Caso 1 – Infraestructura computacional

Integrantes:

**Santiago Celis Rengifo – 202111131**

**Cristian Camilo Rey Vargas – 202116752**

**Julián Escobar Rivera - 202111361**

Diagrama de Clases

Aplicación

Descripción generada automáticamente con confianza media

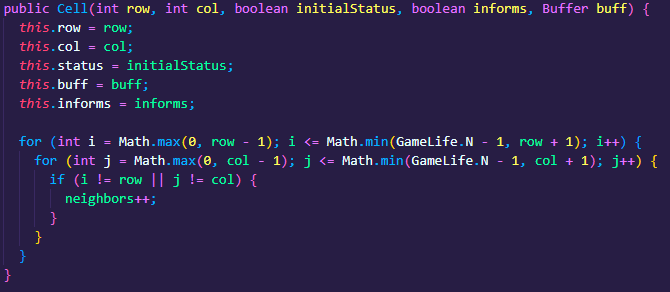
Funcionamiento del Programa

**Cell.java (esta clase extiende de Thread):**

**Texto

Descripción generada automáticamente**

Los atributos de la clase Cell son la fila y columna en la que se encuentra la célula, el estado de esta (viva o muerta - True/False), el buffer, la variable que determina si está informando su estado o no y, por último, la cantidad de células (celdas de la matriz) vecinas tiene.



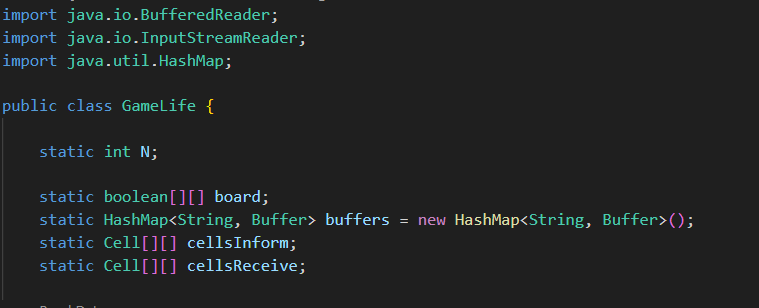
En el constructor se asignan los valores a cada variable dependiendo de los datos que se obtengan en la lectura del archivo de entrada. También, se recorre en un doble ciclo las demás casillas del alrededor para obtener el número de casillas vecinas para cada celda (este valor se almacena en la variable neighbors).

Texto

Descripción generada automáticamente

El método run() se divide principalmente en dos situaciones: la célula (celda) en cuestión está informando su estado o lo está cambiando. Cuando la célula está informando su estado se recorren todas las células vecinas y se informa al buffer de cada una de estas sobre el estado de la célula en cuestión (a excepción del buffer de la celda actual). Por otro lado, si una célula está por cambiar su estado, se recorren los valores del buffer para determinar cuántas células vivas hay alrededor. Posteriormente, dependiendo del resultado anterior se actualiza el estado de la célula actual teniendo en cuenta las reglas del juego.

**GameLife.java**



Importaciones

import java.io.BufferedReader: Importa la clase BufferedReader, la cual se utiliza para leer texto de una entrada de datos, como por ejemplo un archivo o la entrada estándar del sistema.

import java.io.InputStreamReader: Importa la clase InputStreamReader, que es un puente entre flujos de bytes y flujos de caracteres. Se utiliza para leer bytes y decodificarlos a caracteres.

import java.util.HashMap: Importa la clase HashMap, que es una estructura de datos que almacena elementos en pares clave-valor. Se utiliza para mapear claves únicas a valores.

