



Laboratorio 2

Programación concurrente, condiciones de carrera, esquemas de sincronización,
colecciones sincronizadas y concurrentes.

url repositorio:

https://github.com/ARSW2023-2/ARSW_Lab02.git

Integrantes:

Luisa Fernanda Bermúdez Girón

Karol Daniela Ladino Ladino

Squad:

Inside Out

Profesor:

Javier Iván Toquica Barrera

Curso:

ARSW – 1

Fecha De Entrega:

01-09-2023

I. Introducción

En este laboratorio, exploraremos conceptos fundamentales de programación concurrente, condiciones de carrera y esquemas de sincronización. Estudiaremos cómo manejar múltiples hilos de ejecución de manera eficiente y segura en entornos paralelos.

A través de ejercicios prácticos, nos adentraremos en el uso de herramientas cruciales como wait y notify, perfeccionaremos nuestras habilidades para identificar y resolver condiciones de carrera, y aplicaremos estos conocimientos al enriquecimiento y optimización de una aplicación de juego interactiva.

II. Desarrollo del laboratorio

Parte I

Control de hilos con wait/notify.

1. Descargue el proyecto *PrimeFinder*. Este es un programa que calcula números primos entre 0 y M (Control.MAXVALUE), concurrentemente, distribuyendo la búsqueda de los mismos entre n (Control.NTHREADS) hilos independientes.
2. Se necesita modificar la aplicación de manera que cada t milisegundos de ejecución de los threads, se detengan todos los hilos y se muestre el número de primos encontrados hasta el momento. Luego, se debe esperar a que el usuario presione ENTER para reanudar la ejecución de los mismos. Utilice los mecanismos de sincronización provistos por el lenguaje (wait y notify, notifyAll).

Tenga en cuenta:

- La construcción synchronized se utiliza para obtener acceso exclusivo a un objeto.
- La instrucción A.wait() ejecutada en un hilo B pone en modo suspendido al hilo B (independientemente de qué objeto 'A' sea usado como 'lock'). Para reanudarlo, otro hilo activo puede reanudar a B haciendo 'notify()' al objeto usado como 'lock' (es decir, A).
- La instrucción notify(), despierta el primer hilo que hizo wait() sobre el objeto.
- La instrucción notifyAll(), despierta todos los hilos que están esperando por el objeto (hicieron wait() sobre el objeto).

Nota: El último commit de esta parte fue: Pantallazos parte I. El tag es v1.0

- Creamos la clase Semaforo la cual nos permitirá conocer en que estado se encuentra el hilo

```

package main.java.edu.eci.arsw.primefinder;

public class Semaforo {
    private boolean bandera;
    public synchronized boolean getBandera(){
        return bandera;
    }
    public synchronized void setBandera(boolean bandera){
        this.bandera = bandera;
    }
}

```

- Modificación de la clase Control

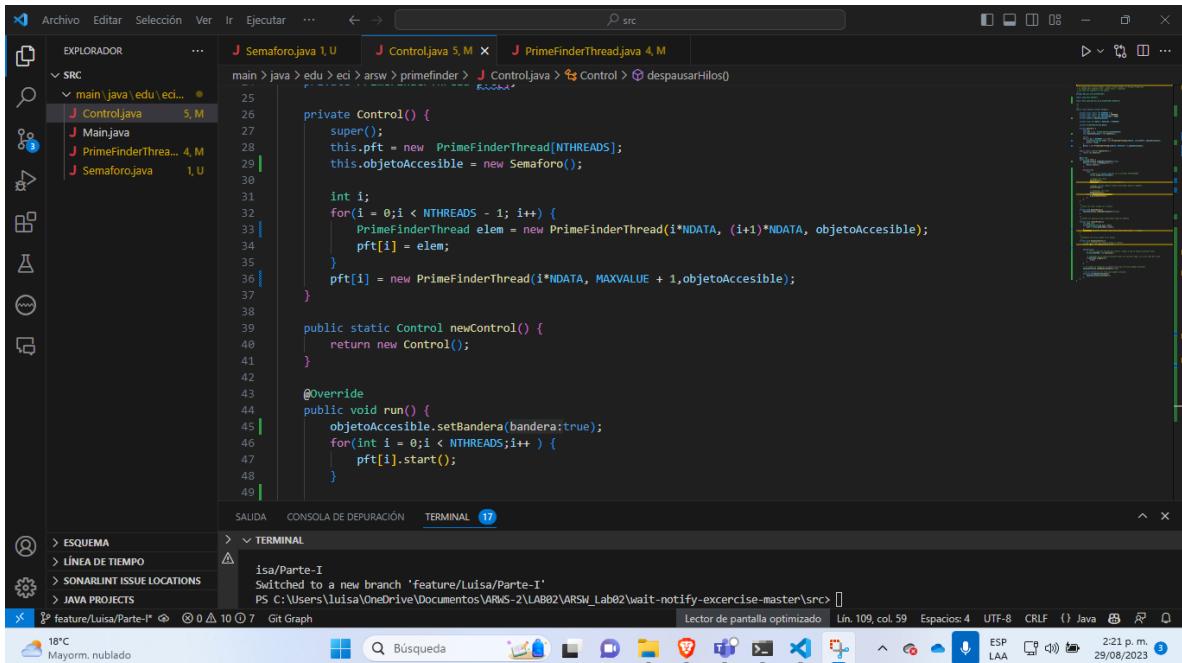
```

package edu.eci.arsw.primefinder;

import java.util.Scanner;
import main.java.edu.eci.arsw.primefinder.Semaforo;

public class Control extends Thread {
    private final static int NTHREADS = 3;
    private final static int MAXVALUE = 30000000;
    private final static int TMILISECONDS = 5000;
    private Semaforo objetoAccesible;
    private final int NDATA = MAXVALUE / NTHREADS;
    private PrimeFinderThread pft[];
}

```



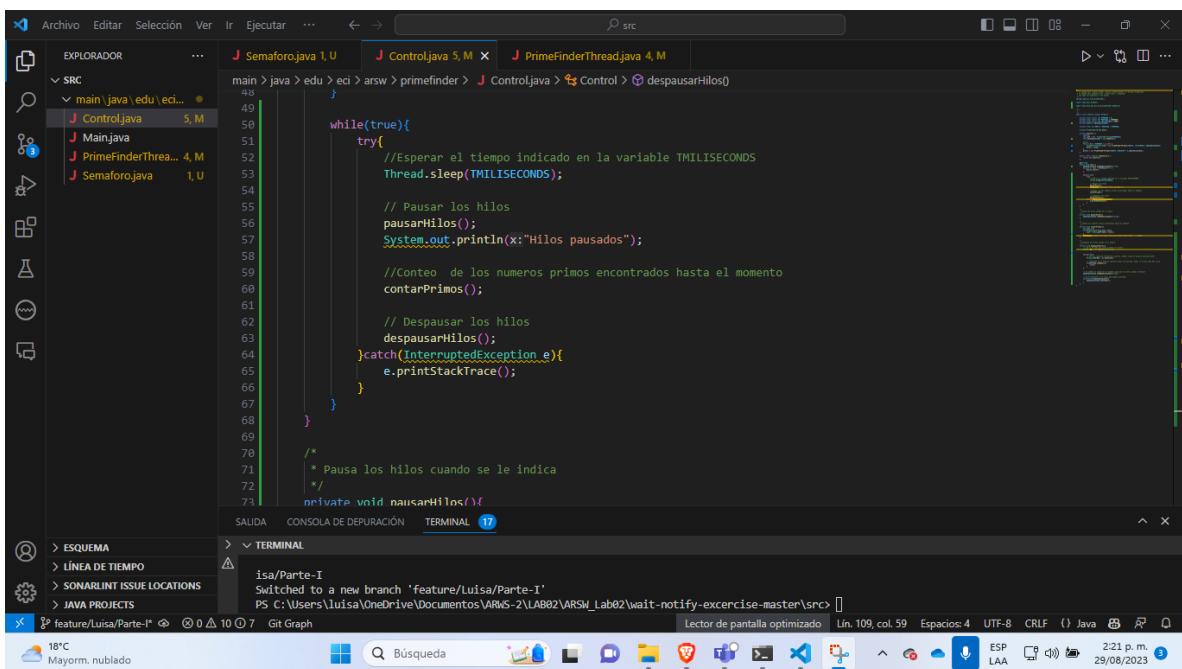
```
main > java > edu > eci > arsw > primefinder > J Controljava > Control > Control.java
```

```
private Control() {
    super();
    this.pft = new PrimeFinderThread[NTHREADS];
    this.objetoAccesible = new Semaforo();

    int i;
    for(i = 0;i < NTHREADS - 1; i++) {
        PrimeFinderThread elem = new PrimeFinderThread(i*NDATA, (i+1)*NDATA, objetoAccesible);
        pft[i] = elem;
    }
    pft[i] = new PrimeFinderThread(i*NDATA, MAXVALUE + 1,objetoAccesible);
}

public static Control newControl() {
    return new Control();
}

@Override
public void run() {
    objetoAccesible.setBandera(bandera:true);
    for(int i = 0;i < NTHREADS;i++ ) {
        pft[i].start();
    }
}
```



```
while(true){
    try{
        //Esperar el tiempo indicado en la variable TMILLISECONDS
        Thread.sleep(TMILLISECONDS);

        // Pausar los hilos
        pausartodos();
        System.out.println(x:"Hilos pausados");

        //Conteo de los numeros primos encontrados hasta el momento
        contarPrimos();

        // Despausar los hilos
        despausartodos();
    }catch(InterruptedException e){
        e.printStackTrace();
    }
}

/*
 * Pausa los hilos cuando se le indica
 */
private void pausartodos(){
```

Screenshot of the Eclipse IDE interface. The code editor shows three Java files: `Semaforo.java`, `Control.java`, and `PrimeFinderThread.java`. The `Control.java` file is currently selected. The terminal window shows the user has switched to a new branch named 'feature/Luisa/Parte-I'. The status bar at the bottom right indicates the date is 29/08/2023.

```
main > java > edu > eci > arsw > primefinder > J Control.java > Control > despausarHilos()
```

```
09
78     /*
79      * Pausa los hilos cuando se le indica
80      */
81
82      private void pausarHilos(){
83          objetoAccesible.setBandera(false);
84      }
85
86      /*
87      * cuenta los numeros primos encontrados hasta el momento
88      */
89      private void contarPrimos(){
90          int cont = 0;
91          for(PrimeFinderThread hilo: pft){
92              cont += hilo.getPrimes().size();
93          }
94          System.out.println("Cantidad numero de primos encontrados: " + cont);
95      }
96
97      /*
98      * Despresa los hilos cuando se le indica
99      */
100     private void despausarHilos(){
101         // Lee la entrada del usuario desde la consola
102         Scanner sc = new Scanner(System.in);
103
104         while(true){
105             // Lee una linea de entrada del usuario, espera a que el usuario presione enter
106             String enterKey = sc.nextLine();
107
108             // comprueba si el usuario presiono enter sin escribir nada, si es asi sale del ciclo
109             if(enterKey.isEmpty()){
110                 break;
111             }
112
113             // Se cambia el estado de la bandera para que los hilos puedan continuar
114             objetoAccesible.setBandera(bandera:true);
115
116             // Se le notifica a los hilos que pueden continuar
117             synchronized(objetoAccesible){
118                 objetoAccesible.notifyAll();
119             }
120         }
121     }
122 }
```

Screenshot of the Eclipse IDE interface, identical to the one above but with a different section of the code visible in the editor. The code editor now shows the continuation of the `despausarHilos()` method, including the loop that checks for an enter key and the synchronization block that notifies threads.

```
main > java > edu > eci > arsw > primefinder > J Control.java > Control > despausarHilos()
```

```
88
89     /*
90      * Despresa los hilos cuando se le indica
91      */
92
93     private void despausarHilos(){
94         // Lee la entrada del usuario desde la consola
95         Scanner sc = new Scanner(System.in);
96
97
98         while(true){
99             // Lee una linea de entrada del usuario, espera a que el usuario presione enter
100             String enterKey = sc.nextLine();
101
102             // comprueba si el usuario presiono enter sin escribir nada, si es asi sale del ciclo
103             if(enterKey.isEmpty()){
104                 break;
105             }
106
107             // Se cambia el estado de la bandera para que los hilos puedan continuar
108             objetoAccesible.setBandera(bandera:true);
109
110             // Se le notifica a los hilos que pueden continuar
111             synchronized(objetoAccesible){
112                 objetoAccesible.notifyAll();
113             }
114         }
115     }
116 }
```

```
main > java > edu > eci > arsw > primefinder > J Control.java 5, M > J PrimeFinderThread.java 4, M
String enterKey = sc.nextLine();
98
99
100 // comprueba si el usuario presiono enter sin escribir nada, si es asi sale del ciclo
101 if(enterKey.isEmpty()){
102     break;
103 }
104
105
106 // Se cambia el estado de la bandera para que los hilos puedan continuar
107 objetoAcessible.setBandera(bandera:true);
108
109 // Se le notifica a los hilos que pueden continuar
110 synchronized(objetoAcessible){
111     objetoAcessible.notifyAll();
112 }
113
114 }
115
```

