Proyecto 2. Una situación cotidiana paralelizable

Situación: Un parque de diversiones y sus juegos mecánicos.

Para explicar la situación, se aclaran los conceptos que se van a manejar:

- Vagón o unidad: En este documento se le llama de esta forma al conjunto de asientos que van juntos en un mismo viaje. Ver Fig.01
- Trayectoria o recorrido: Se le llama de esta forma a todo el camino que van a transitar las unidades. Cada juego tiene una longitud de recorrido distinta.
- Sector: Para abstraer el problema, se necesitó dividir todo el recorrido por sectores, por lo que cada unidad se encuentra en determinado sector y avanza por los rieles ocupando y desocupando sectores.

Fig. 01. Ejemplo de Unidad con asientos.

En un parque de diversiones o ferias hay juegos mecánicos en los cuales se necesitan tener concurrencia en diferentes aspectos:

1. Las unidades o vagones necesitan ocupar determinado sector y ningún otro puede estar en el mismo sector al mismo tiempo.
2. Las personas que llegan a la fila para entrar al juego pueden pasar a sentarse en los asientos de las unidades, pero solo pueden pasar un número máximo de personas por unidad, este número corresponde con el número de asientos.
3. Como los juegos consumen recursos energéticos, se tiene un número mínimo de personas por unidad, si la cantidad de personas esperando no es mayor al número mínimo, se tiene que esperar a que lleguen más personas.
4. Al terminar el recorrido, las personas tienen que salir de la unidad de forma ordenada antes de que el vagón se vaya.

Consecuencias de no controlar la concurrencia en los casos mostrados:

1. Las unidades podrían ocupar el mismo lugar. En el mundo real esto seria igual a un choque entre unidades.
2. Las personas en la cola pueden pasar sin límite, lo que podría ocasionar que se exceda el numero de asientos. En la vida real esto no puede ocurrir por seguridad de todas las personas.
3. Si no se lleva a cabo el número mínimo por unidades, se desperdiciarían recursos. Imaginemos mover al "Superman" de SixFlags con solo una persona abordo.
4. Si no detenemos la salida de las personas, antes de concluir el viaje, las personas saldrían.

¿Hay eventos concurrentes para los cuales el ordenamiento relativo no resulta importante?

Para modelar la situación, las personas no necesitan llegar en orden a la fila del juego mecánico y las personas se pueden ir subiendo mientras el Juego mecánico decida sin necesidad de estar ordenados.

Documentación:

Mecanismos de sincronización empleados:

Los mecanismos para resolver los problemas de concurrencia se encapsularon en objetos para facilitar una lectura del código y reducir la posibilidad de errores.

Para resolver el problema de concurrencia de los vagones en los sectores, se implementó un Array de mutex, cada índice representaba el número del sector y los vagones. Cada vez que un vagón avanzaba, realiza un aquire() al mutex que le corresponde. Y se encargaba de liberar el mutex anterior, para dejar a otro vagón avanzar.

En el caso de los problemas con las personas, se decidió implementar barreras para no dejar pasar mas de las que deberían. Se implementó mediante señalizaciones para que no presentaran problemas como los ocurridos en clase.

Un problema extra se presentó, cuando se esperaban mas personas y el programa ya no lanzaba mas hilos personas, se quedaba esperando. Es por ello que se diseñó una manera de terminar la ejecución y así ningún hilo se quedara funcionando esperando.

El problema principal es la interacción de las personas con los vagones, al mismo tiempo los vagones interactuaban con otros vagones, es por ello que se decidió implementar un modelo Jefe-Trabajador con una perspectiva diferente. El jefe es el hilo creado a partir de la Clase JuegoMecanico, este crea a sus hilos Vagones automáticamente dependiendo de los hilos Persona que reciba en su cola.

Lógica de Operación:

Estado Compartido y La interacción entre los hilos:

El hilo jefe se encargará de crear su Cola de entrada al juego, y su Cola de salida de este. Estas colas funcionan como barreras, para un buen funcionamiento necesitábamos saber cuántas personas había en la cola. La clase Semaphore no nos proporciona el numero de hilos que hacen aquire(), es por ello que se encapsuló el Semaphore de la barrera junto con un número que contaba las personas en la cola. De este modo, cuando una persona se formaba en la cola llamando el método

formarCola(), se incrementaba el numero y se realizaba un aquire(). Ocurria lo contrario para salirCola(), decrementar el numero y realizar un release(). Es por ello que la Cola de entrada debía de ser compartida con todos los hilos persona y al mismo tiempo debe ser administrado por el hilo jefe, para dejar pasar a un número que el decidiera prudente.