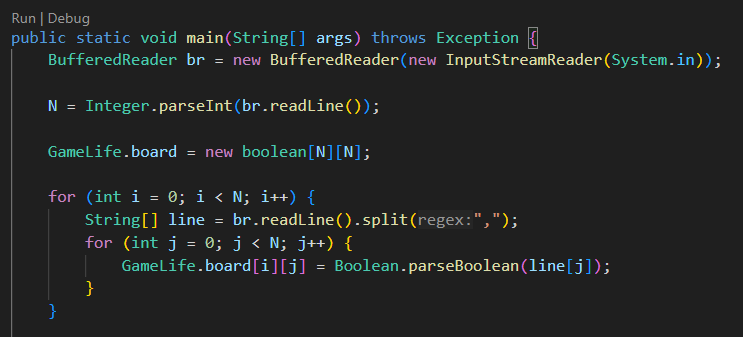
Variables inicializadas

N: Una variable compartida que define el tamaño del tablero de juego.

board: Una matriz bidimensional que representa el tablero con celdas vivas o muertas.

buffers: Un mapa que asocia cada posición en el tablero con un Buffer, utilizado para la comunicación entre celdas.

cellsInform y cellsReceive: Dos matrices que contienen instancias de Cell que gestionan el envío y recepción de estados de celdas a sus vecinas.



Este es el método main, en él se hace la inicialización del BufferedReader: bufferedReader br = new BufferedReader(new InputStreamReader(System.in));

Se crea un BufferedReader que envuelve un InputStreamReader. Este último lee bytes de la entrada estándar del sistema (System.in, normalmente el teclado) y los convierte en caracteres para que el BufferedReader los procese.

Lectura del tamaño del tablero:

N = Integer.parseInt(br.readLine()); La primera línea de la entrada se lee y se convierte de String a int, estableciendo el tamaño N del tablero.

Inicialización del tablero: gameLife.board = new boolean[N][N]; Se inicializa la matriz board de la clase GameLife con el tamaño leído anteriormente. Esta matriz representa el tablero del Juego de la Vida, donde cada celda puede ser true (viva) o false (muerta).

Rellenar el tablero con el estado inicial: en el bucle for que recorre las filas del tablero (for (int i = 0; i < N; i++)), cada línea del estado inicial del tablero se lee desde la entrada estándar.

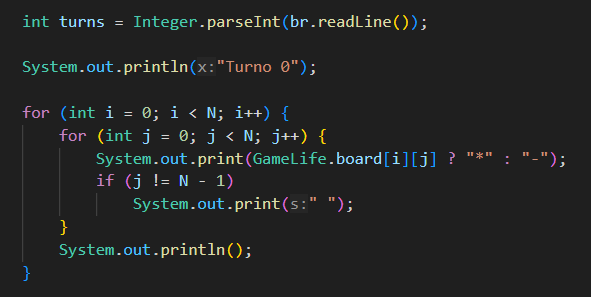
String[] line = br.readLine().split(",");

Cada línea leída se divide en un arreglo de String usando la coma , como delimitador.

Luego, otro bucle for recorre las columnas de la matriz (for (int j = 0; j < N; j++)).

GameLife.board[i][j] = Boolean.parseBoolean(line[j]);

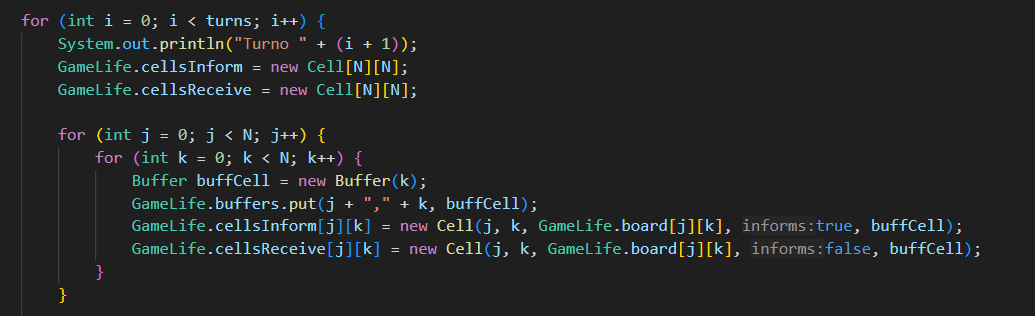
Se convierte cada elemento del arreglo de String a boolean y se asigna a la posición correspondiente en la matriz board.



