describir como una tupla de objetos caracterizado por los siguientes componentes:

- Una **rejilla**, o cuadrícula, de enteros infinitamente extendida con dimensión entera positiva. Cada celda se conoce como **célula**.
- Cada célula puede tomar un valor entero a partir de un conjunto finito de estados.
- Cada célula se caracteriza por su **vecindad**, un conjunto finito de células en las cercanías de la misma.
- De acuerdo con esto, se aplica a todas las células de la cuadrícula una **función de transición** que toma como argumentos los valores de la célula en cuestión y los valores de sus vecinos, y retorna el nuevo estado que la célula tendrá en la siguiente etapa de tiempo.

El juego de la vida [2] es un ejemplo de autómata celular diseñado por el matemático británico John Horton Conway en 1970. Está caracterizado por una rejilla en dimensión dos. Cada célula puede estar en uno de los dos estados posibles: “viva" o "muerta"; y la vecindad se corresponde con las ocho células adyacentes en la cuadrícula.

El juego de la vida es un juego de cero jugadores, lo que quiere decir que su evolución está determinada por el estado inicial y no necesita ninguna entrada de datos posterior. La **rejilla** se extiende hasta el infinito en todas las direcciones. El estado de las células evoluciona en unidades de tiempo discretas, que denominaremos **turnos**.

En un turno cada célula de la rejilla actualiza su estado en función de su valor de estado y del estado de sus ocho células vecinas en el turno anterior. El cambio de estado dependerá del número de células vecinas vivas según las siguientes **reglas de transición**:

- Una célula “muerta” con exactamente 3 células vecinas con estado “viva” pasa al estado “viva” en el siguiente turno. En cualquier otro caso permanece “muerta”

- Una célula “viva” con 2 ó 3 células vecinas con estado “viva” continúa “viva” en el siguiente turno. En cualquier otro caso pasa al estado “muerta”.

4. Notas de implementación

1. El objeto célula, `Cell`, que representa la presencia o ausencia de vida en una posición del tablero mediante su atributo estado con los valores: “viva” o “muerta”.

    a. El estado es un atributo privado de la célula al que se accede mediante los métodos:

    ```
    State Cell::getState() const;
    State Cell::setState(State);
    ```

    b. La posición `(i,j)` que ocupa una célula en la rejilla también es un atributo privado.

    c. Cada célula es responsable de actualizar su estado, cuando le corresponda en el turno actual, siguiendo para ello las reglas de transición que definen el juego. Para ello implementa un método público:

    ```
    void Cell::updateState();
    ```

    d. Puesto que las reglas de transición se definen en función del número de células vecinas vivas en su vecindad, cada célula debe interactuar con sus 8 células vecinas para consultarles sus estados. Para ello dispone de un método público que recibe como parámetro la rejilla:

    ```
    int Cell::neighbors(const Grid&);
    ```

    e. Las 8 células vecinas de la célula `(i,j)` son las que ocupan las posiciones en la rejilla:

    ```
    (i-1,j-1)|(i-1,j)|(i-1,j+1)
    --------------------------
      (i,j-1)|   (i,j)|   (i,j+1)
    --------------------------
    (i+1,j-1)|(i+1,j)|(i+1,j+1)
    ```

    f. Una célula es responsable de su visualización en pantalla, utilizando el carácter `'X'` para representar el estado “viva”, y el carácter blanco `' '` para el estado “muerta”. Para ello se sobrecarga el operador de inserción en flujo.

    ```
    ostream& operator<<(ostream&, const Cell&);
    ```

2. El objeto rejilla, `Grid`, contiene un array de `NxM` células. Este objeto es responsable de crear y almacenar las células que constituyen el juego y de establecer su valor de estado inicial.

    a. El constructor recibe el tamaño de la rejilla y crea el array de células en memoria dinámica. Cada célula se inicia con la posición `(i,j)` que ocupa en la rejilla y el valor de su estado en el inicio. Todas las células están inicialmente en estado “muerta”, salvo aquellas que el usuario indique en tiempo de ejecución que deben estar en estado “viva” en el turno inicial.

    b. La rejilla da acceso de sólo lectura a cualquiera de sus células.

    ```
    const Cell& Grid::getCell(int, int) const;
    ```

c. El objeto rejilla es responsable de contar las unidades de tiempo, **turnos**, de forma que en cada turno todas las células actualizan su estado a partir de los valores del estado de las células en el turno anterior. Esta operación se implementa en el método:

```
void Grid::nextGeneration();
```

d. Para garantizar que la actualización de las células en cada turno tiene en cuenta los valores del turno anterior, la rejilla se recorre dos veces. En el primer recorrido cada célula cuenta sus vecinas; y en el segundo recorrido cada célula actualiza su estado.

e. En cada turno se realiza un tercer recorrido de la rejilla para que cada célula se muestra en pantalla de la siguiente forma:

Turno 0:

```
XX  XX
XX  X X XXX
      X
```

f. Hay que tener en cuenta que las células del borde de la rejilla no cumplen la condición de tener 8 células vecinas. Para evitar la complicación en el cómputo de las condiciones de frontera se creará una rejilla con las dimensiones `(N+2)x(M+2)`, pero sólo se actualizan y visualizan las `NxM` células del interior de la rejilla, que son las que cumplen la condición de tener las 8 células vecinas.

3. El funcionamiento del programa principal es el siguiente:

   a. Solicita el tamaño de la rejilla, N y M, y el número máximo de turnos que tendrá el juego.

   b. Crea un objeto `Grid` con todas sus células en estado “muerta”.

   c. Solicita las posiciones de las células que deben estar en estado “viva” en el turno 0 y actualiza el estado en dichas células de la rejilla.

   d. Muestra por pantalla el estado de la rejilla.

   e. Ejecuta un bucle donde cada iteración se corresponde a un turno en la evolución de juego. En cada turno se actualizan y se muestran por pantalla el estado de las células en la rejilla.

4. Durante las sesiones de laboratorio se podrán solicitar cambios y/o ampliaciones en la especificación de este enunciado.

5. Referencias

[1] Wikipedia: https://es.m.wikipedia.org/wiki/Autómata_celular
[2] Wikipedia: https://es.m.wikipedia.org/wiki/Juego_de_la_vida