Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2023-2024

Práctica 2: El juego de la Vida

---

1. Objetivo

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

2. Entrega

Se realizará durante la sesión de entrega en el laboratorio entre el 26 de febrero y el 1 de marzo.

3. Enunciado

El juego de la vida [2] es un ejemplo de autómata celular diseñado por el matemático británico John Horton Conway [3] en 1970. Siguiendo la caracterización de autómata celular dada en el enunciado de la práctica anterior [4], podemos describir el juego de la vida con los siguientes componentes:

- El espacio celular se define mediante un retículo bidimensional, un tablero donde cada casilla es una célula.
- El alfabeto contiene dos estados posibles para cada célula: «viva» o «muerta».
- La vecindad, denominada vecindad de Moore [7], está constituida para cada célula por sus ocho células adyacentes sobre el tablero.
- La función de transición local, que cada célula aplica para calcular su estado en la siguiente generación, depende de su estado actual y del número de células con valor de estado «viva» en su vecindad. La versión original del juego utiliza la regla de transición denotada como “23/3”, que consiste en:
    - Una célula en estado «viva» con 2 ó 3 células vecinas en estado «viva» continúa en estado «viva» en la siguiente generación. En otro caso pasa al estado «muerta».
    - Una célula en estado «muerta» con exactamente 3 células vecinas en estado «viva» pasa al estado «viva» en la siguiente generación. En otro caso permanece en estado «muerta».
- Las condiciones de frontera, además de las descritas en la práctica anterior [4]: frontera abierta, frontera periódica y frontera reflectora; se incluye un nuevo tipo de condición denominado “Sin frontera” [1].
    - Se define un retículo que cambia de tamaño de forma dinámica cada vez que alguna célula con estado «viva» en el borde necesita interactuar con sus vecinas.

El juego de la vida pertenece a la categoría de **juego de cero jugadores**, lo que quiere decir que su evolución en sucesivas generaciones está determinada únicamente por la configuración inicial, y no necesita ninguna entrada de datos posterior. En [2] se describen algunos de los comportamientos observados al utilizar determinadas configuraciones iniciales de células: osciladores, vidas estáticas, naves espaciales, matusalenes, etc.

4. Notas de implementación

En esta práctica se mantienen las especificaciones de las clases dadas en la práctica anterior [4] para la implementación del autómata celular elemental. También se utiliza la sobrecarga de operadores para implementar algunas de las operaciones.

1. La clase célula, `Cell`, representa la presencia o ausencia de vida en una posición del tablero. Mantiene la especificación vista en la práctica anterior, siendo responsable de encapsular su estado y posición, de conocer su vecindad y la función de transición local, y de actualizar su estado para la siguiente generación. Hay que tener en cuenta las siguientes consideraciones:

    a. En un espacio bidimensional cada posición del retículo se identifica con un par de coordenadas enteras `(coordenada_0,coordenada_1)`. Por tanto, hay que redefinir el tipo de dato `Position` para contener esta pareja de coordenadas. Para ello se puede declarar una nueva clase o se puede utilizar el tipo `std::pair` [5] definido en la librería `<utility>` de la STL [6].

    b. En la vecindad de Moore cada célula, con posición `(i,j)`, tiene 8 vecinas que ocupan las siguientes posiciones en el tablero:

```
(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)
```

    c. La visualización en pantalla utiliza el carácter `'X'` para representar el estado «viva», y el carácter espacio `' '` para el estado «muerta». Se sobrecarga el operador de inserción en flujo.

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

2. La clase retículo, `Lattice`, es el objeto responsable de crear y almacenar las células que forman parte del juego. En este caso se utilizará un array bidimensional de `MxN` células. También es responsable de establecer el estado de cada célula en la configuración inicial.

    a. Como en el caso del autómata celular elemental, el retículo dispone de un constructor que crea las células en memoria dinámica, con valor inicial de estado «muerta». Este constructor se apoya en un método auxiliar para solicitar por teclado las posiciones de las células que deben estar vivas en la configuración inicial.

```
Lattice::Lattice(int N, int M);
```

    b. El retículo implementa un segundo constructor que recibe como parámetro el nombre de un fichero. La primera fila del fichero de texto contiene el número de filas (`M`) y columnas (`N`) del tablero. A continuación contiene las M cadenas de N caracteres, donde `' '` indica una célula «muerta» y `'X'` indica una célula «viva».

```
Lattice::Lattice(const char*);
```

c. Se añade a esta clase la responsabilidad de conocer la **población**. Esto es, el número de células en estado «viva» en la generación actual.

```
std::size_t Lattice::Population() const;
```

d. Para dar acceso a las células por la posición que ocupa en el retículo se sobrecarga el operador.

```
Cell& Lattice::operator[](const Position&) const;
```

e. En la implementación de la condición de frontera "Sin frontera" el tablero se crea inicialmente con un tamaño dado. Cuando tras un cambio de generación alguna de las células del borde pasa a estado «viva», se incrementa el tamaño del tablero creando una nueva fila, o columna, de células en estado «muerta» que será el nuevo borde en la siguiente generación. Para contener las células del tablero se utilizará una estructura de datos dinámica, como `std::vector`, que permitirá cambiar el tamaño de la estructura durante la ejecución. Hay que tener en cuenta que al añadir elementos por principio del vector se cambia la posición de todos los elementos que contiene.

3. El funcionamiento del programa principal es el siguiente:

a. Recibe por línea de comandos los siguientes argumentos:

`-size <M> <N>`, `M` es el número de filas y `N` es el número de columnas del tablero.

`-init <file>`, (opcional) `file` es un nombre del fichero que contiene los valores iniciales para el estado de las células del tablero.

`-border <b>`, `b=reflective`
`b=noborder`

b. Crea e inicializa los componentes del juego de la vida. Los argumentos -size e -init son excluyentes.

Con el argumento `-size` se utiliza el primer constructor del retículo.
Con el argumento `-init` se utiliza el segundo constructor del retículo.

c. Realiza la evaluación en generaciones del juego, mostrando en cada generación el tablero por pantalla. El usuario controla la simulación con los siguientes comandos introducidos por teclado:

'x' : Finaliza la ejecución del programa
'n' : Calcula y muestra la siguiente generación
'L' : Calcula y muestra las siguientes cinco generaciones
'c' : Los comandos 'n' y 'L' dejan de mostrar el estado del tablero y sólo se muestra la población, esto es, el número de células en estado «viva»
's' : Salva el tablero a un fichero

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

5. Referencias

[1] Wikipedia: Autómata celular
https://es.m.wikipedia.org/wiki/Autómata_celular

[2] Wikipedia: El juego de la vida
https://es.m.wikipedia.org/wiki/Juego_de_la_vida

[3] Wikipedia: John Horton Conway
https://es.m.wikipedia.org/wiki/John_Horton_Conway

[4] Moodle: Enunciado práctica 1 de AyEDA
https://campusingenieriaytecnologia2324.ull.es/mod/resource/view.php?id=18524

[5] cpluplus: Enlace a la descripción del tipo `std::pair`
https://cplusplus.com/reference/utility/pair/

[6] cpluspus: Referencia a la Standard Template Library
https://cplusplus.com/reference/

[7] Wikipedia: Vecindad de Moore
https://es.wikipedia.org/wiki/Vecindad_de_Moore