SALIDA CONSOLA DE DEPURACIÓN TERMINAL 17

ESQUEMA LÍNEA DE TIEMPO SONARLINT ISSUE LOCATIONS JAVA PROJECTS feature/Luisa/Parte-I* 0 10 7 Git Graph 18°C Mayorm. nublado

Lector de pantalla optimizado Lín. 109, col. 59 Espacios: 4 UTF-8 CRLF Java ESP LAA 2:22 p. m. 29/08/2023

➤ Modificación de la clase PrimeFinderThread

```
J Semaforo.java 1, U J Control.java 5, M J PrimeFinderThread.java 4, M
main > java > edu > eci > arsw > primefinder > J PrimeFinderThread.java 4, M > PrimeFinderThread > J PrimeFinderThread(int, int, Semaforo)
1 package edu.eci.arsw.primefinder;
2
3 import java.util.LinkedList;
4 import java.util.List;
5
6 import main.java.edu.eci.arsw.primefinder.Semaforo;
7
8 public class PrimeFinderThread extends Thread{
9
10     int a,b;
11
12     private List<Integer> primes;
13
14     private Semaforo candado;
15
16
17     public PrimeFinderThread(int a, int b, Semaforo candado) {
18         super();
19         this.primes = new LinkedList<>();
20         this.a = a;
21         this.b = b;
22         this.candado = candado;
23     }
24
25     @Override
26     public void run() {
27         for (int i = a; i <= b; i++) {
28             if (esPrimo(i)) {
29                 primes.add(i);
30             }
31         }
32         Semaforo semaforo = new Semaforo();
33         semaforo.signal();
34     }
35
36     private boolean esPrimo(int numero) {
37         if (numero < 2) {
38             return false;
39         }
40         for (int i = 2; i < numero; i++) {
41             if (numero % i == 0) {
42                 return false;
43             }
44         }
45         return true;
46     }
47 }
```

SALIDA CONSOLA DE DEPURACIÓN TERMINAL 17

ESQUEMA LÍNEA DE TIEMPO SONARLINT ISSUE LOCATIONS JAVA PROJECTS feature/Luisa/Parte-I* 0 10 7 Git Graph 18°C Mayorm. nublado

Lector de pantalla optimizado Lín. 17, Col. 12 (17 seleccionados) Espacios: 4 UTF-8 CRLF Java ESP LAA 2:23 p. m. 29/08/2023

Screenshot of an IDE showing the PrimeFinderThread.java file. The code implements a thread to find prime numbers using a semaphore. It includes comments explaining the use of the semaphore's flag and synchronized blocks.

```
main > java > edu > eci > arsw > primefinder > PrimeFinderThread.java > PrimeFinderThread > PrimeFinderThread(int, int, Semaforo)
22     this.candado = candado;
23 }
24
25 @Override
26 public void run(){
27     for (int i= a;i < b;i++){
28
29         // Si la bandera del candado es falsa el hilo debera esperar
30         if(!candado.getBandera()){
31             try{
32                 synchronized(candado){
33                     candado.wait();
34                 }
35             }catch(InterruptedException e){
36                 e.printStackTrace();
37             }
38         }
39         if (isPrime(i)){
40             primes.add(i);
41             System.out.println(i);
42         }
43     }
44
45     boolean isPrime(int n){
46         boolean ans;
47         if (n > 2){
48
49             if (n % 2 != 0){
50                 for(int i = 3;ans && i*i <= n; i+=2 ) {
51                     ans = n % i != 0;
52                 }
53             } else {
54                 ans = n == 2;
55             }
56         }
57         return ans;
58     }
59
60     public List<Integer> getPrimes() {
61         return primes;
62     }
63 }
```

Screenshot of an IDE showing the PrimeFinderThread.java file with a different implementation of the isPrime method. This version uses a more efficient trial division algorithm.

```
main > java > edu > eci > arsw > primefinder > PrimeFinderThread.java > PrimeFinderThread > PrimeFinderThread(int, int, Semaforo)
43 }
44
45
46     boolean isPrime(int n) {
47         boolean ans;
48         if (n > 2) {
49             ans = n%2 != 0;
50             for(int i = 3;ans && i*i <= n; i+=2 ) {
51                 ans = n % i != 0;
52             }
53         } else {
54             ans = n == 2;
55         }
56
57         return ans;
58     }
59
60     public List<Integer> getPrimes() {
61         return primes;
62     }
63 }
```

➤ Resultados ejecución:

The screenshot shows the Visual Studio Code interface with the following details:

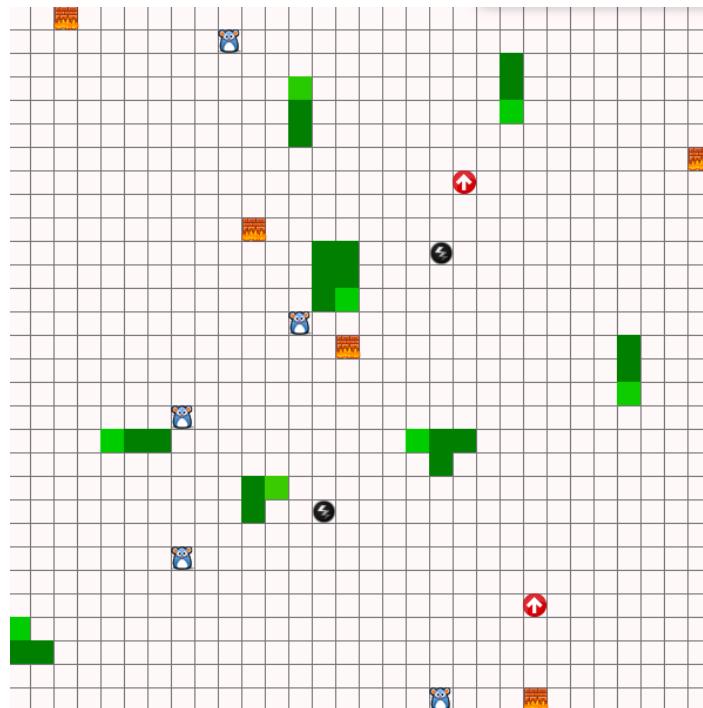
- Explorer View:** Shows the project structure under the `SRC` folder:
 - `main\java\edu\eci\arsw\primefinder`
 - `J Control.java`, `J Main.java`, `J PrimeFinderThrea...`, `J Semaforojava`
- Code Editor:** The `Main.java` file is open, displaying Java code for prime number finding.
- Terminal:** The terminal window shows the output of the program, listing many prime numbers starting from 372973 up to 373199, with a total count of 40174 found.
- Status Bar:** Shows the file `feature/Luisa/Parte-1*`, line 2, column 1, and other system information like temperature (18°C) and date (29/06/2023).

The screenshot shows the Eclipse IDE interface. On the left, the 'Explorador' (File Explorer) shows a Java project structure with files like Main.java, Control.java, and Semáforo.java. The 'Main.java' file is currently selected and open in the central editor area. The editor displays a list of prime numbers from 1 to 85761. Below the editor, the 'TERMINAL' view shows the command-line output of the program. The status bar at the bottom right indicates there are 18 issues found in the project.

Parte II

SnakeRace es una versión autónoma, multi-serpiente del famoso juego 'snake', basado en el proyecto de João Andrade -este ejercicio es un 'fork' del mismo-. En este juego:

- N serpientes funcionan de manera autónoma.
- No existe el concepto de colisión entre las mismas. La única forma de que mueran es estrellándose contra un muro.
- Hay ratones distribuidos a lo largo del juego. Como en el juego clásico, cada vez que una serpiente se come a un ratón, ésta crece.
- Existen unos puntos (flechas rojas) que teletransportan a las serpientes.
- Los rayos hacen que la serpiente aumente su velocidad.



Ejercicio

1. Analice el código para entender cómo hace uso de hilos para crear un comportamiento autónomo de las N serpientes.

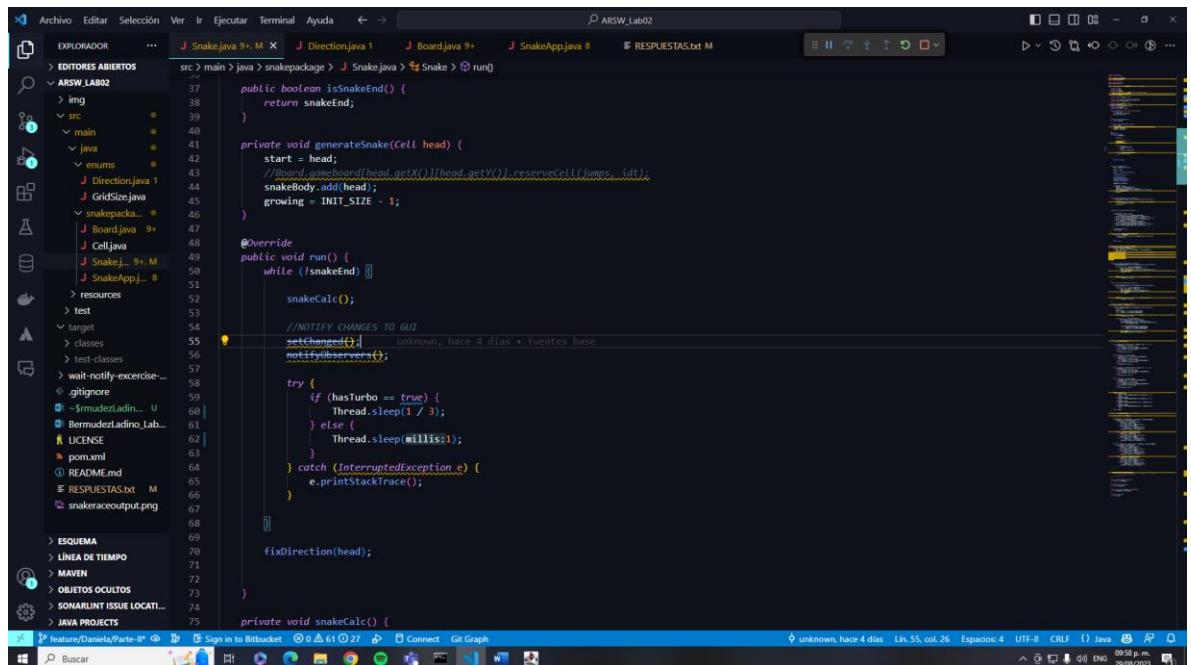
La clase Snake implementa la interfaz Runnable, lo que significa que cada instancia de serpiente puede ejecutarse en su propio hilo el cual ejecuta la lógica de movimiento y comportamiento de cada una de las serpientes.

SnakeApp crea instancias de serpientes y hilos para cada serpiente, permitiendo que todas las serpientes se muevan y actúen autónomamente en paralelo.

2. De acuerdo con lo anterior, y con la lógica del juego, identifique y escriba claramente (archivo RESPUESTAS.txt):

- Posibles condiciones de carrera.
 1. Revisar si una celda está libre o no.
 2. Si hay comida disponible en una casilla.
 3. Si una flecha roja está disponible o no.
 4. Si un rayo está disponible o no.
- Uso inadecuado de colecciones, considerando su manejo concurrente (para esto, aumente la velocidad del juego y ejecútelo varias veces hasta que se genere un error).

Dentro de la clase Snake, ajustamos los valores de los tiempos de espera en el método run(). Para esto reducimos los valores cambiando de 500 milisegundos a 1 milisegundo dentro del método Thread.sleep() para que las serpientes se movieran más rápido.



The screenshot shows a Java development environment with several files open in editors. The main file is `Snake.java`, which contains the following code:

```
public boolean isSnakeEnd() {
    return snakeEnd;
}

private void generateSnake(Cell head) {
    start = head;
    //Board.gameboard(head.getx())(head.gety()).reserveCell(jumps_, id);
    snakeBody.add(head);
    growing = INIT_SIZE - 1;
}

@Override
public void run() {
    while (!snakeEnd) {
        snakeCalc();
        //NOTIFY CHANGES TO GUI
        setChanged();
        notifyObservers();
    }
}

try {
    if (hasTurbo == true) {
        Thread.sleep(1 / 3);
    } else {
        Thread.sleep(millis:1);
    }
} catch (InterruptedException e) {
    e.printStackTrace();
}

fixDirection(head);
}

private void snakeCalc() {
```

Como podemos ver en la siguiente imagen luego de ejecutar el juego el error que sale es ConcurrentModificationException, este error se puede dar en el momento en que se está dibujando la serpiente ya que la referencia que pide el tablero para dibujarla se está iterando mientras la serpiente se modifica (agregando o eliminando elementos).