Se lee el número de turnos para la simulación: int turns = Integer.parseInt(br.readLine()); Aquí se lee una línea del BufferedReader que se espera contenga el número de generaciones (o turnos) que se van a simular en el Juego de la Vida. Este valor se convierte de String a int y se almacena en la variable turns.

Se imprime el estado inicial del tablero: System.out.println("Turno 0");

Se imprime en la consola la cadena "Turno 0", que indica que se mostrará el estado inicial del tablero antes de comenzar la simulación. Luego, se usan dos bucles for anidados para recorrer cada celda del tablero (board). El primer bucle recorre las filas (i) y el segundo las columnas (j). Dentro del bucle más interno, se utiliza un operador ternario para imprimir un asterisco \* si la celda está viva (GameLife.board[i][j] es true) o un guion - si la celda está muerta (GameLife.board[i][j] es false). Después de imprimir el estado de cada celda, se comprueba si la celda actual no es la última de la fila (if (j != N - 1)). Si no lo es, se imprime un espacio en blanco para separar el estado de la siguiente celda en la misma fila. Finalmente, después de imprimir todos los estados de las celdas de una fila, se utiliza System.out.println(); para imprimir un salto de línea, lo que permite comenzar a imprimir la siguiente fila en una nueva línea.



Se inicia un bucle que se ejecuta por la cantidad de turnos especificados. En cada iteración, se imprime en la consola el turno actual.

Se crean dos matrices nuevas de celdas, cellsInform y cellsReceive, para manejar la comunicación entre las celdas durante la simulación.

Se recorre cada celda del tablero con dos bucles anidados. Para cada celda se:

inicializa un nuevo Buffer con una capacidad basada en la columna actual.

Asocia este Buffer con la celda actual en el HashMap buffers.

Crea dos objetos Cell: uno para informar y otro para recibir el estado de la celda. Estos se almacenan en las matrices correspondientes.



Inicio de los Hilos de Celdas:

Utiliza dos bucles anidados para iniciar (start()) los hilos de todas las celdas del tablero. Hay dos tipos de hilos para cada celda: uno que maneja el envío de información (cellsInform) y otro que maneja la recepción (cellsReceive).

Espera a que los Hilos Terminen:

Después de iniciar los hilos, los bucles anidados se usan de nuevo para esperar (join()) a que todos los hilos de celdas terminen su ejecución. Esto asegura que todas las celdas hayan completado su turno antes de proceder.

Impresión del Estado Actual del Tablero:

Finalmente, se utiliza otro conjunto de bucles anidados para imprimir el estado actualizado del tablero después de que todas las celdas hayan terminado su turno. Las celdas vivas se muestran con un asterisco "\*", y las muertas con un guion "-". Se imprime un espacio entre cada celda para una mejor visualización.

**Buffer.java**



La clase del buffer tiene un valor entero en el que se tiene la capacidad y también, un ArrayList de valores booleanos en el que se almacenan los distintos estados de las células que informan.

Texto

Descripción generada automáticamente

Mientras el tamaño del buffer sea igual a la capacidad de este, se espera hasta que se haya liberado espacio. Cuando hay espacio para almacenar valores, se agrega al ArrayList de valores booleanos el nuevo estado de una célula.

Texto

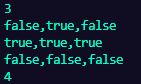
Descripción generada automáticamente

Cuando una célula esta por cambiar de estado está revisa su buffer con el método remove(). En este método, si el buffer está vacío los Threads realizan la operación de yield(), a la espera de que el buffer tenga recursos. Estos Theads pasan de estar en “ejecutando” a “listo”, cuando hay recursos se remueven del buffer y se notifica a los demás Threads.

