Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

Grado en Ingeniería Informática
Algoritmos y Estructuras de Datos Avanzadas

Curso 2019-2020

Práctica 3: El juego de la Vida Generalizado

1. Objetivo

El objetivo de la práctica es comprender el funcionamiento de la herencia y del polimorfismo dinámico en el lenguaje C++.

2. Entrega

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

Sesión tutorada: del 16 al 19 de marzo de 2020
Sesión de entrega: del 23 al 26 de marzo de 2020

3. Enunciado

En la versión estándar del juego las transiciones una célula sólo dependen del número de células vecinas vivas:

[Regla de supervivencia]: Una célula viva con 2 ó 3 células vecinas vivas sigue viva, en otro caso muere.
[Regla de nacimiento]: Una célula muerta con exactamente 3 células vecinas vivas "nace".

Estas reglas de transición se denotan como "23/3" donde el primer número, o lista de números, indica el número de células vecinas vivas que requiere una célula para que siga viva (regla de supervivencia); y el segundo número, o lista de números, indica el número de células vecinas vivas que requiere una célula para nacer (regla de nacimiento).

Se han desarrollado variantes del juego que utilizan distintas reglas de transición. A continuación se presentan algunas de las variantes estudiadas [1], aunque casi todas son demasiado caóticas o demasiado desoladas:

/3	(estable)	casi todo es una chispa
5678/35678	(caótico)	diamantes, catástrofes
1357/1357	(crece)	todo son replicantes
1358/357	(caótico)	un reino equilibrado de amebas
23/3	(caótico)	"Juego de la Vida de Conway"
23/36	(caótico)	"HighLife" (tiene replicante)
235678/3678	(estable)	mancha de tinta que se seca rápidamente
245/368	(estable)	muerte, locomotoras y naves
34/34	(crece)	"Vida 34"
51/346	(estable)	"Larga vida" casi todo son osciladores

En esta segunda práctica se implementará una nueva versión del juego de la vida en la que coexistan, de forma simultánea sobre el mismo tablero, distintos tipos de células con distintas reglas de supervivencia.

Mediante herencia de la clase base `Celula` se definen los distintos tipos de células (1, 2, 3, …), donde cada tipo de célula implementa una reglas de transición propias. De esta forma se genera una jerarquía de clases con métodos polimórficos que implementarán un comportamiento diferente en cada clase derivada.

Se requiere implementar, al menos, los siguiente tipos de células:

Célula 1: 23/3
Célula 2: 245/368
Célula 3: 51/346

En el tablero, que contiene una malla de punteros a células base, puede alojar los distintos tipos de tipos de células.

Durante las sesiones de laboratorio se podrán proponer modificaciones y mejoras en el enunciado.

4. Notas de implementación

- En la clase `Celula`, implementada en la práctica anterior, modificar los siguientes métodos.

```
class Celula {
...
  public:
  static Celula* createCelula(int, int, int);
  virtual int getEstado() const;
  virtual int contarVecinas(const Tablero&);
  virtual int actualizarEstado();      // Reglas de nacimiento
  virtual ostream& mostrar(ostream&) const;
...
};
```

- Las células muertas se representarán mediante objetos de la clase base `Celula`. De esta forma, las reglas de nacimiento para todos los tipos de células sólo se implementan en el método `actualizarEstado()` de la clase base. Cuando una célula muerta deba nacer por la aplicación de una regla de nacimiento, el método `actualizarEstado()` retornará un valor entero (`1`, `2`, `3`, …) que identifica al tipo de célula que debe nacer.

- Los distintos tipos de células vivas se representan mediante las clases que derivan de la clase base `Celula`. En cada clase derivada se redefinen los comportamientos para actualizar y mostrar la célula según el tipo al que representa.

```
class Celula1: public Celula {
...
  public:
  int actualizaEstado();              // Reglas de supervivencia
  stream& mostrar(ostream&) const;
...
};
```

- El método `actualizarEstado()` de una clase derivada sólo implementa la regla de supervivencia para el tipo de célula que representa.

- Cuando una célula viva deba morir por la aplicación de su regla de supervivencia, el método `actualizarEstado()` retornará el valor `0` para indicar que debe sustituirse por una célula muerta.

- El valor entero retornado por el método `actualizarEstado()` de cada célula de la malla informa al objeto tablero del tipo de célula que debe existir en dicha posición de la malla para el siguiente turno. Es responsabilidad de la clase `Tablero` sustituir dicha célula, destruir la antigua y crear la nueva, cuando sea necesario cambiar su tipo.

- Se añade el método de clase `createCelula()` que realiza la creación en memoria dinámica de una célula de cualquiera de los tipos. Este método recibe un parámetro que indica el tipo de célula (`0` para una célula base, `1`, `2`, `3`, … para las células derivadas) a crear en memoria dinámica y retorna un puntero a la célula creada. Para evitar el posible conflicto cuando en una posición de la malla se cumplan simultáneamente las reglas de nacimiento para varios tipos de célula, este método establece la siguiente prioridad de creación: `1`, `2`, `3`, …

- En esta nueva implementación el estado de una célula se define únicamente por el tipo de objeto que la representa. Por tanto, se puede eliminar el atributo `Estado` en la definición de la clase base `Celula` y hacer que el valor devuelto por el método `getEstado()` sólo dependa del tipo de célula.

- Cada tipo de célula derivada se visualiza utilizando un carácter distinto que indica su tipo (`'1','2','3'`).Las células muertas, del tipo base, se visualizan con un carácter blanco `' '`. Se añade a la clase base el método polimórfico `mostrar()`, que en cada tipo de célula muestra el carácter que corresponda.

- El método de la clase `Tablero` que inicializa el estado en el turno 0, estado inicial, no sólo solicitará al usuario las posiciones donde habrá vida inicialmente, también solicita un tipo para cada una de estas células, las crea y sustituye en la correspondiente posición del tablero.

- El método de la clase `Tablero` que controla el cambio de turno debe realizar tres pasos:

    1. Cada célula del tablero inspecciona a sus vecinas mediante su método `contarVecinas()`.

    2. Cada célula del tablero actualiza su estado, mediante su método `actualizarEstado()`, y retorna un código que informa al tablero si debe sustituir a dicha célula (destruir la célula actual y crear una nueva).

    3. Cada célula del tablero se muestra en pantalla, mediante su método `mostrar()`, para visualizar la evolución del juego.

5. Referencias

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