The screenshot shows a Java IDE interface with multiple tabs open. The active tab is 'src\main\java\snakepackage\SnakeApp.java'. A stack trace is displayed in the terminal window:

```
Exception in thread "Main EventQueue-0" java.util.ConcurrentModificationException
at java.base/java.util.LinkedList$ListIterator.checkForConModification(LinkedList.java:970)
at java.base/java.util.LinkedList$ListIterator.next(LinkedList.java:892)
at snakepackage.Board.drawSnake(Board.java:186)
at snakepackage.Board.paintComponent(Board.java:111)
at javax.swing.JComponent.paint(JComponent.java:1119)
at javax.swing.JComponent.paintChildren(JComponent.java:952)
at javax.swing.JComponent.paint(JComponent.java:1128)
at javax.swing.JComponent.paintAll(JComponent.java:531)
at javax.swing/Desktop$WindowManager$PaintScreen$DoubleBufferedRepaintManagerImpl$RepaintManager.paint(RepaintManager.java:1657)
at javax.swing/Desktop$WindowManager$PaintScreen$DoubleBufferedRepaintManagerImpl$RepaintManager.paint(RepaintManager.java:1632)
at javax.swing/Desktop$WindowManager$PaintManager$PaintManager.paint(RepaintManager.java:1570)
at javax.swing/Desktop$WindowManager$PaintManager.paint(RepaintManager.java:1337)
at javax.swing/Desktop$WindowManager$PaintManager.paintImmediately(JComponent.java:5259)
at javax.swing/Desktop$WindowManager$PaintManager.paintImmediately(JComponent.java:5069)
at javax.swing/Desktop$WindowManager$PaintManager$4.run(RepaintManager.java:879)
at javax.swing/Desktop$WindowManager$PaintManager$4.run(RepaintManager.java:835)
at java.awt.event.InvocationEvent.dispatch(InvocationEvent.java:399)
at java.awt.EventQueue.dispatchEventImpl(EventQueue.java:771)
at java.awt.EventQueue.access$400(EventQueue.java:102)
at java.awt.EventQueue$4.run(EventQueue.java:716)
at java.awt.EventQueue$4.run(EventQueue.java:715)
at java.base/java.security.AccessController.doPrivileged(AccessController.java:399)
at java.base/java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:86)
at java.desktop/java.awt.EventQueue.dispatchEvent(EventQueue.java:741)
```

- Uso innecesario de esperas activas.

La espera innecesaria está en la clase SnakeApp en el while del metodo init, ya que este bucle verifica constantemente si todas las serpientes han llegado al final del juego. Cuando todas las serpientes hayan terminado, el bucle se rompe.

3. Identifique las regiones críticas asociadas a las condiciones de carrera, y haga algo para eliminarlas. Tenga en cuenta que se debe sincronizar estrictamente LO NECESARIO. En su documento de respuestas indique, la solución realizada para cada ítem del punto 2. Igualmente tenga en cuenta que en los siguientes puntos NO se deben agregar más posibles condiciones de carrera.

Para evitar las posibles condiciones de carrera que se encontraban en la clase Snake mencionadas anteriormente:

- Utilizamos el **bloque synchronized** para sincronizar el acceso a la colección snakeBody en todos los lugares donde se modifica o itera sobre esta.
 - Sincronizamos los métodos checkIfTurboBoost, chekIfJumpPad y checkIfFood utilizando la palabra clave synchronized. De esta manera se asegura que solo un hilo pueda ejecutar cualquiera de estos métodos en un momento dado.

➤ Modificación de la clase Snake

```
src > main > java > snakepackage > J_Snake.java > checkJumpPad(Cell)
41     private void generateSnake(Cell head) {
42         start = head;
43         //Board.gameboard[head.getX()][head.getY()].reserveCell(jumps, ldt);
44
45         //Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
46         synchronized(snakeBody){
47             snakeBody.add(head);
48         }
49         growing = INIT_SIZE - 1;
50     }
51
52     @Override
53     public void run() {
54         while (!snakeEnd) {
55
56             snakeCalc();
57
58             //NOTIFY CHANGES TO GUI
59             setChanged();
60             notifyObservers();
61
62             try {
63                 if (hasTurbo == true) {
64                     Thread.sleep(500 / 3);
65                 } else {
66                     Thread.sleep(millis*500);
67                 }
68             } catch (InterruptedException e) {
69                 e.printStackTrace();
70             }
71         }
72
73         fixDirection(head);
74
75     }
76
77     private void snakeCalc() {
78
79         private void checkJumpPad(Cell
```

The screenshot shows an IDE interface with the title bar "ARSW_Lab02". The left sidebar lists files and folders: "EDTORES ABIERTOS", "src", "main", "java", "enums", "Direction.java", "GridSize.java", "snakepackage..", "Board.java", "Cell.java", "Snake.java", "resources", "test", "target", "classes", "test-classes", "wait-notify-exercise..", "gitignore", ".gitignore", ".gitattributes", "LICENSE", "pom.xml", "README.md", "RESPUESTAS.txt", and "snakeraceoutput.png". The right pane displays the "Snake.java" file with the following code:

```
private void snakeCalc() {
    //Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
    synchronized(snakeBody) {
        head = snakeBody.peekFirst();
    }

    newCell = head;
    newCell = changeDirection(newCell);
    randomMovement(newCell);

    checkIfFood(newCell);
    checkIfJumpPad(newCell);
    checkIfBarrier(newCell);
    checkIfFruit(newCell);

    //Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
    synchronized(snakeBody) {
        snakeBody.push(newCell);
    }

    if (growing <= 0) {
        newCell = snakeBody.peekLast();
        snakeBody.remove(snakeBody.peekLast());
        Board.gameboard[newCell.getX()][newCell.getY()].freeCell();
    } else if (growing > 0) {
        growing--;
    }
}

private void checkIfBarrier(Cell newCell) {
    if (Board.gameboard[newCell.getX()][newCell.getY()].isBarrier()) {
        //crash
        System.out.println("[" + idt + "] " + "CRASHED AGAINST BARRIER."
            + newCell.toString());
        snakeEnd=true;
    }
}

private void checkIfJumpPad(Cell newCell) {
    if (Board.gameboard[newCell.getX()][newCell.getY()].isJumpPad()) {
        //jump
        System.out.println("[" + idt + "] " + "JUMPED ON PAD."
            + newCell.toString());
        snakeEnd=true;
    }
}
```

The screenshot shows an IDE interface with the title bar "ARSW_Lab02". The left sidebar lists files and folders: "EDTORES ABIERTOS", "src", "main", "java", "enums", "Direction.java", "GridSize.java", "snakepackage..", "Board.java", "Cell.java", "Snake.java", "resources", "test", "target", "classes", "test-classes", "wait-notify-exercise..", "gitignore", ".gitignore", ".gitattributes", "LICENSE", "pom.xml", "README.md", "RESPUESTAS.txt", and "snakeraceoutput.png". The right pane displays the "Snake.java" file with the following code:

```
private Cell fixDirection(Cell newCell) {
    // revert movement
    if (direction == Direction.LEFT) {
        if (head.getX() + 1 < GridSize.GRID_WIDTH) {
            newCell = Board.gameboard[head.getX() + 1][head.getY()];
        } else if (direction == Direction.RIGHT && head.getX() - 1 > 0) {
            newCell = Board.gameboard[head.getX() - 1][head.getY()];
        } else if (direction == Direction.UP) {
            if (head.getY() + 1 < GridSize.GRID_HEIGHT) {
                newCell = Board.gameboard[head.getX()][head.getY() + 1];
            } else if (direction == Direction.DOWN && head.getY() - 1 > 0) {
                newCell = Board.gameboard[head.getX()][head.getY() - 1];
            }
        }
    }

    randomMovement(newCell);
    return newCell;
}

private boolean checkIfOnBody(Cell newCell) {
    for (Cell c : snakeBody) {
        if (newCell.getX() == c.getX() && newCell.getY() == c.getY()) {
            return true;
        }
    }
    return false;
}

private void randomMovement(Cell newCell) {
    Random random = new Random();
    int temp = random.nextInt(GridSize.GRID_SIZE * 2);
    if (temp % 2 == 0 && direction == Direction.LEFT) {
        direction = Direction.RIGHT;
    } else if (temp % 2 == 1 && direction == Direction.RIGHT) {
        direction = Direction.LEFT;
    } else if (temp % 2 == 0 && direction == Direction.UP) {
        direction = Direction.DOWN;
    } else if (temp % 2 == 1 && direction == Direction.DOWN) {
        direction = Direction.UP;
    }
}
```

```
//Se sincroniza el metodo checkIfTurboBoost para que solo un hilo pueda ejecutarlo a la vez
private synchronized void checkIfTurboBoost(Cell newCell) {
    if (Board.gameboard[newCell.getX()][newCell.getY()].isTurbo_boost()) {
        //get turbo boost
        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());
    }
}

//Se sincroniza el metodo checkIfJumpPad para que solo un hilo pueda ejecutarlo a la vez
private synchronized void checkIfJumpPad(Cell newCell) {
    if (Board.gameboard[newCell.getX()][newCell.getY()].isJump_pad()) {
        //get jump pad
        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());
    }
}
```

```
//Se sincroniza el metodo checkIfFood para que solo un hilo pueda ejecutarlo a la vez
private synchronized void checkIfFood(Cell newCell) {
    Random random = new Random();
    if (Board.gameboard[newCell.getX()][newCell.getY()].isFood()) {
        // eat food
        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++) {
            if (Board.food[i].getX() == newCell.getX()
                && Board.food[i].getY() == newCell.getY()) {
                Board.gameboard[Board.food[i].getX()][Board.food[i].getY()]
                    .setFood(false);
                while (Board.gameboard[x][y].hasElements()) {
                    x = random.nextInt(GridSize.GRID_HEIGHT);
                    y = random.nextInt(GridSize.GRID_WIDTH);
                }
                Board.food[i] = new Cell(x, y);
                Board.gameboard[x][y].setFood(true);
            }
        }
    }
}
```

```
src > main > java > snakepackage > J_Snake.java > Snake > checkJumpPad(Cell)

    }

    private Cell changeDirection(Cell newCell) {
        // Avoid out of bounds
        while (direction == Direction.UP && (newCell.getY() - 1) < 0) {
            if ((head.getX() - 1) < 0) {
                this.direction = Direction.RIGHT;
            } else if ((head.getX() + 1) == gridSize.GRID_WIDTH) {
                this.direction = Direction.LEFT;
            } else {
                randomMovement(newCell);
            }
        }
        while (direction == Direction.DOWN
                && (head.getY() + 1) == gridSize.GRID_HEIGHT) {
            if ((head.getX() - 1) < 0) {
                this.direction = Direction.RIGHT;
            } else if ((head.getX() + 1) == gridSize.GRID_WIDTH) {
                this.direction = Direction.LEFT;
            } else {
                randomMovement(newCell);
            }
        }
        while (direction == Direction.LEFT && (head.getX() - 1) < 0) {
            if ((newCell.getY() - 1) < 0) {
                this.direction = Direction.DOWN;
            } else if ((newCell.getY() + 1) == gridSize.GRID_HEIGHT) {
                this.direction = Direction.UP;
            } else {
                randomMovement(newCell);
            }
        }
        while (direction == Direction.RIGHT
                && (head.getX() + 1) == gridSize.GRID_WIDTH) {
            if ((newCell.getY() - 1) < 0) {
                this.direction = Direction.DOWN;
            } else if ((newCell.getY() + 1) == gridSize.GRID_HEIGHT) {
                this.direction = Direction.UP;
            } else {
                randomMovement(newCell);
            }
        }
    }

    public void searchObjective(Cell objective) {
        Random random = new Random();
        // MOVE DIRECTLY TO OBJECTIVE
        if (direction == Direction.LEFT || direction == Direction.RIGHT) {
            if (direction == Direction.LEFT) {
                if (head.getY() > objective.getY()) {
                    direction = Direction.UP;
                } else if (head.getY() < objective.getY()) {
                    direction = Direction.DOWN;
                } else if (head.getX() == objective.getX()) {
                    direction = random.nextInt(2) + 3;
                }
            }
        }
    }
}
```

```
src > main > java > snakepackage > J_Snake.java > Snake > checkJumpPad(Cell)

    }

    private Cell changeDirection(Cell newCell) {
        // Avoid out of bounds
        while (direction == Direction.UP && (newCell.getY() - 1) < 0) {
            if ((head.getX() - 1) < 0) {
                this.direction = Direction.RIGHT;
            } else if ((head.getX() + 1) == gridSize.GRID_HEIGHT) {
                this.direction = Direction.OP;
            } else {
                randomMovement(newCell);
            }
        }
        switch (direction) {
            case Direction.UP:
                newCell = Board.gameboard[head.getX()][head.getY() - 1];
                break;
            case Direction.DOWN:
                newCell = Board.gameboard[head.getX()][head.getY() + 1];
                break;
            case Direction.LEFT:
                newCell = Board.gameboard[head.getX() - 1][head.getY()];
                break;
            case Direction.RIGHT:
                newCell = Board.gameboard[head.getX() + 1][head.getY()];
                break;
        }
        return newCell;
    }

    public void searchObjective(Cell objective) {
        Random random = new Random();
        // MOVE DIRECTLY TO OBJECTIVE
        if (direction == Direction.LEFT || direction == Direction.RIGHT) {
            if (direction == Direction.LEFT) {
                if (head.getY() > objective.getY()) {
                    direction = Direction.UP;
                } else if (head.getY() < objective.getY()) {
                    direction = Direction.DOWN;
                } else if (head.getX() == objective.getX()) {
                    direction = random.nextInt(2) + 3;
                }
            }
        }
    }
}
```

```
src > main > java > snakepackage > J_Snake.java < objective.getX() {
    && head.getX() < objective.getX() {
        direction = random.nextInt(bound2) + 3;
    }
} else if (direction == Direction.RIGHT) {
    if (head.getX() > objective.getY()) {
        direction = Direction.UP;
    } else if (head.getY() < objective.getX()) {
        direction = Direction.DOWN;
    } else if (head.getY() == objective.getX()) {
        && head.getX() > objective.getX() {
            direction = random.nextInt(bound2) + 3;
        }
    }
} else if (direction == Direction.UP || direction == Direction.DOWN) {
    if (direction == Direction.UP) {
        if (head.getX() > objective.getX()) {
            direction = Direction.LEFT;
        } else if (head.getX() < objective.getX()) {
            direction = Direction.RIGHT;
        } else if (head.getX() == objective.getX() && head.getY() < objective.getY()) {
            direction = random.nextInt(bound2) + 1;
        }
    }
} else if (direction == Direction.DOWN) {
    if (head.getX() > objective.getX()) {
        direction = Direction.LEFT;
    } else if (head.getX() < objective.getX()) {
        direction = Direction.RIGHT;
    } else if (head.getX() == objective.getX() && head.getY() > objective.getY()) {
        direction = random.nextInt(bound2) + 1;
    }
}
}

public void setObjective(Cell c) {
    System.out.println("Setting objective - " + c.getX() + ":" + c.getY());
    + " for Snake" + this.id);
}
```

```
src> main > java > snakepackage > J_Snake.java > ¶ Snake > checkIfJumpPad(Cell)
    if (head.getX() > objective.getX()) {
        direction = Direction.LEFT;
    } else if (head.getX() < objective.getX()) {
        direction = Direction.RIGHT;
    } else if (head.getX() == objective.getX()
              && head.getY() > objective.getY()) {
        direction = random.nextInt(bound) + 1;
    }
}

//public void setObjective(Cell c) {
//    System.out.println("Setting objective - " + c.getX() + ":" + c.getY());
//    this.objective = c;
//}

public LinkedList<Cell> getBody() {
    //Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
    synchronized(snakeBody){
        return new LinkedList<Cell>(this.snakeBody);
    }
}

public boolean isSelected() {
    return isSelected;
}

public void setSelected(boolean isSelected) {
    this.isSelected = isSelected;
}

public int getIdt() {
    return idt;
}
```