Para la cola de salida, cada vagón decidía cuantas personas liberar, dependiendo de las personas que llevara, las personas se formaban con formarCola() y el vagón las sacaba con salirCola(). Es por ello que la Cola de Salida debe ser proporcionada por el JuegoMecanico a sus hilos vagon y a los hilos persona.

Por otro lado, los mutex de la trayectoria se encapsularon en una clase en donde cada que se llamara a su método ocuparLugar() se realizaba un aquire() y lo contrario para desocuparLugar(), este objeto debía de ser compartido por todos los vagones. Para esto, el hilo JuegoMecanico se lo proporcionaba a cada unidad que creara.

Estas variables compartidas se distribuyeron por objetos ya que una persona no debería poder hacer cambios en las trayectorias, y un vagón que ya está andando no debería poder desencolar personas de la entrada, solo de la salida. Con la finalidad de cumplir con el modelo de programación orientada a objetos.

Descripción Algorítmica del Avance de Cada Hilo/proceso

Persona

Descripción

La persona se forma, y se detiene para esperar a que el juego le indique que puede pasar (aquire). Una vez que el juego lo despierta, revisa si el juego esta funcionando, si el juego no esta funcionando, se va. En caso de que esté funcionando, se disfruta el juego , se encola para salir y esperar a que el vagón le indique que ya termino si recorrido(aquire). Una vez el vagón lo dejó continuar, se termina la visita.

Seudocódigo de Persona

```
persona{
	incrementar colaEntrada;
	pasarAlJuego.aquire();
	if(juegoFuncionando==false){
		marcharse();
	}else{
		disfrutarJuego();
		incrementar colaSalida;
		salirDelJuego.aquire();
	}
	terminarVisita();
}
```

JuegoMecanico

Descripción

Primero, desencola menos personas que el cupo máximo y mas que el cupo mínimo(release). Este número de personas que desencolo lo guarda en una variable. Se le pasa este numero al vagón, para que sepa cuantas personas lleva abordo.

Dentro de un ciclo, el juego mecánico crea vagones mientras que el ultimo vagón creado tenga personas. Si no tiene personas entonces no lo lanza y termina la ejecución.

Seudocódigo de JuegoMecanico

```
juegoMecanico{
	int desencolaPersonas=desencolarersonas();
	for (i=0;i<desencolaPersonas;i++){
		decrementa cola;
		pasarAlJuego.release();
	}
	VagonEstacionado.set(desencolaPersonas);
	while(true){
		if(vagonEstacionado.personasAbordo>0){
			vagonEstacionado.start();
			crearNuevoVagonEstacionado();
			int desencolaPersonas=desencolarersonas();
			for (i=0;i<desencolaPersonas;i++){
				decrementa cola;
				pasarAlJuego.release();
			}
			VagonEstacionado.set(desencolaPersonas);
		else{
			detenerJuego();
			break while;
		}

	}
}
```

Vagon

Descripción

Primero arranca y comienza por ocupar la zona 0(aquire), después va recorriendo todas las zonas ocupándolas (aquire) mientras ocupa una zona nueva, desocupa la anterior (release). Al terminar el recorrido, comienza a dejar irse a las personas (release) y por ultimo, desocupa la ultima zona(release).

Seudocódigo de Vagon

```
vagon{
	arrancar();
	ocuparLugar(0);
	trayectoria[0].aquire();
	for(i=i;i<longitudJuego;i++){
		ocuparLugar(i);
		trayectoria[i].aquire();
		desocuparLugar(i-1);
		trayectoria[i-1].release();
	}
	terminarViaje();
	for(i=0;i<personasAbordo;i++){
		colaSalidaPersonas.release();
	}
	desocuparLugar(longitudJuego-1);
	trayectoria[longitudJuego-1].release();
}
```

Descripción del entorno de desarrollo

Para una experiencia optima, se recomienda ejecutar el archivo con extensión .jar

Lenguaje de programación:

Java 8

Bibliotecas más allá del estándar del lenguaje:

