Informe Trabajo Práctico 2 JuegoDeLaVida 2.0

Grupo Life Compiler

Integrantes	Padrón
Julián Morales	106999
Zoilo Pazos	107740
Tapia Francisco	107128
Villarinos Luca	104975

Objetivo

Realizar una aplicación que lleve adelante la ejecución del juego de la vida en un tablero tipo cubo definido por usuario, de n filas por m columnas por l planos, a partir de una configuración inicial de células vivas inidcadas por el usuario. Adicionalmente a la versión desarrollada en el TP1 se agregan funcionalidades al juego: * Posibilidad de elegir la dificultad del juego * Comportamientos de las celdas del juego que modifican la dificultad del juego

Herramientas usadas

Para la realización de la estructura de trabajo como equipo en primera instancia desarrollamos el código en el lenguaje C++. Por cuestiones de compilador incorporado al IDE, utilizamos VSCode. De todas formas todo lo que desarrollamos en este IDE también lo probamos en Eclipse con los flags correspondientes. Para sincronizar el trabajo y tener un repositorio en común decidimos utilizar Github. Con el fin de evitar perdidas de memoria en el código utilizamos Valgrind. Nos resultó de gran utilidad ya que en algunos casos no eramos conscientes de que estabamos dejando leaks de memoria. A pesar de esto, detecta dos bloques de perdidas de memoria que se deben a la función remove(). Estas creemos que no son realmente perdidas, pero la verificación de esto escapa de nuestras manos.

Estructura

Para la realización del código decidimos establecer y usar clases para simular el comportamiento real de los distintos objetos que interactúan en el juego. Por esa razón establecimos las clases Nodo, ListaDoblementeEnlazada, Celula, Celda, Tablero, JuegoDeLaVida, Configuraciones. La división fundamental que planteamos es que todas las clases funcionan como contenedores de información en su generalidad. En cambio, la clase JuegoDeLaVida contiene en sus métodos las funciones para orquestrar el funcionamiento del juego. A continuación una breve descripción de los aspectos importantes a tener en cuenta para el desarrollo de las clases:

- **Nodo**: Como decidimos hacer una lista doblemente enlazada la clase Nodo con diferencia a la ofrecida en el Github de la cátedra también tiene un atributo puntero anterior. Esta implementada en forma de template.
- **ListaDoblementeEnlazadaCircular**: Para el fácil acceso a los vecinos de las células decidimos que los mejor sería una lista doblemente enlazada y circular ya que permite moverse sobre ella en ambas direcciones. A través de la sobrecarga de el operador indexación [] logramos que el acceso a las vecinas de una célula sea mucho mas rapido que O(n), sino que si se accede a la misma lista con la diferencia en 1 es O(1). El método esta implementado como un template.
- **Tablero**: El tablero tiene la funcionalidad de formar una estructura tridimensional simulada a traves de la anidación de objetos de lista y establece el tipo de dato contenido como una celda. Tiene los métodos para obtener una celda y para mostrar a través de capas por la terminal el estado de las células del juego y las estadísticas.
- **JuegoDeLaVida**: Contiene instancias de el resto de Clases desarrolladas, fundamentalmente sincroniza los turnos y la lógica de vida o muerte. Además contiene la lógica de como se pasan los genes y como los comportamientos de las celdas afectan al juego. Tiene un método particular que a traves del módulo EasyBMP muestra el estado del juego y las estadísticas en tantos archivos como capas hayan en el juego y una adicional para las estadísticas.
- **Configuraciones**: Esta clase tiene la funcionalidad de ser reutilizable (del modo que esta implementada no lo es, pero con algunos cambios podría serlo) ya que su función principal es la gestión de distintas dificultades y la carga de estas a través de archivos. La dificultad del juego se define a través de un Struct Dificultad que solo contiene datos.

Cuestionario

1. ¿Qué es "SVN"?