Se modifico la clase SnakeApp realizando un bucle for para esperar que todos los hilos de las serpientes finalicen su ejecución. Dentro de este bucle, llamamos el método join en cada hilo para que cuando todas las serpientes terminen se consoliden los resultados y así poder imprimir el estado de los hilos. A demás de esto comentamos el while del método init para evitar el uso innecesario de esperas activas.

➤ Modificación de la clase SnakeApp

```

    public class SnakeApp {
        private static SnakeApp app;
        public static final int MAX_THREADS = 8;
        Snake[] snakes = new Snake[MAX_THREADS];
        private static final Cell[] spawn = {
            new Cell(x:1, (GridSize.GRID_HEIGHT / 2) / 2),
            new Cell(GridSize.GRID_WIDTH - 2,
                3 * (GridSize.GRID_HEIGHT / 2) / 2),
            new Cell(3 * (GridSize.GRID_WIDTH / 2) / 2, y:1),
            new Cell((GridSize.GRID_WIDTH / 2) / 2, GridSize.GRID_HEIGHT - 2),
            new Cell(x:1, 3 * (GridSize.GRID_HEIGHT / 2) / 2),
            new Cell(GridSize.GRID_WIDTH - 2, (GridSize.GRID_HEIGHT / 2) / 2),
            new Cell((GridSize.GRID_WIDTH / 2) / 2, 2, y:1),
            new Cell(3 * (GridSize.GRID_WIDTH / 2) / 2,
                GridSize.GRID_HEIGHT - 2)};
        private JFrame frame;
        private static Board board;
        int nr_selected = 0;
        Thread[] thread = new Thread[MAX_THREADS];
        public SnakeApp() {
            Dimension dimension = Toolkit.getDefaultToolkit().getScreenSize();
            frame = new JFrame("The Snake Race");
            frame.setLayout(new BorderLayout());
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // Frame.setSize(618, 640);
            frame.setSize(GridSize.GRID_WIDTH * Gridsize.WIDTH_BOX + 17,
                GridSize.GRID_HEIGHT * Gridsize.HEIGHT_BOX + 40);
            frame.setLocation(dimension.width / 2 - frame.getWidth() / 2,
                dimension.height / 2 - frame.getHeight() / 2);
            board = new Board();
            frame.add(board, BorderLayout.CENTER);
            JPanel actionsBPabel=new JPanel();
            actionsBPabel.setLayout(new FlowLayout());
            actionsBPabel.add(new JButton(text:"Action"));
            frame.add(actionsBPabel,BorderLayout.SOUTH);
        }
        public static void main(String[] args) {
            app = new SnakeApp();
            app.init();
        }
        private void init() {
            for (int i = 0; i <= MAX_THREADS; i++) {
                snakes[i] = new Snake(i + 1, spawn[i], i + 1);
                snakes[i].addObserver(board);
                thread[i] = new Thread(snakes[i]);
                thread[i].start();
            }
            frame.setVisible(true);
        }
    }

```

```

    public static void main(String[] args) {
        app = new SnakeApp();
        app.init();
    }
    private void init() {
        for (int i = 0; i <= MAX_THREADS; i++) {
            snakes[i] = new Snake(i + 1, spawn[i], i + 1);
            snakes[i].addObserver(board);
            thread[i] = new Thread(snakes[i]);
            thread[i].start();
        }
        frame.setVisible(true);
    }
    public void run() {
        int x = 0;
        for (int i = 0; i <= MAX_THREADS; i++) {
            if (snakes[i].isSnakeEnd() == true) {
                x++;
            }
        }
        if (x == MAX_THREADS) {
            break;
        }
    }
}

```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the ARSW_LAB02 project structure, including src, img, main, enums, resources, test, target, classes, test-classes, wait-notify-exercise..., gignore, LICENSE, pom.xml, README.mnd, RESPUESTAS.txt, and snakeaceoutput.png.
- Code Editor:** Displays the content of SnakeApp.java. The code is a Java application for a snake game, utilizing multithreading. It includes imports for java.util.concurrent, java.util, and java.awt.*. The main logic involves creating threads for snakes and joining them to consolidate results. A snippet of the code is shown below:

```
src > main > java > snakepackage > J_SnakeApp.java > ...
84     int x = 0;
85     for (int i = 0; i < MAX_THREADS; i++) {
86         if (snakes[i].isSnakeEnd() == true) {
87             x++;
88         }
89     }
90     if (x == MAX_THREADS) {
91         break;
92     }
93 }
94 //Espera a que todos los hilos finalicen su ejecucion
95 for(int i=0; i<MAX_THREADS; i++){
96     try{
97         //Espera a que todos los hilos terminen para consolidar los resultados
98         thread[i].join();
99     }catch(InterruptedException e){
100         e.printStackTrace();
101     }
102 }
103
104 System.out.println("Thread (snake) status:");
105 for (int i = 0; i < MAX_THREADS; i++) {
106     System.out.println("[" + i + "] :" + thread[i].getState());
107 }
108
109 public static SnakeApp getApp() {
110     return app;
111 }
112 }
```

The code editor also shows annotations like `Override` and `catch(InterruptedException e)`. The status bar at the bottom indicates "Ln. 121, col. 1 Espacio: 4 UTF-8 CRLF () Java".

Resultados ejecución:

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Same as the first screenshot, showing the ARSW_LAB02 project structure.
- Code Editor:** Displays the content of SnakeApp.java, identical to the first screenshot.
- Terminal:** Shows the output of the application execution. The terminal window title is "TERMINAL" and it displays the following text:

```
[2] : TERMINATED
[4] : TERMINATED
[5] : TERMINATED
[6] : TERMINATED
[7] : TERMINATED
```
- Status Bar:** Shows "You, hace 3 minutos Lin. 46, col. 33 Espacio: 4 UTF-8 CRLF () Java" and the date "30/08/2023".

```

    Arquivo | Editar | Selección | Ver | Ir | Ejecutar | Terminal | Ayuda | < - > | ARSW_Lab02
    EXPLORADOR ... J_SnakeJava 9+ M J_CellJava J_DirectionJava J_BoardJava J_SnakeAppJava 9+ M RESPUESAS.txt
    EDITORES ABIERTOS
    > ARSW_LAB02
    > Img
    > src
    > enums
    > snakepackage...
    > Direction.java
    > GridSize.java
    > Cell.java
    > Board.java
    > Snake...
    > J_Snake...
    > resources
    > test
    > target
    > classes
    > test-classes
    > wait-notify-exercise...
    > githignore
    > .gitignore
    > BermudezLadino...
    > LICENSE
    > pom.xml
    > README.md
    > RESPUESTAS.txt
    > snakeraceoutput.png
    > EQUEMA
    > LÍNEA DE TIEMPO
    > OBJETOS OCULTOS
    > SONARLINT ISSUE LOCAT...
    > JAVA PROJECTS
    > MAVEN
    PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL GITLENS SQL CONSOLE COMENTARIOS
    Windows PowerShell
    Copyright (C) Microsoft Corporation. Todos los derechos reservados.
    Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/powershell
    Advertencia: PowerShell detectó que es posible que estés usando un lector de pantalla y que hayas deshabilitado PSReadLine con fines de compatibilidad. Siquieres volver a habilitarlo, ejecuta "Import-Module PSReadLine".
    PS C:\Users\ala de dios\Documents\ARSW\ARSW_Lab02> & 'C:\Program Files\Java\jdk-17.0.2\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\ala de dios\Documents\ARSW\ARSW_Lab02\target\classes' 'snakepackage.SnakeApp'
    [6] EATING x: 28 y: 6
    [1] CRASHED AGAINST BARRIER x: 3 y: 8
    [3] GETTING TURBO BOOST x: 18 y: 2
    [2] CRASHED AGAINST BARRIER x: 25 y: 19
    [4] CRASHED AGAINST BARRIER x: 26 y: 6
    [8] EATING x: 26 y: 19
    [7] CRASHED AGAINST BARRIER x: 3 y: 8
    [8] EATING x: 16 y: 26
    [8] EATING x: 13 y: 28
    [5] CRASHED AGAINST BARRIER x: 6 y: 16
    [2] CRASHED AGAINST BARRIER x: 25 y: 19
    [6] EATING x: 25 y: 19
    [4] CRASHED AGAINST BARRIER x: 25 y: 19
    [8] CRASHED AGAINST BARRIER x: 25 y: 19
    Thread (snake) status:
    [0] :TERMINATED
    [1] :TERMINATED
    [2] :TERMINATED
    [3] :TERMINATED
    [4] :TERMINATED
    [5] :TERMINATED
    [6] :TERMINATED
    [7] :TERMINATED
    Lin. 121, col. 1 Espacio: 4 UTT-8 CRLF ( ) Java
    Buscar
  
```

Al ejecutar nuevamente la aplicación con la misma velocidad del experimento del punto 2 y luego de haber realizado las respectivas modificaciones mencionadas anteriormente podemos observar que ya no nos sale el error de ConcurrentModificationException.

- Como se puede observar, el juego está incompleto. Haga los ajustes necesarios para que a través de botones en la interfaz se pueda Iniciar/Pausar/Reanudar el juego: iniciar el juego no se ha iniciado aún, suspender el juego si está en ejecución, reactivar el juego si está suspendido. Para esto tenga en cuenta:

- Al pausar (suspender) el juego, en alguna parte de la interfaz (agregue los componentes que deseé) se debe mostrar:
 - La serpiente viva más larga.
 - La peor serpiente: la que primero murió.

Recuerde que la suspensión de las serpientes NO es instantánea, y que se debe garantizar que se muestre información consistente.

- Creamos la clase Semaforo la cual nos permitirá conocer en qué estado se encuentra el hilo, así mismo nos proporciona una forma de controlar el acceso a ciertos recursos o secciones críticas, en las cuales los hilos podrán consultar o modificar el estado de la bandera para coordinar sus acciones y evitar problemas de concurrencia.

```

src > main > java > snakepackage > J_Snake.java U > J_Snake.java 9+, M
1 package snakepackage;
2
3 import java.util.LinkedList;
4 import java.util.Observable;
5 import java.util.Random;
6 import java.util.concurrent.ConcurrentLinkedQueue;
7
8 import enums.Direction;
9 import enums.GridSize;
10
11 public class Snake extends Observable implements Runnable {
12
13     private int idt;
14     private Cell head;
15     private Cell newCell;
16     private LinkedList<Cell> snakeBody = new LinkedList<Cell>();
17     //private Cell objective = null;
18     private Cell start = null;
19
20     private boolean snakeEnd = false;
21
22     private int direction = Direction.NO_DIRECTION;
23     private final int INIT_SIZE = 3;
24
25     private boolean hasTurbo = false;
26     private int jumps = 0;
27     private boolean isSelected = false;
28     private int growing = 0;

```

- Hicimos los ajustes necesarios para que a través de botones en la interfaz se pueda Iniciar, Pausar y Reanudar el juego como se indica anteriormente.