Validación Realizada

Prueba #1: Ejemplo de la guía

Entrada:



Salida:

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja

Prueba #2:

Entrada:

Texto

Descripción generada automáticamente

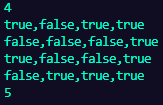
Salida:

Imagen de la pantalla de una computadora

Descripción generada automáticamente con confianza baja

Prueba #3:

Entrada:



Salida:

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Prueba #4:

Entrada:

Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza baja

Salida:

Imagen de la pantalla de un celular

Descripción generada automáticamente con confianza baja

Prueba #5:

Entrada:

Texto

Descripción generada automáticamente

Salida:

Imagen de la pantalla de una computadora

Descripción generada automáticamente con confianza baja

Explicación sincronización





En la clase Buffer tenemos los dos métodos principales con los que el programa funciona de manera adecuada. En el primero de ellos, se emplea el “synchronized” para que solo un thread pueda acceder al método de add(). Por otro lado, para el método remove() no se tiene está notación al comienzo de la declaración. Lo anterior se debe a que al tener ambos métodos con “synchronized” se generan errores al tener un funcionamiento basado en espera pasiva y espera semi-activa. Sin embargo, remove() si tiene un fragmento de código con “synchronized”:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Esto se realiza para que al ingresar al método los distintos Threads puedan realizar la operación de yield() que se encuentra antes de lo mostrado, para luego que solo únicamente uno de esos hilos pueda modificar el buffer quitando recursos de este (buff.remove).

La espera pasiva se tiene en la notificación del estado de una célula. Esto se realiza teniendo la función wait() después de verificar si el buffer está lleno (a su máxima capacidad). La espera semi-activa se realiza en el método remove(), esto se lleva a cabo con la función yield() que se llama sobre el thread. En ambos métodos (remove y add), se notifica a los demás Threads de la finalización de sus respectivos procesos, esto usando notifyAll().

Por consiguiente, se tiene que la comunicación entre Threads, se basa en espera pasiva y espera semi-activa. Esto se realiza cuando se notifica el estado de una célula y cuando se quitan los recursos del buffer (se reinicia para la siguiente iteración del juego) respectivamente. Por otro lado, se debe tener en cuenta el uso de barreras para garantizar un correcto funcionamiento del programa. En nuestro programa se usa join() para garantizar que los N-1 Threads notifiquen su estado para que posteriormente con el último se pueda continuar con el proceso.

Funcionamiento global del sistema

La clase GameLife contiene el método principal main, que inicializa el tablero de juego (board) a partir de la entrada del usuario, estableciendo las dimensiones (N) y el estado inicial de cada celda (viva o muerta). Además, se lee el número de turnos (generaciones) a simular.

Configuración de Celdas y Buzones:

Para cada celda del tablero, se crea un buzón (Buffer) con una capacidad específica. Además, se crean dos objetos Cell por celda: uno para informar a las celdas vecinas de su estado (cellsInform) y otro para recibir información de las vecinas y calcular el estado siguiente (cellsReceive).

Proceso de Simulación:

En cada turno, se inician los hilos de todas las celdas cellsInform para enviar su estado actual a las celdas vecinas y los hilos cellsReceive para recibir estados y calcular el nuevo estado de la celda. Los hilos trabajan de manera concurrente, pero la sincronización se asegura mediante el uso de métodos join(), que esperan a que todos los hilos de un turno completen su trabajo antes de avanzar.

Actualización y Visualización del Tablero:

Después de que todas las celdas han actualizado su estado, el tablero se imprime en la consola, mostrando el resultado de la generación actual. Las celdas vivas se muestran con asteriscos y las muertas con guiones.

Comunicación entre Celdas:

La clase Buffer es crítica para la comunicación entre celdas. Funciona como un buzón de mensajes con espera pasiva y semiactiva, permitiendo a los hilos esperar de manera segura a que haya espacio disponible para enviar información o a que haya información disponible para recibir, asegurando la correcta secuencia de actualizaciones.

Concurrencia y Sincronización:

La concurrencia se manejó con el fin garantizar que todas las celdas se actualicen simultáneamente por generación. Los métodos synchronized en Buffer y el control de hilos con start() y join() son fundamentales para mantener la consistencia del tablero y evitar condiciones de carrera.