   SVN o Apache Subversion es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Es software libre bajo una licencia de tipo Apache/BSD. Una herramienta de control de versiones se encarga de generar versiones de un producto, programa o los elementos del mismo. Estas versiones tienen la funcion de representar al producto en distintos estados según el momento de desarrollo en el que se encuentra. Esto permite poder acceder a versiones anteriores. SVN permite generar un repositorio digital y de fácil acceso desde distintas computadoras, lo que permite el trabajo en equipo ya que varias personas pueden modificar y administrar el mismo conjunto de datos desde su respectivas ubicaciones.

2. ¿Qué es un "Git" y "Github"?

   Un Git es un software de control de versiones, similarmente a el SVN permite el trabajo en equipo a través del control de versiones, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.
   Github es un sitio web que permite crear proyectos y sus respectivos repositorios de git, y que estos sean accesibles de forma remota. Ya sea para la exposición publica de un proyecto Open Source o para el trabajo en conjunto.

3. ¿Qué es "Valgrind"?

   Valgrind en un software Open Source que se utiliza para el Debugging de problemas de memoria y rendimiento de programas. Durante el trabajo nosotros utilizamos Valgrind para revisar las fugas de memoria que tenia nuestro código. También Valgrind ofrece la detección de los siguientes errores:

- Uso de memoria no inicializada
- Lectura/escritura de memoria que ha sido previamente liberada
- Lectura/escritura fuera de los límites de bloques de memoria dinámica

Manual de usuario JuegoDeLaVida 2.0

Grupo Life Compiler

Descripción

El juego de la vida es en realidad 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.

En esta versión modificada el "tablero de juego" es de tres dimensiones, formado por "células", que en nuestro caso estará limitado a NxMxL (configurado al inicio).

Cada célula tiene 26 células vecinas, que son las que están próximas a ella, incluso en las diagonales y al llegar al lateral del cubo, continua por el lado opuesto. Las células tienen dos estados: están "vivas" o "muertas". El estado de la malla evoluciona a lo largo de unidades de tiempo discretas (se podría decir que por turnos).

El estado de todas las células se tiene en cuenta para calcular el estado de las mismas al turno siguiente. Todas las células se actualizan simultáneamente. Las transiciones dependen del número de células vecinas vivas:

- Una célula muerta con exactamente X1 células vecinas vivas "nace" (al turno siguiente estará viva). El valor de X1 será dado al inicio.
- Una célula viva con X2 a X3 células vecinas vivas sigue viva, en otro caso muere o permanece muerta (por "soledad" o "superpoblación"). Si la célula está inerte no cuenta.

Análisis de partes del juego

1. El espacio sera un cubo con tres dimensiones n,m,l.
2. Cada célula tiene 26 células vecinas que continúan hasta el lado opuesto, es decir, no existen limites.
3. Las células tienen dos estados: a. Vivas b. Muertas
4. Las células y los casilleros son independientes.
5. El juego tiene que ofrecer las siguientes opciones: a. Ejecutar uno o varios turnos. b. Reiniciar el juego. c. Terminar.

Configuración del juego definidas por el usuario

1. Dificultad del juego: habrán algunas cargadas por defecto. También se podra personalizar.
2. Cantidad y posición de células a inicializar: el usuario dará las coordenadas de las células que comenzarían vivas, indicando x, y, z. O si no quiere hacerlo, existe la opción de que la mitad de las células comiencen vivas en posiciones aleatorias.

Dificultades del juego

Las dificultades del juego son dadas a partir de los siguientes parámetros: * Dimensión del tablero (filas x columnas x capas). * Requisitos para que una célula nazca, muera o siga viva si ya lo estaba. * Cantidad de celdas con cada comportamiento (Contaminada, Envenenada, Procreadora, Portal, Radioactiva).