✓ Modificación de la clase Snake para mostrar la serpiente que primero murió

Vamos a realizar un seguimiento del identificador del primer hilo de la serpiente que muere en el juego. Adicional a esto tenemos un objeto de tipo Semaforo que se utiliza para sincronizar el hilo de la serpiente, para que de esta forma se conozca el estado del semáforo y así saber si el semáforo nos permite avanzar o se encuentra bloqueado y el hilo deba esperar hasta que el semáforo le permita continuar.

```

src > main > java > snakepackage > J_Snake.java U > J_Snake.java 9+, M
1 package snakepackage;
2
3 import java.util.LinkedList;
4 import java.util.Observable;
5 import java.util.Random;
6 import java.util.concurrent.ConcurrentLinkedQueue;
7
8 import enums.Direction;
9 import enums.GridSize;
10
11 public class Snake extends Observable implements Runnable {
12
13     private int idt;
14     private Cell head;
15     private Cell newCell;
16     private LinkedList<Cell> snakeBody = new LinkedList<Cell>();
17     //private Cell objective = null;
18     private Cell start = null;
19
20     private boolean snakeEnd = false;
21
22     private int direction = Direction.NO_DIRECTION;
23     private final int INIT_SIZE = 3;
24
25     private boolean hasTurbo = false;
26     private int jumps = 0;
27     private boolean isSelected = false;
28     private int growing = 0;

```

The screenshot shows the Eclipse IDE interface with the following details:

- Toolbar:** Archivo, Editar, Selección, Ver, Ir, Ejecutar, ...
- Project Explorer (Left):** ARSW_LAB02, src, main, java, enums, snakepackage, Board.java, Cell.java, Semaforo.java, Snake.java, resources, test, target, wait-notify-exercise..., .gitignore, BermudezLadino_Lab..., LICENSE, pom.xml, README.md, RESPUESTAS.txt, snakeraceoutput.png.
- Code Editor (Center):** The active file is Snake.java, showing the following code:

```
private boolean snakeEnd = false;
private int direction = Direction.NO_DIRECTION;
private final int INIT_SIZE = 3;

private boolean hasTurbo = false;
private int jumps = 0;
private boolean isSelected = false;
private int growing = 0;
public boolean goal = false;
public Semaforo semaforo;
public static Integer primeraMuerte = Integer.MIN_VALUE;

public Snake(int idt, Cell head, int direction, Semaforo semaforo) {
    this.idt = idt;
    this.direction = direction;
    this.semaforo = semaforo;
    generateSnake(head);
}

public boolean isSnakeEnd() {
    return snakeEnd;
}

private void generateSnake(Cell head) {
    start = head;
}
```

- Terminal (Bottom):** SALIDA, CONSOLA DE DEPURACIÓN, TERMINAL (77), JAVA PROJECTS, MAVEN.
- System Status (Bottom Left):** 18°C, Parc soleado.
- System Icons (Bottom Right):** powershell, Lector de pantalla optimizado, Lin. 373, col. 1, Espacios: 4, UTF-8, CRLF, Java, Git Graph.
- System Date/Time (Bottom Right):** 11:13 p.m., 1/09/2023.

```
src > main > java > snakepackage > J Snakejava > Snake.java <
43     }
44
45     private void generateSnake(Cell head) {
46         start = head;
47         //Board.gameboard[head.getX()][head.getY()].reserveCell(jumps, idt);
48
49         //Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
50         synchronized(snakeBody){
51             snakeBody.add(head);
52         }
53         growing = INIT_SIZE - 1;
54     }
55
56     @Override
57     public void run() {
58         while (!snakeEnd) {
59             if(!semaforo.getBandera()){
60                 synchronized(semaforo){
61                     try{
62                         semaforo.wait();
63                     }catch(InterruptedException e){
64                         e.printStackTrace();
65                     }
66                 }
67             }
68             snakeCalc();
69         }
70     }
71 }
```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (Left):** Shows the project structure under "ARSW_LAB02". The "src" folder contains "main" and "java" packages. "main.java" includes "enums", "snakepackage", and "Board.java" (5 lines). "java" includes "enums", "snakepackage", and "Semaforo.java" (U). "Semaforo.java" is currently selected and open in the editor.
- Editor Area (Center):** Displays the Java code for "Semaforo.java". The code handles thread synchronization for a "Snake" object, including methods like "setChanged()", "notifyObservers()", and "fixDirection(head)". It also contains a private method "snakeCalc()" which synchronizes access to the "snakeBody" list.
- Terminal (Bottom):** Shows the command line output: "PS C:\Users\luisa\OneDrive\Documentos\ARWS-2\LAB02\ARSH_Lab02>".
- Status Bar (Bottom):** Provides information about the current file: "feature/Luisa/Parte-II-Punto04.java", line count (Lin. 373), character count (col. 1), and encoding (UTF-8).

```
private void snakeCalc() {
    //Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
    synchronized(snakeBody){
        head = snakeBody.peekFirst();
    }

    newCell = head;

    newCell = changeDirection(newCell);

    randomMovement(newCell);

    checkIfFood(newCell);
    checkIfJumpPad(newCell);
    checkIfTurboBoost(newCell);
    checkIfBarrier(newCell);

    //Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
    synchronized(snakeBody){
        snakeBody.push(newCell);
    }

    if (growing <= 0) {
        newCell = snakeBody.peekLast();
        snakeBody.remove(snakeBody.peekLast());
    }
}
```

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_LAB02
- Open Files:** SnakeApp.java, Semaforo.java, and Snake.java
- Code Editor:** Snake.java (selected)
- Code Content:** Java code for a Snake game, specifically the Snake class. It includes methods like checkIfBarrier, fixDirection, and checkIfOwnBody.
- Toolbars:** Archivo, Editar, Selección, Ver, Ir, Ejecutar, etc.
- Status Bar:** Lector de pantalla optimizado, Lin. 373, col. 1, Espacios: 4, UTF-8, CRLF, Java, powershell, ESP LAA, 11:15 p. m., 1/09/2023

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_LAB02
- Open Files:** SnakeApp.java, Semaforo.java, and Snake.java
- Code Editor:** Snake.java (selected)
- Code Content:** Java code for a Snake game, specifically the Snake class. It includes methods like fixDirection and checkIfOwnBody.
- Toolbars:** Archivo, Editar, Selección, Ver, Ir, Ejecutar, etc.
- Status Bar:** Lector de pantalla optimizado, Lin. 373, col. 1, Espacios: 4, UTF-8, CRLF, Java, powershell, ESP LAA, 11:15 p. m., 1/09/2023

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_LAB02
- Open Files:** SnakeApp.java, Semaforo.java, and Snake.java
- Code Editor:** Snake.java (selected)
- Code Content:** The code implements logic for a snake game, including methods like `checkIfOwnBody`, `randomMovement`, and `checkIfTurboBoost`. It uses `Random` to determine movement direction and `Board` to manage the game board.
- Terminal:** Shows the command PS C:\Users\luisa\OneDrive\Documentos\ARWS-2\LAB02\ARSW_Lab02>
- Status Bar:** Includes file paths, line numbers, and system information (18°C, 100% battery, 115 p.m., 1/09/2023).

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_LAB02
- Open Files:** SnakeApp.java, Semaforo.java, and Snake.java
- Code Editor:** Snake.java (selected)
- Code Content:** The code includes logic for a "turbo boost" feature. It checks if a cell is a turbo boost cell and updates the board accordingly. It also handles jump pads.
- Terminal:** Shows the command PS C:\Users\luisa\OneDrive\Documentos\ARWS-2\LAB02\ARSW_Lab02>
- Status Bar:** Includes file paths, line numbers, and system information (18°C, 100% battery, 116 p.m., 1/09/2023).

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_LAB02
- Open Files:** SnakeApp.java, Semaforo.java, Snake.java
- Code Editor:** Snake.java (highlighted)
- Code Content:**

```
src > main > java > snakepackage > J_Snakejava > Semaforo.java U J_Snake.java 9+ M
  ...
  //Se sincronizo el metodo checkIfJumpPad para que solo un hilo pueda ejecutarlo a la vez
  private synchronized void checkIfJumpPad(Cell newCell) {
    ...
  }
  ...
  //Se sincronizo el metodo checkIfFood para que solo un hilo pueda ejecutarlo a la vez
  private synchronized void checkIfFood(Cell newCell) {
    Random random = new Random();
    ...
  }
  ...
  
```

- Terminal:** powershell
- Status Bar:** Lín. 373, col. 1 Espacios: 4 UTF-8 CRLF () Java 18°C Parc. soleado 116 p. m. 1/09/2023

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_LAB02
- Open Files:** SnakeApp.java, Semaforo.java, Snake.java
- Code Editor:** Snake.java (highlighted)
- Code Content:**

```
src > main > java > snakepackage > J_Snakejava > Semaforo.java U J_Snake.java 9+ M
  ...
  //Se sincronizo el metodo checkIfFood para que solo un hilo pueda ejecutarlo a la vez
  private synchronized void checkIfFood(Cell newCell) {
    Random random = new Random();
    ...
  }
  ...
  if (Board.gameboard[newCell.getX()][newCell.getY()].isFood()) {
    ...
  }
  ...
  for (int i = 0; i < Board.NR_FOOD; i++) {
    ...
  }
  ...
  
```

- Terminal:** powershell
- Status Bar:** Lín. 373, col. 1 Espacios: 4 UTF-8 CRLF () Java 18°C Parc. soleado 116 p. m. 1/09/2023

The screenshot shows a Java code editor interface with the following details:

- File Explorer (EXPLORADOR):** Shows the project structure for "ARSW_LAB02".
- Code Editor:** Displays the content of the `SnakeApp.java` file.
- Terminal:** Shows the command line prompt: `PS C:\Users\luisa\OneDrive\Documentos\ARWS-2\LAB02\ARSW_Lab02>`.
- Bottom Bar:** Includes icons for search, file operations, and system status (language, battery, date).

```
src > main > java > snakepackage > J_Snakejava > Snake
248     }
249
250     private Cell changeDirection(Cell newCell) {
251         // Avoid out of bounds
252
253         while (direction == Direction.UP && (newCell.getY() - 1) < 0) {
254             if ((head.getX() - 1) < 0) {
255                 this.direction = Direction.RIGHT;
256             } else if ((head.getX() + 1) == Gridsize.GRID_WIDTH) {
257                 this.direction = Direction.LEFT;
258             } else {
259                 randomMovement(newCell);
260             }
261         }
262         while (direction == Direction.DOWN
263             && (head.getY() + 1) == Gridsize.GRID_HEIGHT) {
264             if ((head.getX() - 1) < 0) {
265                 this.direction = Direction.RIGHT;
266             } else if ((head.getX() + 1) == Gridsize.GRID_WIDTH) {
267                 this.direction = Direction.LEFT;
268             } else {
269                 randomMovement(newCell);
270             }
271         }
272         while (direction == Direction.LEFT && (head.getX() - 1) < 0) {
273             if ((newCell.getY() - 1) < 0) {
274                 this.direction = Direction.DOWN;
275             } else if ((newCell.getY() + 1) == Gridsize.GRID_HEIGHT) {
276                 this.direction = Direction.UP;
277             } else {
278                 randomMovement(newCell);
279             }
280         }
281         while (direction == Direction.RIGHT
282             && (head.getX() + 1) == Gridsize.GRID_WIDTH) {
283             if ((newCell.getY() - 1) < 0) {
284                 this.direction = Direction.DOWN;
285             } else if ((newCell.getY() + 1) == Gridsize.GRID_HEIGHT) {
286                 this.direction = Direction.UP;
287             } else {
288                 randomMovement(newCell);
289             }
290         }
291
292         switch (direction) {
293             case Direction.UP:
294                 newCell = Board.gameboard[head.getX()][head.getY() - 1];
295                 break;
296             case Direction.DOWN:
```

The screenshot shows the same Java code editor interface, but with a different section of the `SnakeApp.java` file displayed in the code editor.

