LABORATORIO 2

SANTIAGO OSPINA
ISABELLA MANRIQUE

ARQUITECTURA DE SOFTWARE

DIEGO TRIVIÑO

ESCUELA COLOMBIANA DE INGENIERIA JULIO GARAVITO
INGENIERIA DE SISTEMAS
2023-1
BOGOTA D.C

INTRODUCCION

Mediante este laboratorio, llevaremos a la implementación la teoría de los hilos en Java analizando diferentes proyectos que los usan y cómo podemos llegar a mejorar su uso en el código.

PARTE 1

1. Descarga del código PrimeFinder que permite encontrar los números primos entre 0 y M concurrentemente (usando hilos)

2. Para poder hacer que los hilos se detenieran cada x milisegundos y mostraran cuantos números primos habían encontrado lo primero que hicimos fue en el método “run” de la clase PrimeFinderThread implementamos una espera con el método .wait() de la clase Thread usando también synchronized.

```
@Override
public void run(){
    for (int i= a;i < b;i++){
        if (isPrime(i)){
            primes.add(i);
            //System.out.println(i);
        }
                        synchronized (this){
                                while(waiting){
                                        try {
                                                this.wait();
                                        } catch (InterruptedException e) {
                                                throw new RuntimeException(e);
                                        }
                                }
                        }
    }
}
```

Luego, en la clase de “Control” implementamos varios métodos que nos ayudarían a:

a. Contar el tiempo en el que queremos que los hilos paren para mostrar sus resultados:

```
public void timeout(){
    boolean a = Boolean.FALSE;
    long start  = System.currentTimeMillis();
    long fin;
    while (!a){
        fin = System.currentTimeMillis() - start;
        if (fin >= TMILISECONDS){
            a = Boolean.TRUE;
        }
    }
}
```

b. Parar los hilos, con ayuda de una variable booleana que cambiaría para cada uno:

```
public void stopThreads(){
    for (int i = 0; i < NTHREADS; i++) {
        pft[i].stopThread(true);
    }
}
```

c. Imprimir los resultados:

```
public void showPrimes(){
    System.out.println(primes);
    System.out.println();
}
```

d. Escanear el “ENTER” del usuario para continuar con la ejecución:

```
public String scannEnter(){
    Scanner sc = new Scanner(System.in);
    return sc.nextLine();
}
```

e. Reanudar la ejecución de los hilos:

```
public void restartThreads(){
    for (int i = 0; i < NTHREADS; i++) {
        pft[i].restartThread();
    }
}
```

Todos estos métodos implementados en el método “run”:

```
@Override
public void run() {
    startThreads();

    while(pft[pft.length-1].isAlive()) {
        timeout();
        stopThreads();
        showPrimes();

        String entrada = scannEnter();
        if (entrada.equals("")) {
            restartThreads();
        }
    }
}
```

PARTE 2

Explicaciones de los puntos 1 y 2 están en el documento RESPUESTAS.txt en el repositorio de GitHub.

3. Para las condiciones de carrera se realizaron los siguientes syncronized:
   a. En el método de checkIfTurboBoost:

```
private void checkIfTurboBoost(Cell newCell) {
    if (Board.gameboard[newCell.getX()][newCell.getY()].isTurbo_boost()) {
        synchronized (newCell){
            for (int i = 0; i != Board.NR_TURBO_BOOSTS; i++) {
                if (Board.turbo_boosts[i] == newCell) {
                    Board.turbo_boosts[i].setTurbo_boost(false);
                    Board.turbo_boosts[i] = new Cell(-5, -5);
                    hasTurbo = true;
                }
            }
            System.out.println("[" + idt + "] " + "GETTING TURBO BOOST "
                    + newCell.toString());
        }

    }
}
```