Comportamientos de las celdas

- Contaminada: No permite que nazca una célula.
- Envenenada: Mata un gen (Cambia su valor a 0)
- Procreadora: Modifica el numero de vecinas vivas necesarias para nacer.
- Portal: Es la entrada de un portal. Tendrá una celda "salida de portal". Si en la celda entrada nace una célula, nace otra en la salida.
- Radioactiva: Afecta a uno o varios genes al heredarse. Cada gen tiene una probabilidad del 50% de ser afectado. Si es afectado se reduce a la mitad

Reglas

Se establecen las siguientes reglas: 1. Cada célula tiene 3 genes, con una carga genética por cada gen de entre 0 y 255. Al

generar una célula, sus genes se calculan en base los genes de sus progenitores, de la siguiente manera (salvo al inicio del juego que, al no haber progenitores, se asignan valores aleatorios): * Gen 1: es un promedio entre los genes 1 de los progenitores. * Gen 2: es el máximo de los genes 2 de los progenitores. * Gen 3: es la suma de los genes 3 de los progenitores, y luego se le resta 255 hasta que sea menor o igual a dicho valor.

2. Con la ejecución de cada turno se podra decidir si se quiere ejecutar otro, reiniciar el juego o salir.

Interfaz Gráfica

El juego va a estar representado graficamente a través de estos dos métodos: * Por terminal: Basado en texto se representará las estadísticas del juego y estado de las células a través de caracteres. No se representarán comportamientos de celdas, ni colores de células basados en los genes. Solo se muestra si la célula está viva o muerta. * Por medio de archivos.bmp: Se generaran imégenes con formato bmp de forma que cada una de estas representará una capa del "tablero" y una extra con las estadísticas del juego. En una capa se podrá ver cada celda diferenciando a las que tengan un comportamiento distinto del normal con la letra inicial del mismo a exepción de las procreadoras que estarán representadas por la letra 'A'. En la celda tambien se mostrará la célula contenida si esta viva a través de un circulo del color dado por los genes de la misma. Las imágenes se almacenarán en la carpeta llamada "imágenes", dentro de la carpeta raiz del juego.

Cada vez que se inicie el juego se limpiara la carpeta de imágenes para agregar las nuevas.

Manual de programador JuegoDeLaVida 2.0

Grupo Life Compiler

Estructura general

El JuegoDeLaVida esta desarrollado en el lenguaje C++. Utilizando la caracteristica principal del lenguaje de POO (Programación Orientada a Objetos). Por esta razón decidimos dividir la estructura de el juego en TDAs contenedores de datos y un TDA principal JuegoDeLaVida que se encargue de la lógica de como interactuan los demás. Por esta razón diseñamos los siguientes TDAs: * Nodo * ListaDoblementeEnlazada * Célula * Celda * Tablero * JuegoDeLaVida * Configuraciones

Diseño de funciones y relaciones entre TDAs

Nodo

El TDA nodo esta diseñado como un template y continene un nodo siguiente y un nodo anterior

> *Las distintas funciones de la clase estan comentadas en el código Nodo.h*

ListaDoblementeEnlazada

El TDA esta diseñado como un template. Contiene elementos almacenados de forma secuencial y ofrece la facilidad de navegar en ambas direcciones a traves de sus elementos.

> *Las distintas funciones de la clase estan comentadas en el código ListaDoblementeEnlazada.h*

Configuraciones

El TDA tiene una lista de configuraciones que se obtienen de un archivo txt.

> *Las distintas funciones de la clase estan comentadas en el código Configuraciones.h*

Célula

El TDA emula el comportamiento de una célula. Puede estar viva o muerta y tiene 3 genes que corresponden a un color RGB.

> *Las distintas funciones de la clase estan comentadas en el código Celula.h*

Celda

El TDA es un contenedor de un TDA Célula y tiene uno de los 5 comportamientos del juego.

> *Las distintas funciones de la clase estan comentadas en el código Celda.h*

Tablero

El TDA a traves de la estructura ListaDoblementeEnlazada simula la estructura tridimensional de un cubo.

> *Las distintas funciones de la clase estan comentadas en el código Tablero.h*

JuegoDeLaVida

El TDA tiene las funciones necesarias para que los demas TDAs interactuen entre si y para que el usuario pueda jugar el juego.

> *Las distintas funciones de la clase estan comentadas en el código JuegoDeLaVida.h*