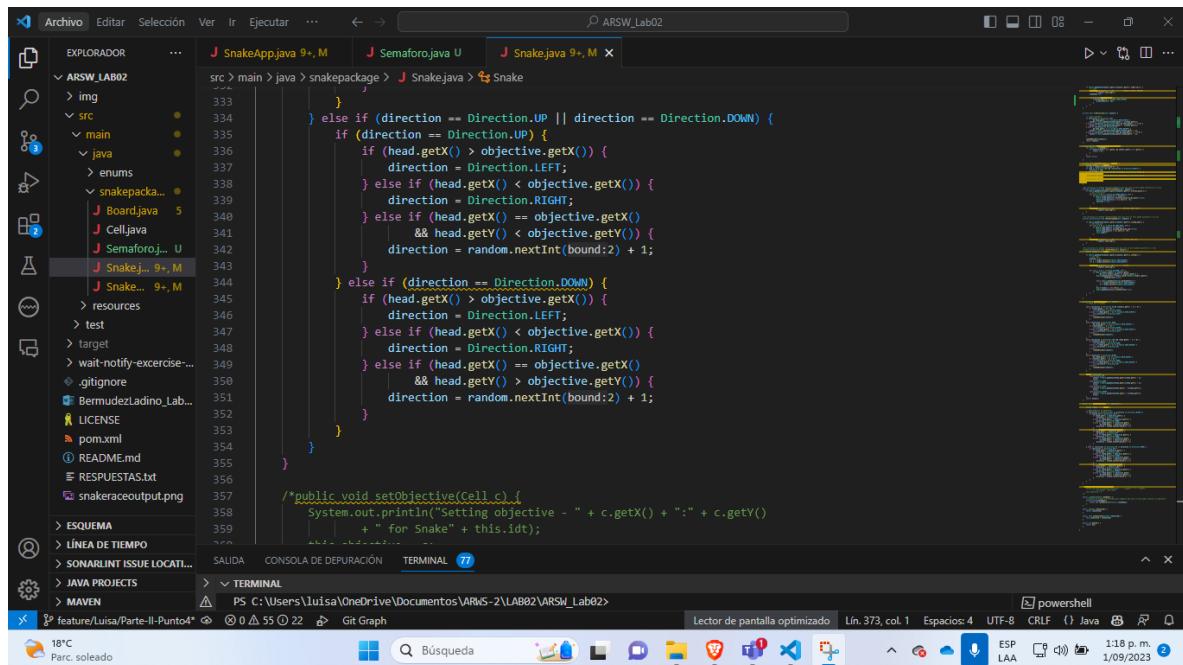
```
src > main > java > snakepackage > J_Snakejava > Snake
248     }
249
250     private Cell changeDirection(Cell newCell) {
251         // Avoid out of bounds
252
253         while (direction == Direction.UP && (newCell.getY() - 1) < 0) {
254             if ((head.getX() - 1) < 0) {
255                 this.direction = Direction.RIGHT;
256             } else if ((head.getX() + 1) == Gridsize.GRID_WIDTH) {
257                 this.direction = Direction.LEFT;
258             } else {
259                 randomMovement(newCell);
260             }
261         }
262         while (direction == Direction.DOWN
263             && (head.getY() + 1) == Gridsize.GRID_HEIGHT) {
264             if ((head.getX() - 1) < 0) {
265                 this.direction = Direction.RIGHT;
266             } else if ((head.getX() + 1) == Gridsize.GRID_WIDTH) {
267                 this.direction = Direction.LEFT;
268             } else {
269                 randomMovement(newCell);
270             }
271         }
272         while (direction == Direction.LEFT && (head.getX() - 1) < 0) {
273             if ((newCell.getY() - 1) < 0) {
274                 this.direction = Direction.DOWN;
275             } else if ((newCell.getY() + 1) == Gridsize.GRID_HEIGHT) {
276                 this.direction = Direction.UP;
277             } else {
278                 randomMovement(newCell);
279             }
280         }
281         while (direction == Direction.RIGHT
282             && (head.getX() + 1) == Gridsize.GRID_WIDTH) {
283             if ((newCell.getY() - 1) < 0) {
284                 this.direction = Direction.DOWN;
285             } else if ((newCell.getY() + 1) == Gridsize.GRID_HEIGHT) {
286                 this.direction = Direction.UP;
287             } else {
288                 randomMovement(newCell);
289             }
289         }
290
291         switch (direction) {
292             case Direction.UP:
293                 newCell = Board.gameboard[head.getX()][head.getY() - 1];
294                 break;
295             case Direction.DOWN:
```

The screenshot shows an IDE interface with the following details:

- File Explorer (EXPLORADOR):** Shows the project structure under ARSW_LAB02, including subfolders like img, src, main, java, enums, snakepackage, and resources.
- Code Editor:** Displays the `Snake.java` file with approximately 370 lines of Java code. The code includes logic for moving the snake based on its current direction (UP, DOWN, LEFT, RIGHT) and a method to search for an objective cell.
- Terminal:** Shows the command `PS C:\Users\luisa\OneDrive\Documentos\ARWS-2\LAB02\ARSW_Lab02>`.
- Bottom Bar:** Includes icons for search, file operations, and system status (language ESP/LAA, date 1/09/2023, time 11:17 p.m.).

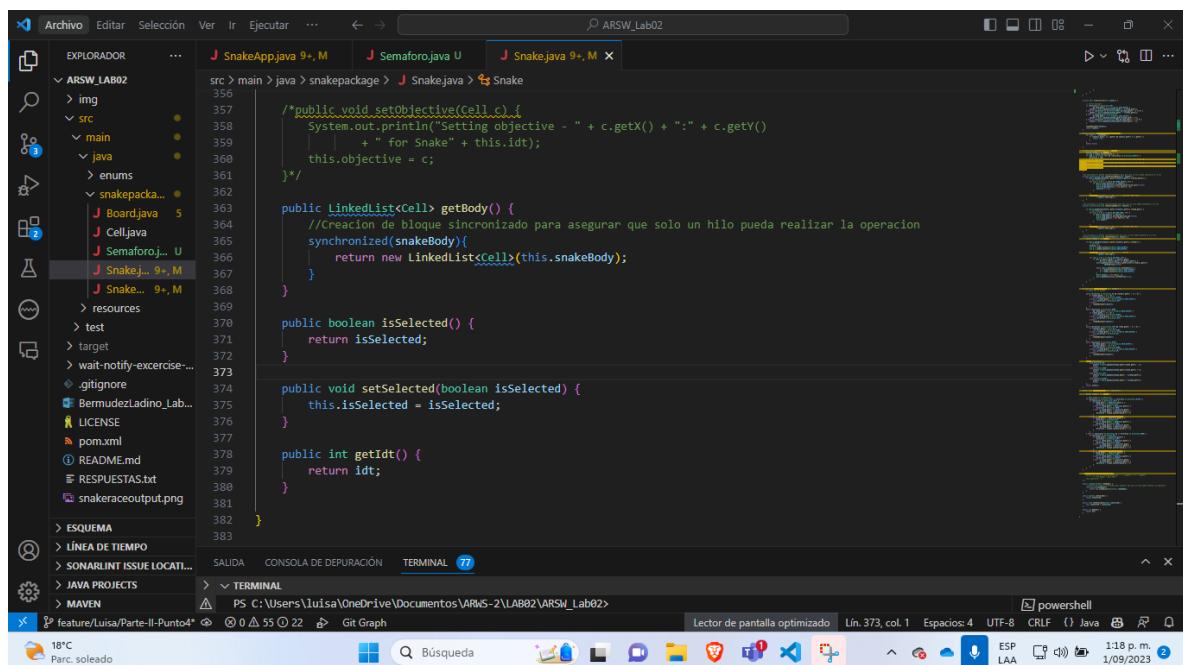
The screenshot shows an IDE interface with the following details:

- File Explorer (EXPLORADOR):** Shows the project structure under ARSW_LAB02, including subfolders like img, src, main, java, enums, snakepackage, and resources.
- Code Editor:** Displays the `Snake.java` file with approximately 370 lines of Java code. The code includes a modified version of the `searchObjective` method, which now handles more cases for moving directly to the objective cell.
- Terminal:** Shows the command `PS C:\Users\luisa\OneDrive\Documentos\ARWS-2\LAB02\ARSW_Lab02>`.
- Bottom Bar:** Includes icons for search, file operations, and system status (language ESP/LAA, date 1/09/2023, time 11:17 p.m.).



ARSW_Lab02

```
src > main > java > snakepackage > J_Snakejava > Snake
333     }
334 } else if (direction == Direction.UP || direction == Direction.DOWN) {
335     if (head.getX() > objective.getX()) {
336         direction = Direction.LEFT;
337     } else if (head.getX() < objective.getX()) {
338         direction = Direction.RIGHT;
339     } else if (head.getX() == objective.getX()
340             && head.getY() < objective.getY()) {
341         direction = random.nextInt(bound/2) + 1;
342     }
343 } else if (direction == Direction.DOWN) {
344     if (head.getX() > objective.getX()) {
345         direction = Direction.LEFT;
346     } else if (head.getX() < objective.getX()) {
347         direction = Direction.RIGHT;
348     } else if (head.getX() == objective.getX()
349             && head.getY() > objective.getY()) {
350         direction = random.nextInt(bound/2) + 1;
351     }
352 }
353 /*public void setObjective(Cell c) {
354     System.out.println("Setting objective - " + c.getX() + ":" + c.getY()
355                         + " for Snake" + this.idt);
356 }*/
357
358 public void setObjective(Cell c) {
359     System.out.println("Setting objective - " + c.getX() + ":" + c.getY()
360                         + " for Snake" + this.idt);
361     this.objective = c;
362 }
363
364 public LinkedList<Cell>getBody() {
365     //Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
366     synchronized(snakeBody){
367         return new LinkedList<Cell>(this.snakeBody);
368     }
369 }
370
371 public boolean isSelected() {
372     return isSelected;
373 }
374
375 public void setSelected(boolean isSelected) {
376     this.isSelected = isSelected;
377 }
378
379 public int getIdt() {
380     return idt;
381 }
382 }
```



```
src > main > java > snakepackage > J_Snakejava > Snake
356     /*
357      *public void setObjective(Cell c) {
358      *    System.out.println("Setting objective - " + c.getX() + ":" + c.getY()
359      *                        + " for Snake" + this.idt);
360      *    this.objective = c;
361      }*/
362
363 public LinkedList<Cell>getBody() {
364     //Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
365     synchronized(snakeBody){
366         return new LinkedList<Cell>(this.snakeBody);
367     }
368 }
369
370 public boolean isSelected() {
371     return isSelected;
372 }
373
374 public void setSelected(boolean isSelected) {
375     this.isSelected = isSelected;
376 }
377
378 public int getIdt() {
379     return idt;
380 }
381 }
```

✓ **Modificación de la clase SnakeApp para mostrar la serpiente que primero murió**

Se agregaron los botones Iniciar y Reanudar a la interfaz con sus respectivas acciones teniendo en cuenta las instrucciones del enunciado.

Adicional a esto se agregó el botón Pausar para que cuando el juego se suspenda muestre la peor serpiente, es decir, la primera en morir.

```
public class SnakeApp {
    private Semaforo semaforo;
    private static SnakeApp app;
    public static final int MAX_THREADS = 8;
    Snake[] snakes = new Snake[MAX_THREADS];
    private static final Cell[] spawn = {
        new Cell(x:1, (GridSize.GRID_HEIGHT / 2) / 2),
        new Cell(GridSize.GRID_WIDTH - 2,
            3 * (GridSize.GRID_HEIGHT / 2) / 2),
        new Cell(3 * (GridSize.GRID_WIDTH / 2) / 2, y:1),
        new Cell((GridSize.GRID_WIDTH / 2) / 2, GridSize.GRID_HEIGHT - 2),
        new Cell(x:1, 3 * (GridSize.GRID_HEIGHT / 2) / 2),
        new Cell(GridSize.GRID_WIDTH - 2, (GridSize.GRID_HEIGHT / 2) / 2),
        new Cell((GridSize.GRID_WIDTH / 2) / 2, y:1),
        new Cell(3 * (GridSize.GRID_WIDTH / 2) / 2,
            GridSize.GRID_HEIGHT - 2);
    };
    private JFrame frame;
    private static Board board;
    int nr_selected = 0;
    Thread[] thread = new Thread[MAX_THREADS];
    public SnakeApp() {
        Dimension dimension = Toolkit.getDefaultToolkit().getScreenSize();
        frame = new JFrame(title:"The Snake Race");
        frame.setLayout(new BorderLayout());
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // frame.setSize(618, 540);
        frame.setSize(GridSize.GRID_WIDTH * GridSize.WIDTH_BOX + 17,
            GridSize.GRID_HEIGHT * GridSize.HEIGHT_BOX + 40);
        frame.setLocation(dimension.width / 2 - frame.getWidth() / 2,
            dimension.height / 2 - frame.getHeight() / 2);
        board = new Board();
        frame.add(board,BorderLayout.CENTER);
        JPanel actionsPabel=new JPanel();
        actionsPabel.setLayout(new FlowLayout());
        //actionsPabel.add(new JButton("Action"));
        JButton bAction = new JButton(text:"Action ");
        JButton bPausar = new JButton(text:"Pausar ");
        bPausar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e){
                if(semaforo.getBandera()){
                    semaforo.switchBandera();
                    Intento numeroMuerto = Snake.numeroMuerto;
                }
            }
        });
        actionsPabel.add(bAction);
        actionsPabel.add(bPausar);
        frame.add(actionsPabel,BorderLayout.SOUTH);
    }
}
```

```
public void actionPerformed(ActionEvent e){
    if(semaforo.getBandera()){
        semaforo.switchBandera();
        Intento numeroMuerto = Snake.numeroMuerto;
    }
}
```