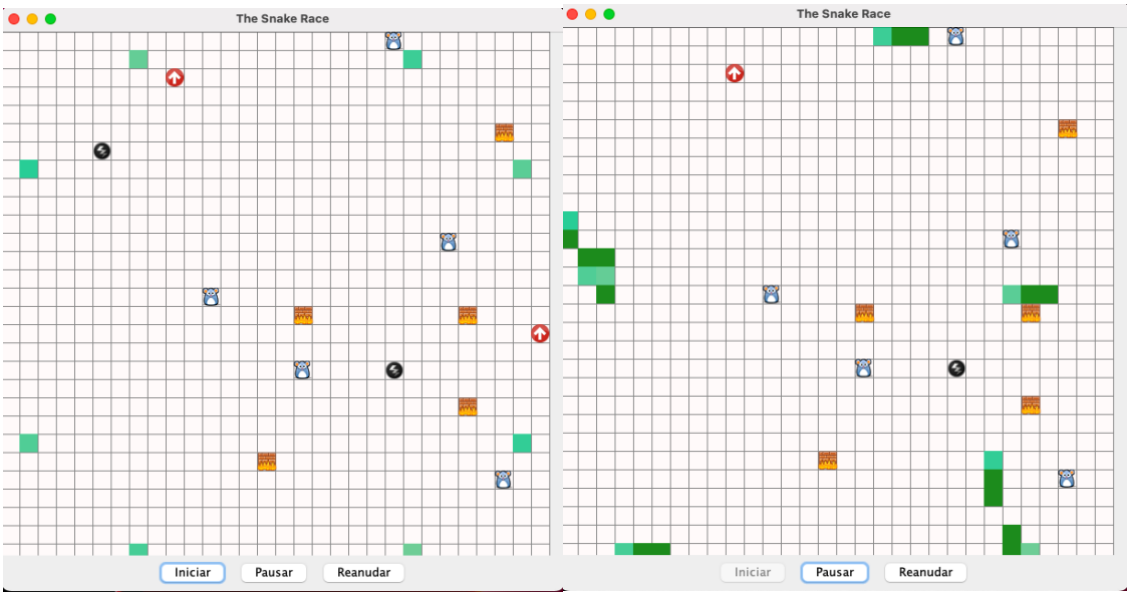
   b. En el método checkIfJumpPad:

```
private void checkIfJumpPad(Cell newCell) {

    if (Board.gameboard[newCell.getX()][newCell.getY()].isJump_pad()) {
        synchronized (newCell){
            for (int i = 0; i != Board.NR_JUMP_PADS; i++) {
                if (Board.jump_pads[i] == newCell) {
                    Board.jump_pads[i].setJump_pad(false);
                    Board.jump_pads[i] = new Cell(-5, -5);
                    this.jumps++;
                }

            }
            System.out.println("[" + idt + "] " + "GETTING JUMP PAD "
                    + newCell.toString());
        }
    }
}
```

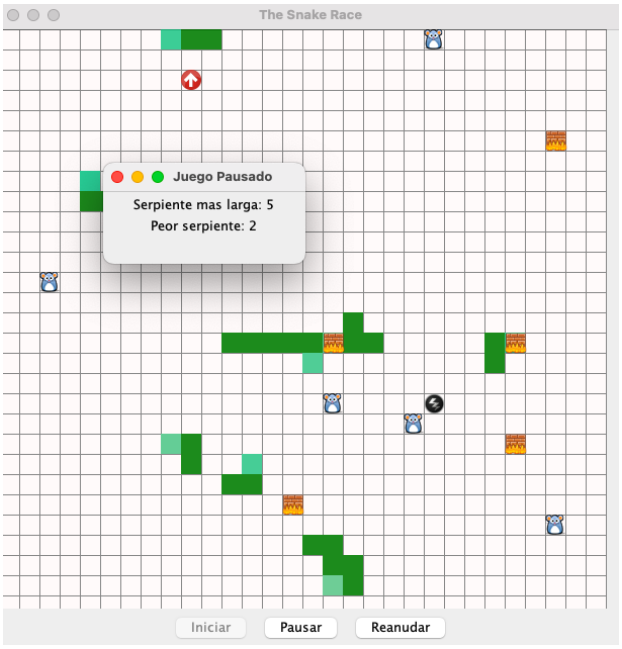
c. En el método checkIfFood:

```
private void checkIfFood(Cell newCell) {
    Random random = new Random();

    if (Board.gameboard[newCell.getX()][newCell.getY()].isFood()) {
        // eat food
        synchronized (newCell) {
            growing += 3;
            int x = random.nextInt(GridSize.GRID_HEIGHT);
            int y = random.nextInt(GridSize.GRID_WIDTH);

            System.out.println("[" + idt + "] " + "EATING "
                    + newCell.toString());

            for (int i = 0; i != Board.NR_FOOD; i++) {
```

4. Implementación de funcionalidades:
   a. Antes y después de oprimir el botón de iniciar



b. Pausar



c. Después de oprimir el botón de reiniciar