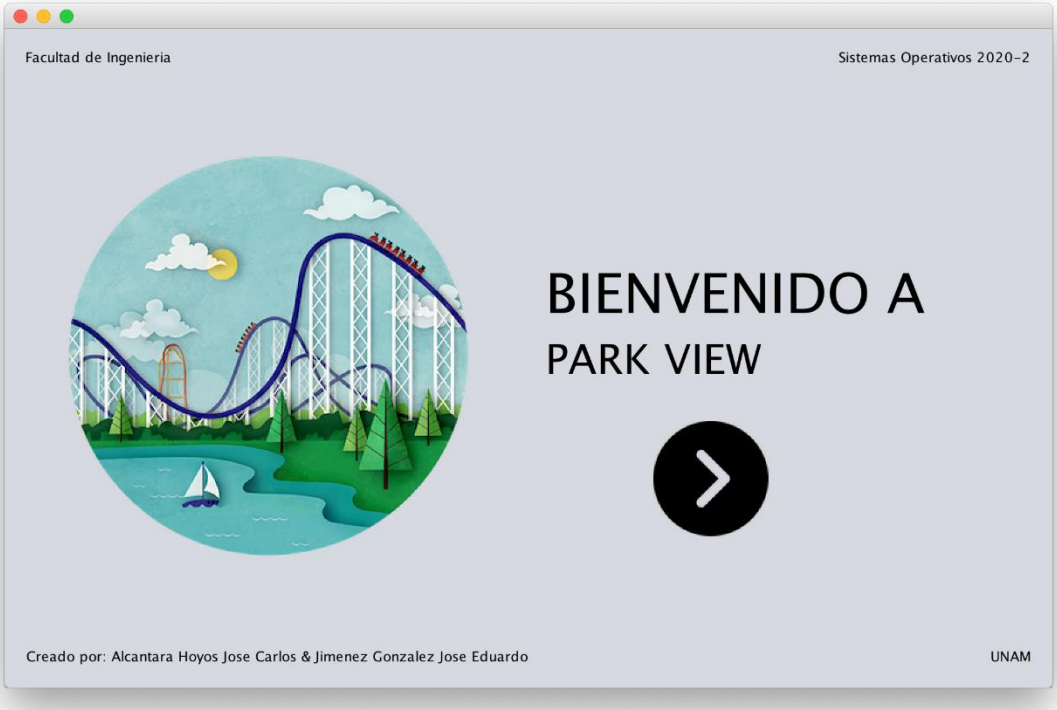
Se utilizaron bibliotecas del API de Java, involucrando las clases contenidas en package javax.swing y las clases java.util.concurrent.Semaphore y java.util.ArrayList.

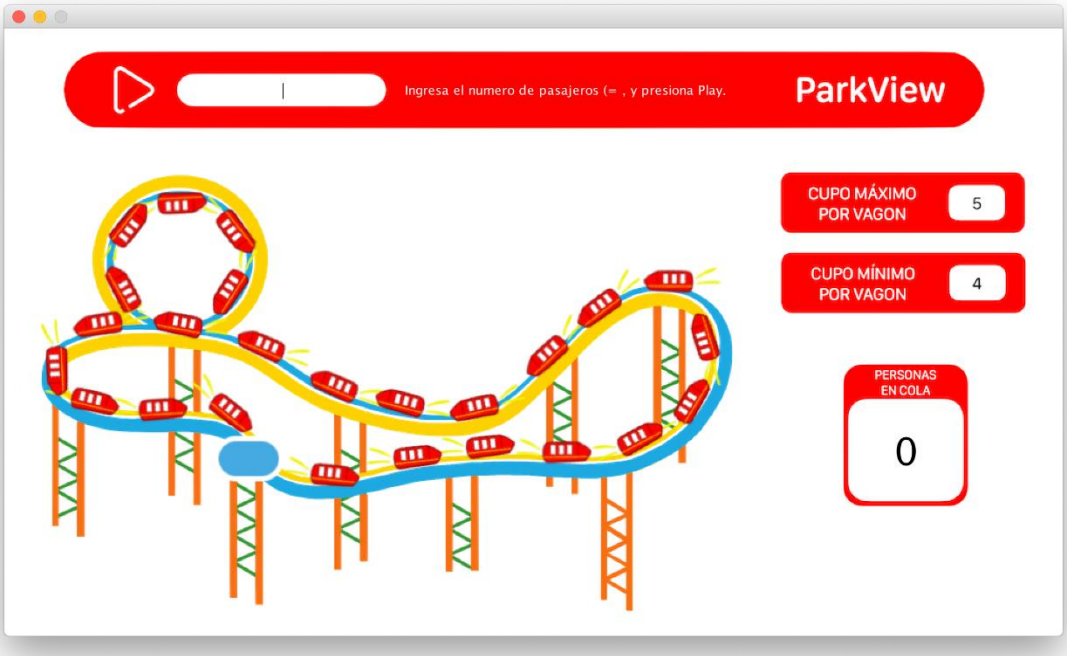
Sistema operativo de pruebas:

- Windows 10
- MacOS Mojave
- Linux/Ubuntu
- Linux/Mint

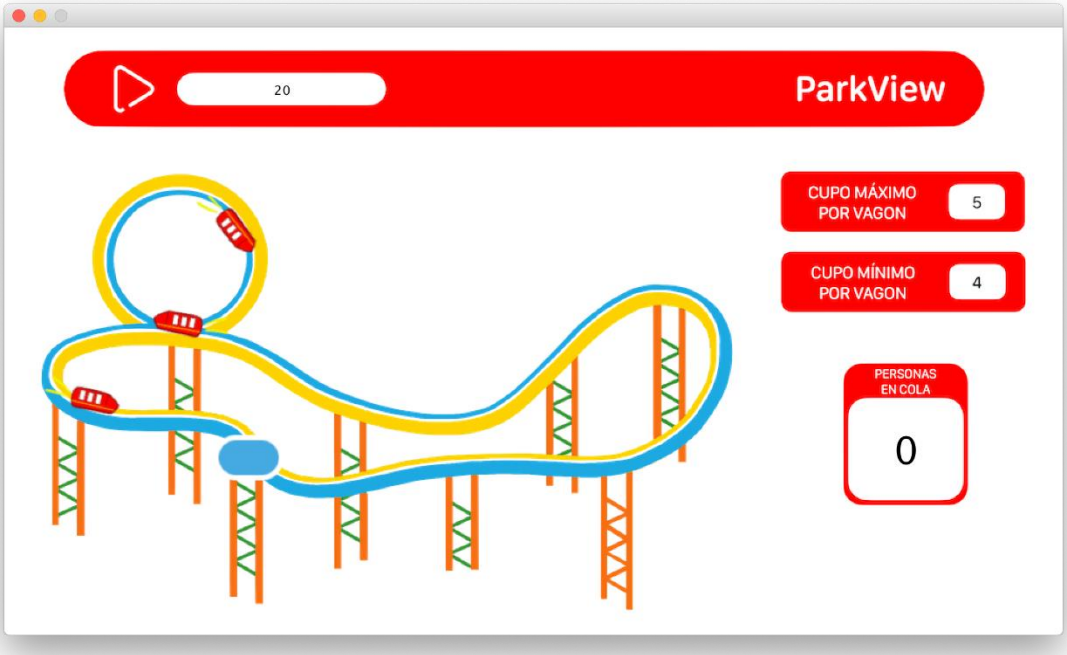
Ejemplos de una ejecución exitosa:

Pantallas de inicio:



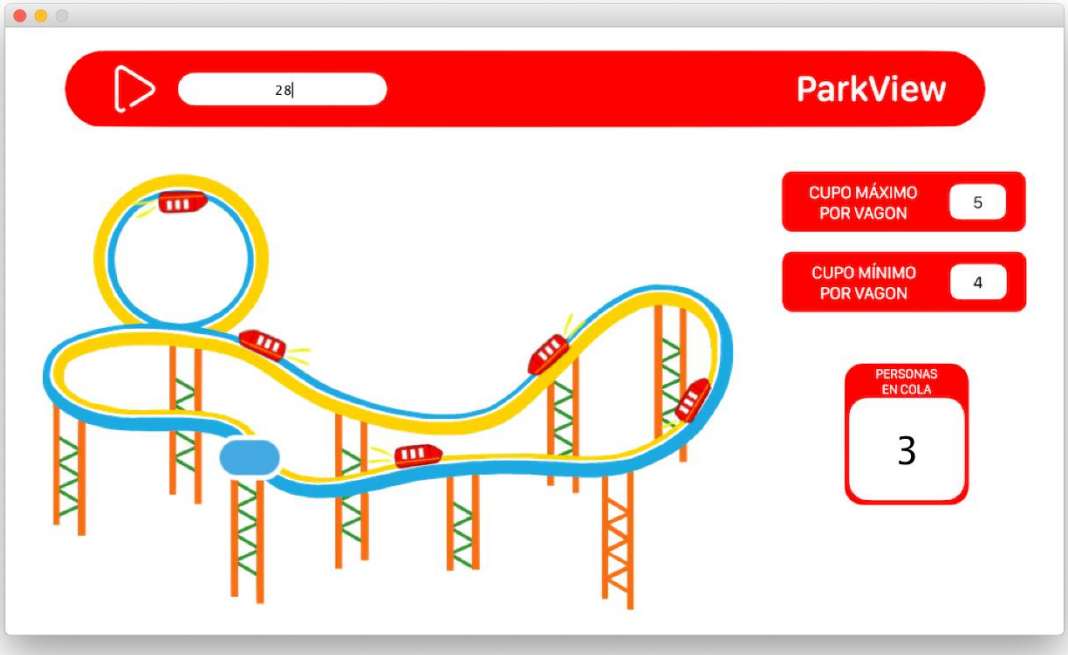


Prueba con 20 personas



Prueba con 28 personas

Esperando más personas para lanzar otro vagon



Personas se salen de la cola y termina la simulación