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_Lab02
- Open Files:** SnakeApp.java, Semaforo.java, Snake.java
- Code Editor:** SnakeApp.java (selected tab)
- Code Content:**

```
src > main > java > snakepackage > J SnakeApp.java > SnakeApp.java
```

```
67
68     bPausar.addActionListener(new ActionListener() {
69         @Override
70         public void actionPerformed(ActionEvent e){
71             if(semaforo.getBandera()){
72                 semaforo.switchBandera();
73                 Integer primeraMuerte = Snake.primerMuerte;
74                 JOptionPane.showMessageDialog(parentComponent:null, (primeraMuerte != Integer.MIN_VALUE)? "La primera s
75             }
76         });
77     });
78
79     JButton bReanudar = new JButton(text:"Reanudar ");
80
81     bReanudar.addActionListener(new ActionListener() {
82         @Override
83         public void actionPerformed(ActionEvent e ){
84             if(!semaforo.getBandera()){
85                 semaforo.switchBandera();
86                 synchronized(semaforo){
87                     semaforo.notifyAll();
88                 }
89             }
90         });
91     });
92
93     actionsBPabel.addAction(bAction);
94     actionsPabel.addAction(bPausar);
```

- Terminal:** powershell
- Status Bar:** Lector de pantalla optimizado, Líne. 23, Col. 14 (8 seleccionada), Espacios: 4, UTF-8, CRLF, Java, 18°C, Mayorm. nublado, 11:0 p. m., 1/09/2023

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_Lab02
- Open Files:** SnakeApp.java, Semaforo.java, Snake.java
- Code Editor:** SnakeApp.java (selected tab)
- Code Content:**

```
src > main > java > snakepackage > J SnakeApp.java > SnakeApp.java
```

```
91
92
93     actionsBPabel.addAction(bAction);
94     actionsPabel.addAction(bPausar);
95     actionsPabel.addAction(bReanudar);
96
97     frame.add(actionsBPabel,BorderLayout.SOUTH);
98
99 }
100
101 Run | Debug
102 public static void main(String[] args) {
103     app = new SnakeApp();
104     app.init();
105 }
106
107 private void init() {
108
109     semaforo = new Semaforo();
110     semaforo.setBandera(bandera:true);
111
112     for (int i = 0; i != MAX_THREADS; i++) {
113         snakes[i] = new Snake(i + 1, spawn[i], i + 1, semaforo);
114         snakes[i].addObserver(board);
115         thread[i] = new Thread(snakes[i]);
116         thread[i].start();
117     }
118 }
```

- Terminal:** powershell
- Status Bar:** Lector de pantalla optimizado, Líne. 23, Col. 14 (8 seleccionada), Espacios: 4, UTF-8, CRLF, Java, 18°C, Mayorm. nublado, 11:11 p. m., 1/09/2023

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_Lab02
- Open Files:** SnakeApp.java, Semaforo.java, Snake.java
- Code Editor:** SnakeApp.java (selected)
- Code Content:**

```
src > main > java > snakepackage > J SnakeApp.java > Semaforo.java > Snake.java
```

```
private void init() {  
    semaforo = new Semaforo();  
    semaforo.setBandera(bandera:true);  
  
    for (int i = 0; i != MAX_THREADS; i++) {  
        snakes[i] = new Snake(i + 1, spawn[i], i + 1,semaforo);  
        snakes[i].addObserver(board);  
        thread[i] = new Thread(snakes[i]);  
        thread[i].start();  
    }  
  
    frame.setVisible(b:true);  
  
    /*while_(true){  
        int x = 0;  
        for (int i = 0; i != MAX_THREADS; i++) {  
            if (snakes[i].isSnakeEnd() == true) {  
                x++;  
            }  
        }  
        if (x == MAX_THREADS) {  
            break;  
        }  
    }*/  
}
```

- Terminal:** PS C:\Users\luisa\OneDrive\Documentos\ARWS-2\LAB02\ARSW_Lab02>
- Status Bar:** Lector de pantalla optimizado, Lín. 23, Col. 14 (8 seleccionada), Espacios: 4, UTF-8, CRLF, Java, 111 p. m., 1/09/2023

The screenshot shows an IDE interface with the following details:

- Project Explorer:** ARSW_Lab02
- Open Files:** SnakeApp.java, Semaforo.java, Snake.java
- Code Editor:** SnakeApp.java (selected)
- Code Content:**

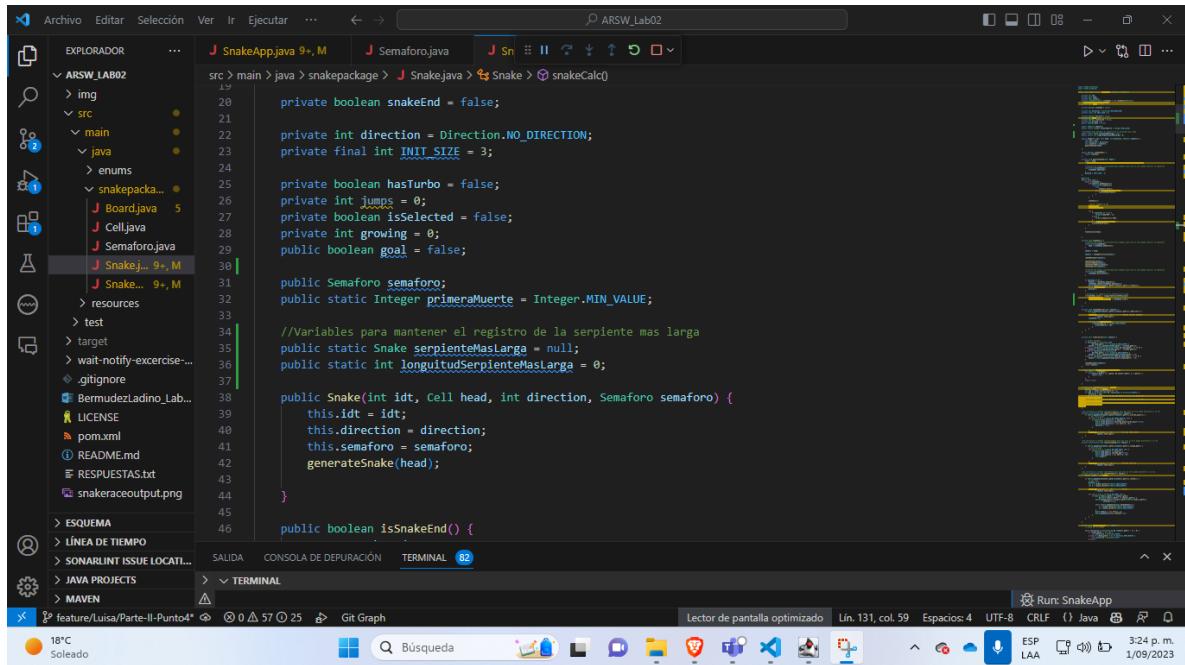
```
src > main > java > snakepackage > J SnakeApp.java > Semaforo.java > Snake.java
```

```
//Espera a que todos los hilos finalicen su ejecucion  
for(int i=0; i!=MAX_THREADS; i++){  
    try{  
        //Espera a que todos los hilos terminen para consolidar los resultados  
        thread[i].join();  
    }catch(InterruptedException e){  
        e.printStackTrace();  
    }  
}  
  
System.out.println(x:"Thread (snake) status:");  
for (int i = 0; i != MAX_THREADS; i++) {  
    System.out.println("[" + i + "] :" + thread[i].getState());  
}  
  
public static SnakeApp getApp() {  
    return app;  
}
```

- Terminal:** PS C:\Users\luisa\OneDrive\Documentos\ARWS-2\LAB02\ARSW_Lab02>
- Status Bar:** Lector de pantalla optimizado, Lín. 23, Col. 14 (8 seleccionada), Espacios: 4, UTF-8, CRLF, Java, 111 p. m., 1/09/2023

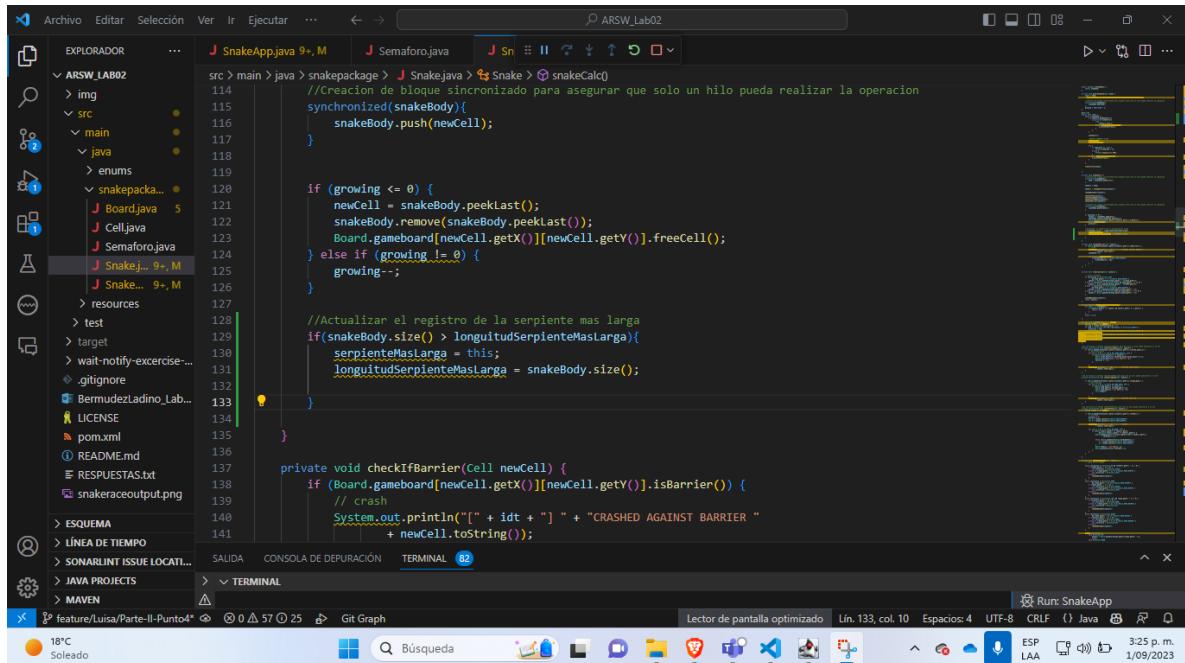
✓ **Modificación de la clase Snake para mostrar la serpiente más larga**

Vamos a mantener un registro de la serpiente más larga.



The screenshot shows the VS Code interface with the project ARSW_Lab02 open. The Explorer sidebar on the left lists files like SnakeApp.java, Semaforo.java, and Board.java. The main editor area displays the SnakeApp.java code. A new variable 'serpienteMasLarga' is added to the class, initialized to null. A static variable 'longitudSerpienteMasLarga' is also added and initialized to 0. The code then checks if the snake's size exceeds this value and updates it if necessary. The status bar at the bottom shows the file is 131 lines long, in UTF-8 encoding, and the current date and time.

```
private boolean snakeEnd = false;
private int direction = Direction.NO_DIRECTION;
private final int INIT_SIZE = 3;
private boolean hasTurbo = false;
private int jumps = 0;
private boolean isSelected = false;
private int growing = 0;
public boolean goal = false;
public Semaforo semaforo;
public static Integer primeraMuerte = Integer.MIN_VALUE;
//Variables para mantener el registro de la serpiente mas larga
public static Snake serpienteMasLarga = null;
public static int longitudSerpienteMasLarga = 0;
public Snake(int idt, Cell head, int direction, Semaforo semaforo) {
    this.idt = idt;
    this.direction = direction;
    this.semaforo = semaforo;
    generateSnake(head);
}
public boolean isSnakeEnd() {
```



The screenshot shows the VS Code interface with the project ARSW_Lab02 open. The Explorer sidebar on the left lists files like SnakeApp.java, Semaforo.java, and Board.java. The main editor area displays the modified SnakeApp.java code. The 'checkIfBarrier' method now includes a check for barriers. If a barrier is found, it prints a crash message and ends the game. The status bar at the bottom shows the file is 133 lines long, in UTF-8 encoding, and the current date and time.

```
//Creacion de bloque sincronizado para asegurar que solo un hilo pueda realizar la operacion
synchronized(snakeBody){
    snakeBody.push(newCell);
}

if (growing <= 0) {
    newCell = snakeBody.peekLast();
    snakeBody.remove(snakeBody.peekLast());
    Board.gameboard[newCell.getX()][newCell.getY()].freeCell();
} else if (growing >= 0) {
    growing--;
}

//Actualizar el registro de la serpiente mas larga
if(snakeBody.size() > longitudSerpienteMasLarga){
    serpienteMasLarga = this;
    longitudSerpienteMasLarga = snakeBody.size();
}
```

✓ **Modificación de la clase SnakeApp para mostrar la serpiente más larga**

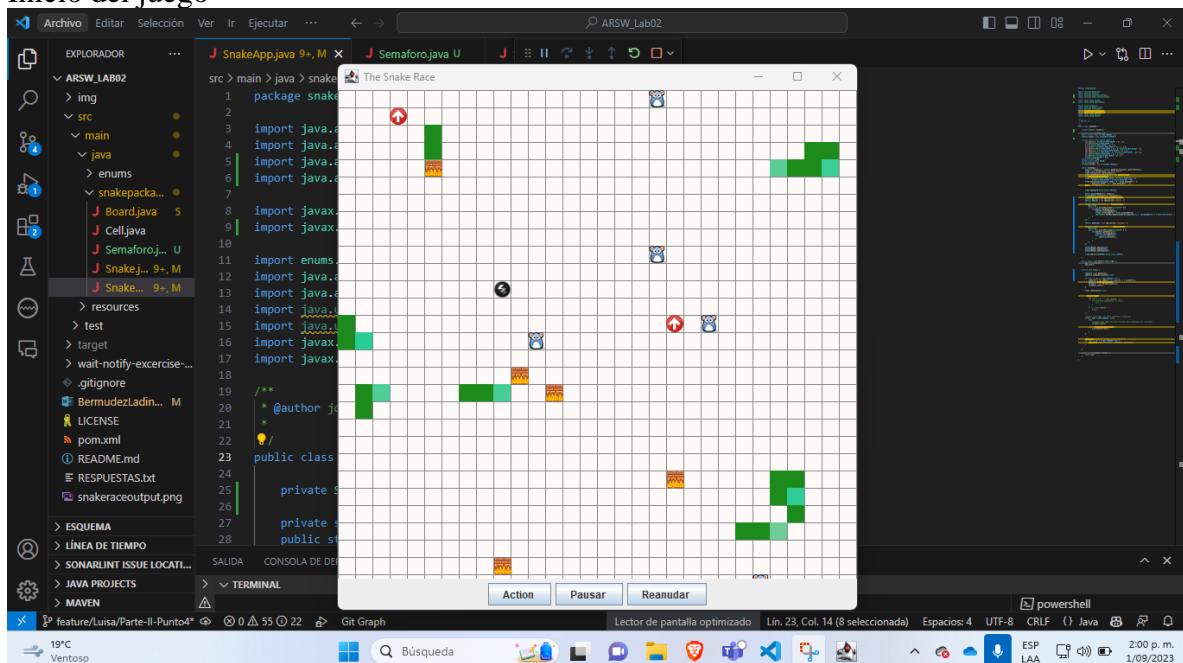
Se agregó un mensaje para qué momento de dar click en el botón Pausar muestre cual es la serpiente mas larga y su longitud.

```
src > main > java > snakepackage > J_SnakeApp.java ...
```

```
59
60         frame.add(board,BorderLayout.CENTER);
61
62     JPanel actionsBPabel=new JPanel();
63     actionsBPabel.setLayout(new FlowLayout());
64     //actionsBPabel.add(new JButton("Action"));
65     JButton bAction = new JButton(text:"Action ");
66     JButton bPausar = new JButton(text:"Pausar ");
67
68     bPausar.addActionListener(new ActionListener() {
69         @Override
70         public void actionPerformed(ActionEvent e){
71             if(semaforo.getBandera()){
72                 semaforo.switchBandera();
73                 Integer primeraMuerte = Snake.primerMuerte;
74                 Integer serpienteMasLarga = Snake.serpienteMasLarga.getId();
75                 JOptionPane.showMessageDialog(parentComponent:null, (primeraMuerte != Integer.MIN_VALUE)? "La primera serpiente que muere es la serpiente "+serpienteMasLarga+", con una longitud de "+(serpienteMasLarga.length)+" bloques": "La primera serpiente que muere es la serpiente "+serpienteMasLarga+", con una longitud de "+(serpienteMasLarga.length)+" bloques");
76             }
77         }
78     });
79
80     JButton bReanudar = new JButton(text:"Reanudar ");
81
82     bReanudar.addActionListener(new ActionListener() {
83         @Override
84         public void actionPerformed(ActionEvent e){
85             if(!semaforo.getBandera()){
86                 semaforo.setBandera(true);
87             }
88         }
89     });
90
91     actionsBPabel.add(bAction);
92     actionsBPabel.add(bPausar);
93     actionsBPabel.add(bReanudar);
94
95     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
96     frame.setVisible(true);
97 }
```

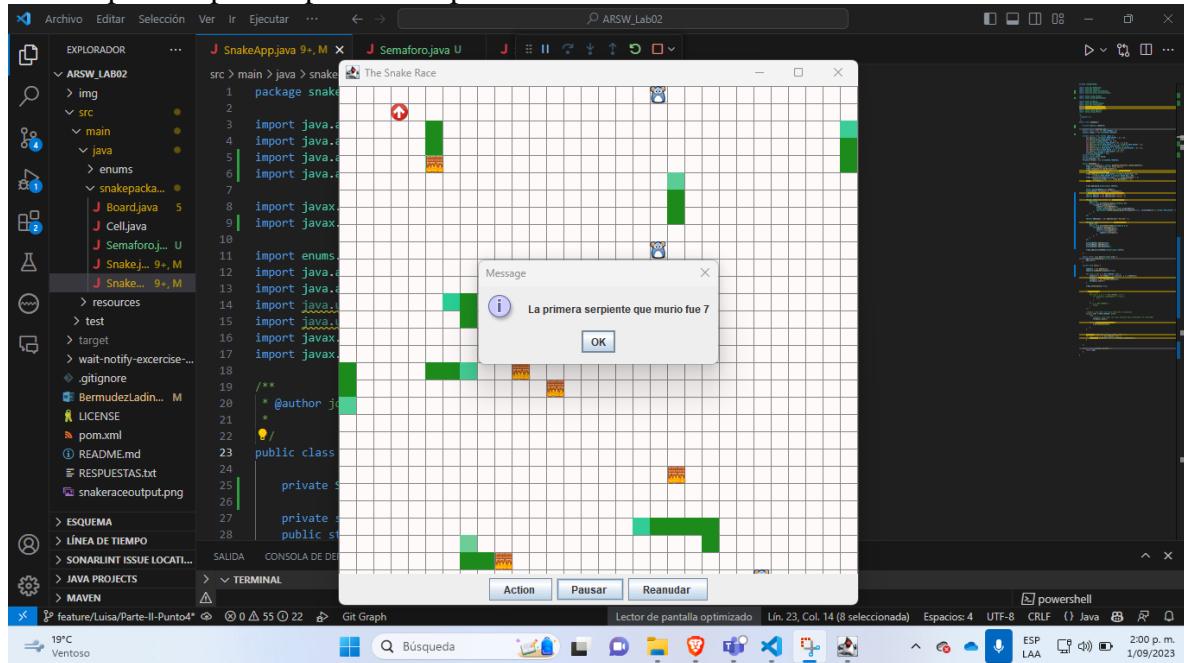
➤ **Resultados Ejecución:**

Inicio del juego

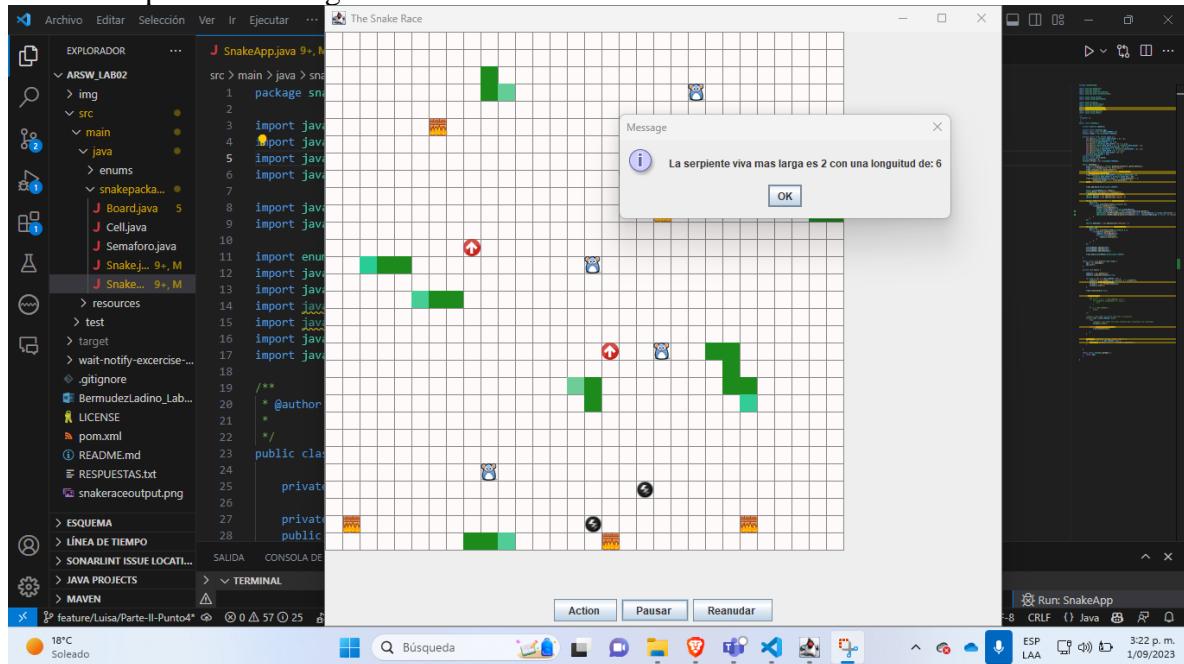


Pausar el juego

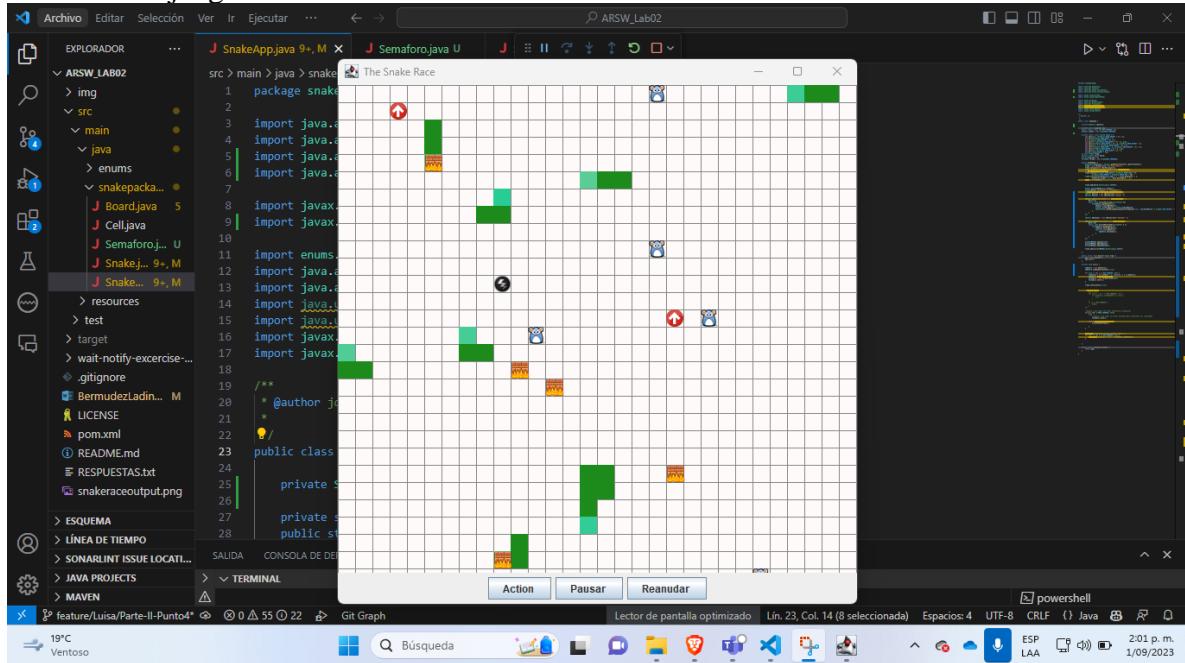
- La peor serpiente: primera serpiente en morir



- La serpiente más larga



Reanudar el juego



III. Conclusiones

- Durante el desarrollo de este laboratorio, hemos explorado los conceptos fundamentales de programación concurrente. Hemos aprendido cómo manejar múltiples hilos de ejecución de manera eficiente y segura en entornos paralelos.
- Hemos aplicado los mecanismos de sincronización proporcionados por el lenguaje Java, como wait, notify y notifyAll, para controlar el comportamiento de los hilos. Esto nos ha permitido pausar y reanudar hilos de manera sincronizada.
- Identificamos posibles condiciones de carrera en el código y con el fin de tomar acciones para evitarlas. Esto incluye el uso de bloqueos synchronized para asegurar el acceso exclusivo a secciones críticas del código.

url repositorio:https://github.com/ARSW2023-2/ARSW_Lab02.git